Ming-Yang Kao
Editor-in-Chief

Encyclopedia of Algorithms

Second Edition

Springer Reference

Encyclopedia of Algorithms

Ming-Yang Kao
Editor

Encyclopedia of Algorithms

Second Edition

With 379 Figures and 51 Tables

Springer Reference

Editor
Ming-Yang Kao
Department of Electrical Engineering
and Computer Science
Northwestern University
Evanston, IL, USA

ISBN 978-1-4939-2863-7 ISBN 978-1-4939-2864-4 (eBook)
ISBN 978-1-4939-2865-1 (print and electronic bundle)
DOI 10.1007/ 978-1-4939-2864-4

Library of Congress Control Number: 2015958521

Printed on acid-free paper

This Springer imprint is published by SpringerNature
The registered company is Springer Science+Business Media LLC New York

Preface

The Encyclopedia of Algorithms provides researchers, students, and practitioners of algorithmic research with a mechanism to efficiently and accurately find the names, definitions, and key results of important algorithmic problems. It also provides further readings on those problems.

This *encyclopedia* covers a broad range of algorithmic areas; each area is summarized by a collection of entries. The entries are written in a clear and concise structure so that they can be readily absorbed by the readers and easily updated by the authors. A typical encyclopedia entry is an in-depth mini-survey of an algorithmic problem written by an expert in the field. The entries for an algorithmic area are compiled by area editors to survey the representative results in that area and can form the core materials of a course in the area.

This 2nd edition of the encyclopedia contains a wide array of important new research results. Highlights include works in tile self-assembly (nanotechnology), bioinformatics, game theory, Internet algorithms, and social networks. Overall, more than 70 % of the entries in this edition and new entries are updated.

This reference work will continue to be updated on a regular basis via a live site to allow timely updates and fast search. Knowledge accumulation is an ongoing community project. Please take ownership of this body of work. If you have feedback regarding a particular entry, please feel free to communicate directly with the author or the area editor of that entry. If you are interested in authoring a future entry, please contact a suitable area editor. If you have suggestions on how to improve the Encyclopedia as a whole, please contact me at kao@northwestern.edu. The credit of this Encyclopedia goes to the area editors, the entry authors, the entry reviewers, and the project editors at Springer, including Melissa Fearon, Michael Hermann, and Sylvia Blago.

About the Editor

Ming-Yang Kao is a Professor of Computer Science in the Department of Electrical Engineering and Computer Science at Northwestern University. He has published extensively in the design, analysis, and applications of algorithms. His current interests include discrete optimization, bioinformatics, computational economics, computational finance, and nanotechnology. He serves as the Editor-in-Chief of Algorithmica.

He obtained a B.S. in Mathematics from National Taiwan University in 1978 and a Ph.D. in Computer Science from Yale University in 1986. He previously taught at Indiana University at Bloomington, Duke University, Yale University, and Tufts University. At Northwestern University, he has served as the Department Chair of Computer Science. He has also cofounded the Program in Computational Biology and Bioinformatics and served as its Director. He currently serves as the Head of the EECS Division of Computing, Algorithms, and Applications and is a Member of the Theoretical Computer Science Group.

For more information, please see www.cs.northwestern.edu/~kao

Area Editors

Algorithm Engineering

Giuseppe F. Italiano* Department of Computer and Systems Science, University of Rome, Rome, Italy

Department of Information and Computer Systems, University of Rome, Rome, Italy

Rajeev Raman* Department of Computer Science, University of Leicester, Leicester, UK

Algorithms for Modern Computers

Alejandro López-Ortiz David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Algorithmic Aspects of Distributed Sensor Networks

Sotiris Nikoletseas Computer Engineering and Informatics Department, University of Patras, Patras, Greece

Computer Technology Institute and Press "Diophantus", Patras, Greece

Approximation Algorithms

Susanne Albers* Technical University of Munich, Munich, Germany

Chandra Chekuri* Department of Computer Science, University of Illinois, Urbana-Champaign, Urbana, IL, USA

Department of Mathematics and Computer Science, The Open University of Israel, Raanana, Israel

Ming-Yang Kao Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA

Sanjeev Khanna* University of Pennsylvania, Philadelphia, PA, USA

Samir Khuller* Computer Science Department, University of Maryland, College Park, MD, USA

---

*Acknowledgment for first edition contribution

Average Case Analysis

Paul (Pavlos) Spirakis* Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece

Computer Science, University of Liverpool, Liverpool, UK

Computer Technology Institute (CTI), Patras, Greece

Bin Packing

Leah Epstein Department of Mathematics, University of Haifa, Haifa, Israel

Bioinformatics

Miklós Csürös Department of Computer Science, University of Montréal, Montréal, QC, Canada

Certified Reconstruction and Mesh Generation

Siu-Wing Cheng Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China

Tamal Krishna Dey Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA

Coding Algorithms

Venkatesan Guruswami* Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA

Combinatorial Group Testing

Ding-Zhu Du Computer Science, University of Minnesota, Minneapolis, MN, USA

Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Combinatorial Optimization

Samir Khuller* Computer Science Department, University of Maryland, College Park, MD, USA

Compressed Text Indexing

Tak-Wah Lam Department of Computer Science, University of Hong Kong, Hong Kong, China

Compression of Text and Data Structures

Gonzalo Navarro Department of Computer Science, University of Chile, Santiago, Chile

Computational Biology

Bhaskar DasGupta Department of Computer Science, University of Illinois, Chicago, IL, USA

Tak-Wah Lam Department of Computer Science, University of Hong Kong, Hong Kong, China

Computational Counting

Xi Chen Computer Science Department, Columbia University, New York, NY, USA

Computer Science and Technology, Tsinghua University, Beijing, China

Computational Economics

Xiaotie Deng AIMS Laboratory (Algorithms-Agents-Data on Internet, Market, and Social Networks), Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China

Department of Computer Science, City University of Hong Kong, Hong Kong, China

Computational Geometry

Sándor Fekete Department of Computer Science, Technical University Braunschweig, Braunschweig, Germany

Computational Learning Theory

Rocco A. Servedio Computer Science, Columbia University, New York, NY, USA

Data Compression

Paolo Ferragina* Department of Computer Science, University of Pisa, Pisa, Italy

Differential Privacy

Aaron Roth Department of Computer and Information Sciences, University of Pennsylvania, Levine Hall, PA, USA

Distributed Algorithms

Sergio Rajsbaum Instituto de Matemáticas, Universidad Nacional Autónoma de México (UNAM) México City, México

Dynamic Graph Algorithms

Giuseppe F. Italiano* Department of Computer and Systems Science, University of Rome, Rome, Italy

Department of Information and Computer Systems, University of Rome, Rome, Italy

Enumeration Algorithms

Takeaki Uno National Institute of Informatics, Chiyoda, Tokyo, Japan

Exact Exponential Algorithms

Fedor V. Fomin Department of Informatics, University of Bergen, Bergen, Norway

External Memory Algorithms

Herman Haverkort Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Game Theory

Mohammad Taghi Hajiaghayi Department of Computer Science, University of Maryland, College Park, MD, USA

Geometric Networks

Andrzej Lingas Department of Computer Science, Lund University, Lund, Sweden

Graph Algorithms

Samir Khuller* Computer Science Department, University of Maryland, College Park, MD, USA

Seth Pettie Electrical Engineering and Computer Science (EECS) Department, University of Michigan, Ann Arbor, MI, USA

Vijaya Ramachandran* Computer Science, University of Texas, Austin, TX, USA

Liam Roditty Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Dimitrios Thilikos AlGCo Project-Team, CNRS, LIRMM, France

Department of Mathematics, National and Kapodistrian University of Athens, Athens, Greece

Graph Drawing

Seokhee Hong School of Information Technologies, University of Sydney, Sydney, NSW, Australia

Internet Algorithms

Edith Cohen Tel Aviv University, Tel Aviv, Israel

Stanford University, Stanford, CA, USA

I/O-Efficient Algorithms

Herman Haverkort Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Kernels and Compressions

Gregory Gutin Department of Computer Science, Royal Holloway, University of London, Egham, UK

Massive Data Algorithms

Herman Haverkort Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Mathematical Optimization

Ding-Zhu Du Computer Science, University of Minnesota, Minneapolis, MN, USA

Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Mechanism Design

Yossi Azar* Tel-Aviv University, Tel Aviv, Israel

Mobile Computing

Xiang-Yang Li* Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA

Modern Learning Theory

Maria-Florina Balcan Department of Machine Learning, Carnegie Mellon University, Pittsburgh, PA, USA

Online Algorithms

Susanne Albers* Technical University of Munich, Munich, Germany

Yossi Azar* Tel-Aviv University, Tel Aviv, Israel

Marek Chrobak Computer Science, University of California, Riverside, CA, USA

Alejandro López-Ortiz David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Parameterized Algorithms

Dimitrios Thilikos AlGCo Project-Team, CNRS, LIRMM, France

Department of Mathematics, National and Kapodistrian University of Athens, Athens, Greece

Parameterized Algorithms and Complexity

Saket Saurabh Institute of Mathematical Sciences, Chennai, India

University of Bergen, Bergen, Norway

Parameterized and Exact Algorithms

Rolf Niedermeier* Department of Mathematics and Computer Science, University of Jena, Jena, Germany

Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Berlin, Germany

Price of Anarchy

Yossi Azar* Tel-Aviv University, Tel Aviv, Israel

Probabilistic Algorithms

Sotiris Nikoletseas Computer Engineering and Informatics Department, University of Patras, Patras, Greece

Computer Technology Institute and Press "Diophantus", Patras, Greece

Paul (Pavlos) Spirakis* Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece

Computer Science, University of Liverpool, Liverpool, UK

Computer Technology Institute (CTI), Patras, Greece

Quantum Computing

Andris Ambainis Faculty of Computing, University of Latvia, Riga, Latvia

Radio Networks

Marek Chrobak Computer Science, University of California, Riverside, CA, USA

Scheduling

Leah Epstein Department of Mathematics, University of Haifa, Haifa, Israel

Scheduling Algorithms

Viswanath Nagarajan University of Michigan, Ann Arbor, MI, USA

Kirk Pruhs* Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

Social Networks

Mohammad Taghi Hajiaghayi Department of Computer Science, University of Maryland, College Park, MD, USA

Grant Schoenebeck Computer Science and Engineering, University of Michigan, Ann Arbor, MI, USA

Stable Marriage Problems, k-SAT Algorithms

Kazuo Iwama Computer Engineering, Kyoto University, Sakyo, Kyoto, Japan

School of Informatics, Kyoto University, Sakyo, Kyoto, Japan

String Algorithms and Data Structures

Paolo Ferragina* Department of Computer Science, University of Pisa, Pisa, Italy

Gonzalo Navarro Department of Computer Science, University of Chile, Santiago, Chile

Steiner Tree Algorithms

Ding-Zhu Du Computer Science, University of Minnesota, Minneapolis, MN, USA

Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Sublinear Algorithms

Andrew McGregor School of Computer Science, University of Massachusetts, Amherst, MA, USA

Sofya Raskhodnikova Computer Science and Engineering Department, Pennsylvania State University, University Park, State College, PA, USA

Tile Self-Assembly

Robert Schweller Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, TX, USA

VLSI CAD Algorithms

Hai Zhou Electrical Engineering and Computer Science (EECS) Department, Northwestern University, Evanston, IL, USA

Contributors

Karen Aardal Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands

Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Ittai Abraham Microsoft Research, Silicon Valley, Palo Alto, CA, USA

Adi Akavia Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, USA

Réka Albert Department of Biology and Department of Physics, Pennsylvania State University, University Park, PA, USA

Mansoor Alicherry Bell Laboratories, Alcatel-Lucent, Murray Hill, NJ, USA

Noga Alon Department of Mathematics and Computer Science, Tel-Aviv University, Tel-Aviv, Israel

Srinivas Aluru Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA

Andris Ambainis Faculty of Computing, University of Latvia, Riga, Latvia

Christoph Ambühl Department of Computer Science, University of Liverpool, Liverpool, UK

Nina Amenta Department of Computer Science, University of California, Davis, CA, USA

Amihood Amir Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

Spyros Angelopoulos Sorbonne Universités, L'Université Pierre et Marie Curie (UPMC), Université Paris 06, Paris, France

Anurag Anshu Center for Quantum Technologies, National University of Singapore, Singapore, Singapore

Alberto Apostolico College of Computing, Georgia Institute of Technology, Atlanta, GA, USA

Vera Asodi Center for the Mathematics of Information, California Institute of Technology, Pasadena, CA, USA

Peter Auer Chair for Information Technology, Montanuniversitaet Leoben, Leoben, Austria

Pranjal Awasthi Department of Computer Science, Princeton University, Princeton, NJ, USA

Department of Electrical Engineering, Indian Institute of Technology Madras, Chennai, Tamilnadu, India

Adnan Aziz Department of Electrical and Computer Engineering, University of Texas, Austin, TX, USA

Moshe Babaioff Microsoft Research, Herzliya, Israel

David A. Bader College of Computing, Georgia Institute of Technology, Atlanta, GA, USA

Michael Bader Department of Informatics, Technical University of Munich, Garching, Germany

Maria-Florina Balcan Department of Machine Learning, Carnegie Mellon University, Pittsburgh, PA, USA

Hideo Bannai Department of Informatics, Kyushu University, Fukuoka, Japan

Nikhil Bansal Eindhoven University of Technology, Eindhoven, The Netherlands

Jérémy Barbay Department of Computer Science (DCC), University of Chile, Santiago, Chile

Sanjoy K. Baruah Department of Computer Science, The University of North Carolina, Chapel Hill, NC, USA

Surender Baswana Department of Computer Science and Engineering, Indian Institute of Technology (IIT), Kanpur, Kanpur, India

MohammadHossein Bateni Google Inc., New York, NY, USA

Luca Becchetti Department of Information and Computer Systems, University of Rome, Rome, Italy

Xiaohui Bei Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore

József Békési Department of Computer Science, Juhász Gyula Teachers Training College, Szeged, Hungary

Djamal Belazzougui Department of Computer Science, Helsinki Institute for Information Technology (HIIT), University of Helsinki, Helsinki, Finland

Aleksandrs Belovs Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA

Aaron Bernstein Department of Computer Science, Columbia University, New York, NY, USA

Vincent Berry Institut de Biologie Computationnelle, Montpellier, France

Randeep Bhatia Bell Laboratories, Alcatel-Lucent, Murray Hill, NJ, USA

Andreas Björklund Department of Computer Science, Lund University, Lund, Sweden

Eric Blais University of Waterloo, Waterloo, ON, Canada

Mathieu Blanchette Department of Computer Science, McGill University, Montreal, QC, Canada

Markus Bläser Department of Computer Science, Saarland University, Saarbrücken, Germany

Avrim Blum School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

Hans L. Bodlaender Department of Computer Science, Utrecht University, Utrecht, The Netherlands

Sergio Boixo Quantum A.I. Laboratory, Google, Venice, CA, USA

Paolo Boldi Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy

Glencora Borradaile Department of Computer Science, Brown University, Providence, RI, USA

School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA

Ulrik Brandes Department of Computer and Information Science, University of Konstanz, Konstanz, Germany

Andreas Brandstädt Computer Science Department, University of Rostock, Rostock, Germany

Department of Informatics, University of Rostock, Rostock, Germany

Gilles Brassard Université de Montréal, Montréal, QC, Canada

Vladimir Braverman Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

Tian-Ming Bu Software Engineering Institute, East China Normal University, Shanghai, China

Adam L. Buchsbaum Madison, NJ, USA

Costas Busch Department of Computer Science, Lousiana State University, Baton Rouge, LA, USA
Jaroslaw Byrka Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands

Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Jin-Yi Cai Beijing University, Beijing, China

Computer Sciences Department, University of Wisconsin–Madison, Madison, WI, USA

Mao-cheng Cai Chinese Academy of Sciences, Institute of Systems Science, Beijing, China

Yang Cai Computer Science, McGill University, Montreal, QC, Canada

Gruia Calinescu Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA

Colin Campbell Department of Physics, Pennsylvania State University, University Park, PA, USA

Luca Castelli Aleardi Laboratoire d'Informatique (LIX), École Polytechnique, Bâtiment Alan Turing, Palaiseau, France

Katarína Cechlárová Faculty of Science, Institute of Mathematics, P. J. Šafárik University, Košice, Slovakia

Nicolò Cesa-Bianchi Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy

Amit Chakrabarti Department of Computer Science, Dartmouth College, Hanover, NH, USA

Deeparnab Chakrabarty Microsoft Research, Bangalore, Karnataka, India

Erin W. Chambers Department of Computer Science and Mathematics, Saint Louis University, St. Louis, MO, USA

Chee Yong Chan National University of Singapore, Singapore, Singapore

Mee Yee Chan Department of Computer Science, University of Hong Kong, Hong Kong, China

Wun-Tat Chan College of International Education, Hong Kong Baptist University, Hong Kong, China

Tushar Deepak Chandra IBM Watson Research Center, Yorktown Heights, NY, USA

Kun-Mao Chao Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

Bernadette Charron-Bost Laboratory for Informatics, The Polytechnic School, Palaiseau, France

Ioannis Chatzigiannakis Department of Computer Engineering and Informatics, University of Patras and Computer Technology Institute, Patras, Greece

Shuchi Chawla Department of Computer Science, University of Wisconsin–Madison, Madison, WI, USA

Shiri Chechik Department of Computer Science, Tel Aviv University, Tel Aviv, Israel

Chandra Chekuri Department of Computer Science, University of Illinois, Urbana-Champaign, Urbana, IL, USA

Department of Mathematics and Computer Science, The Open University of Israel, Raanana, Israel

Danny Z. Chen Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA

Ho-Lin Chen Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan

Jianer Chen Department of Computer Science, Texas A&M University, College Station, TX, USA

Ning Chen Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore

Xi Chen Computer Science Department, Columbia University, New York, NY, USA

Computer Science and Technology, Tsinghua University, Beijing, China

Siu-Wing Cheng Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China

Xiuzhen Cheng Department of Computer Science, George Washington University, Washington, DC, USA

Huang Chien-Chung Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden

Markus Chimani Faculty of Mathematics/Computer, Theoretical Computer Science, Osnabrück University, Osnabrück, Germany

Francis Y.L. Chin Department of Computer Science, University of Hong Kong, Hong Kong, China

Rajesh Chitnis Department of Computer Science, University of Maryland, College Park, MD, USA

Minsik Cho IBM T. J. Watson Research Center, Yorktown Heights, NY, USA

Rezaul A. Chowdhury Department of Computer Sciences, University of Texas, Austin, TX, USA

Stony Brook University (SUNY), Stony Brook, NY, USA

George Christodoulou University of Liverpool, Liverpool, UK

Marek Chrobak Computer Science, University of California, Riverside, CA, USA

Chris Chu Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA

Xiaowen Chu Department of Computer Science, Hong Kong Baptist University, Hong Kong, China

Julia Chuzhoy Toyota Technological Institute, Chicago, IL, USA

Edith Cohen Tel Aviv University, Tel Aviv, Israel

Stanford University, Stanford, CA, USA

Jason Cong Department of Computer Science, UCLA, Los Angeles, CA, USA

Graham Cormode Department of Computer Science, University of Warwick, Coventry, UK

Derek G. Corneil Department of Computer Science, University of Toronto, Toronto, ON, Canada

Bruno Courcelle Laboratoire Bordelais de Recherche en Informatique (LaBRI), CNRS, Bordeaux University, Talence, France

Lenore J. Cowen Department of Computer Science, Tufts University, Medford, MA, USA

Nello Cristianini Department of Engineering Mathematics, and Computer Science, University of Bristol, Bristol, UK

Maxime Crochemore Department of Computer Science, King's College London, London, UK

Laboratory of Computer Science, University of Paris-East, Paris, France

Université de Marne-la-Vallée, Champs-sur-Marne, France

Miklós Csürös Department of Computer Science, University of Montréal, Montréal, QC, Canada

Fabio Cunial Department of Computer Science, Helsinki Institute for Information Technology (HIIT), University of Helsinki, Helsinki, Finland

Marek Cygan Institute of Informatics, University of Warsaw, Warsaw, Poland

Artur Czumaj Department of Computer Science, Centre for Discrete Mathematics and Its Applications, University of Warwick, Coventry, UK

Bhaskar DasGupta Department of Computer Science, University of Illinois, Chicago, IL, USA

Constantinos Daskalakis EECS, Massachusetts Institute of Technology, Cambridge, MA, USA

Mark de Berg Department of Mathematics and Computer Science, TU Eindhoven, Eindhoven, The Netherlands

Xavier Défago School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, Japan

Daniel Delling Microsoft, Silicon Valley, CA, USA

Erik D. Demaine MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA

Camil Demetrescu Department of Computer and Systems Science, University of Rome, Rome, Italy

Department of Information and Computer Systems, University of Rome, Rome, Italy

Ping Deng Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Xiaotie Deng AIMS Laboratory (Algorithms-Agents-Data on Internet, Market, and Social Networks), Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China

Department of Computer Science, City University of Hong Kong, Hong Kong, China

Vamsi Krishna Devabathini Center for Quantum Technologies, National University of Singapore, Singapore, Singapore

Olivier Devillers Inria Nancy – Grand-Est, Villers-lès-Nancy, France

Tamal Krishna Dey Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA

Robert P. Dick Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

Walter Didimo Department of Engineering, University of Perugia, Perugia, Italy

Ling Ding Institute of Technology, University of Washington Tacoma, Tacoma, WA, USA

Yuzheng Ding Xilinx Inc., Longmont, CO, USA

Michael Dom Department of Mathematics and Computer Science, University of Jena, Jena, Germany

Riccardo Dondi Università degli Studi di Bergamo, Bergamo, Italy

Gyorgy Dosa University of Pannonia, Veszprém, Hungary

David Doty Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA, USA

Ding-Zhu Du Computer Science, University of Minnesota, Minneapolis, MN, USA

Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Hongwei Du Department of Computer Science and Technology, Shenzhen Graduate School, Harbin Institute of Technology, Shenzhen, China

Ran Duan Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

Devdatt Dubhashi Department of Computer Science, Chalmers University of Technology, Gothenburg, Sweden

Gothenburg University, Gothenburg, Sweden

Adrian Dumitrescu Computer Science, University of Wisconsin–Milwaukee, Milwaukee, WI, USA

Iréne Durand Laboratoire Bordelais de Recherche en Informatique (LaBRI), CNRS, Bordeaux University, Talence, France

Stephane Durocher University of Manitoba, Winnipeg, MB, Canada

Pavlos Efraimidis Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece

Charilaos Efthymiou Department of Computer Engineering and Informatics, University of Patras, Patras, Greece

Michael Elkin Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Matthias Englert Department of Computer Science, University of Warwick, Coventry, UK

David Eppstein Donald Bren School of Information and Computer Sciences, Computer Science Department, University of California, Irvine, CA, USA

Leah Epstein Department of Mathematics, University of Haifa, Haifa, Israel

Jeff Erickson Department of Computer Science, University of Illinois, Urbana, IL, USA

Constantine G. Evans Division of Biology and Bioengineering, California Institute of Technology, Pasadena, CA, USA

Eyal Even-Dar Google, New York, NY, USA

Rolf Fagerberg Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

Jittat Fakcharoenphol Department of Computer Engineering, Kasetsart University, Bangkok, Thailand

Piotr Faliszewski AGH University of Science and Technology, Krakow, Poland

Lidan Fan Department of Computer Science, The University of Texas, Tyler, TX, USA

Qizhi Fang School of Mathematical Sciences, Ocean University of China, Qingdao, Shandong Province, China

Martín Farach-Colton Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Panagiota Fatourou Department of Computer Science, University of Ioannina, Ioannina, Greece

Jonathan Feldman Google, Inc., New York, NY, USA

Vitaly Feldman IBM Research – Almaden, San Jose, CA, USA

Henning Fernau Fachbereich 4, Abteilung Informatikwissenschaften, Universität Trier, Trier, Germany

Institute for Computer Science, University of Trier, Trier, Germany

Paolo Ferragina Department of Computer Science, University of Pisa, Pisa, Italy

Johannes Fischer Technical University Dortmund, Dortmund, Germany

Nathan Fisher Department of Computer Science, Wayne State University, Detroit, MI, USA

Abraham Flaxman Theory Group, Microsoft Research, Redmond, WA, USA

Paola Flocchini School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON, Canada

Fedor V. Fomin Department of Informatics, University of Bergen, Bergen, Norway

Dimitris Fotakis Department of Information and Communication Systems Engineering, University of the Aegean, Samos, Greece

Kyle Fox Institute for Computational and Experimental Research in Mathematics, Brown University, Providence, RI, USA

Pierre Fraigniaud Laboratoire d'Informatique Algorithmique: Fondements et Applications, CNRS and University Paris Diderot, Paris, France

Fabrizio Frati School of Information Technologies, The University of Sydney, Sydney, NSW, Australia

Engineering Department, Roma Tre University, Rome, Italy

Ophir Frieder Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA

Hiroshi Fujiwara Shinshu University, Nagano, Japan

Stanley P.Y. Fung Department of Computer Science, University of Leicester, Leicester, UK

Stefan Funke Department of Computer Science, Universität Stuttgart, Stuttgart, Germany

Martin Fürer Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

Travis Gagie Department of Computer Science, University of Eastern Piedmont, Alessandria, Italy

Department of Computer Science, University of Helsinki, Helsinki, Finland

Gábor Galambos Department of Computer Science, Juhász Gyula Teachers Training College, Szeged, Hungary

Jianjiong Gao Computational Biology Center, Memorial Sloan-Kettering Cancer Center, New York, NY, USA

Jie Gao Department of Computer Science, Stony Brook University, Stony Brook, NY, USA

Xiaofeng Gao Department of Computer Science, Shanghai Jiao Tong University, Shanghai, China

Juan Garay Bell Laboratories, Murray Hill, NJ, USA

Minos Garofalakis Technical University of Crete, Chania, Greece

Olivier Gascuel Institut de Biologie Computationnelle, Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), CNRS and Université de Montpellier, Montpellier cedex 5, France

Leszek Gąsieniec University of Liverpool, Liverpool, UK

Serge Gaspers Optimisation Research Group, National ICT Australia (NICTA), Sydney, NSW, Australia

School of Computer Science and Engineering, University of New SouthWales (UNSW), Sydney, NSW, Australia

Maciej Gazda Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Raffaele Giancarlo Department of Mathematics and Applications, University of Palermo, Palermo, Italy

Gagan Goel Google Inc., New York, NY, USA

Andrew V. Goldberg Microsoft Research – Silicon Valley, Mountain View, CA, USA

Oded Goldreich Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel

Jens Gramm WSI Institute of Theoretical Computer Science, Tübingen University, Tübingen, Germany

Fabrizio Grandoni IDSIA, USI-SUPSI, University of Lugano, Lugano, Switzerland

Roberto Grossi Dipartimento di Informatica, Università di Pisa, Pisa, Italy

Lov K. Grover Bell Laboratories, Alcatel-Lucent, Murray Hill, NJ, USA

Xianfeng David Gu Department of Computer Science, Stony Brook University, Stony Brook, NY, USA

Joachim Gudmundsson DMiST, National ICT Australia Ltd, Alexandria, Australia

School of Information Technologies, University of Sydney, Sydney, NSW, Australia

Rachid Guerraoui School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland

Heng Guo Computer Sciences Department, University of Wisconsin–Madison, Madison, WI, USA

Jiong Guo Department of Mathematics and Computer Science, University of Jena, Jena, Germany

Manoj Gupta Indian Institute of Technology (IIT) Delhi, Hauz Khas, New Delhi, India

Venkatesan Guruswami Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA

Gregory Gutin Department of Computer Science, Royal Holloway, University of London, Egham, UK

Michel Habib LIAFA, Université Paris Diderot, Paris Cedex 13, France

Mohammad Taghi Hajiaghayi Department of Computer Science, University of Maryland, College Park, MD, USA

Sean Hallgren Department of Computer Science and Engineering, The Pennsylvania State University, University Park, State College, PA, USA

Dan Halperin School of Computer Science, Tel-Aviv University, Tel Aviv, Israel

Moritz Hardt IBM Research – Almaden, San Jose, CA, USA

Ramesh Hariharan Strand Life Sciences, Bangalore, India

Aram W. Harrow Department of Physics, Massachusetts Institute of Technology, Cambridge, MA, USA

Prahladh Harsha Tata Institute of Fundamental Research, Mumbai, Maharashtra, India

Herman Haverkort Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Meng He School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Xin He Department of Computer Science and Engineering, The State University of New York, Buffalo, NY, USA

Lisa Hellerstein Department of Computer Science and Engineering, NYU Polytechnic School of Engineering, Brooklyn, NY, USA

Michael Hemmer Department of Computer Science, TU Braunschweig, Braunschweig, Germany

Danny Hendler Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Monika Henzinger University of Vienna, Vienna, Austria

Maurice Herlihy Department of Computer Science, Brown University, Providence, RI, USA

Ted Herman Department of Computer Science, University of Iowa, Iowa City, IA, USA

John Hershberger Mentor Graphics Corporation, Wilsonville, OR, USA

Timon Hertli Department of Computer Science, ETH Zürich, Zürich, Switzerland

Edward A. Hirsch Laboratory of Mathematical Logic, Steklov Institute of Mathematics, St. Petersburg, Russia

Wing-Kai Hon Department of Computer Science, National Tsing Hua University, Hsin Chu, Taiwan

Seokhee Hong School of Information Technologies, University of Sydney, Sydney, NSW, Australia

Paul G. Howard Akamai Technologies, Cambridge, MA, USA

Peter Høyer University of Calgary, Calgary, AB, Canada

Li-Sha Huang Department of Computer Science and Technology, Tsinghua University, Beijing, China

Yaocun Huang Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Zhiyi Huang Department of Computer Science, The University of Hong Kong, Hong Kong, Hong Kong

Falk Hüffner Department of Math and Computer Science, University of Jena, Jena, Germany

Thore Husfeldt Department of Computer Science, Lund University, Lund, Sweden

Lucian Ilie Department of Computer Science, University of Western Ontario, London, ON, Canada

Sungjin Im Electrical Engineering and Computer Sciences (EECS), University of California, Merced, CA, USA

Csanad Imreh Institute of Informatics, University of Szeged, Szeged, Hungary

Robert W. Irving School of Computing Science, University of Glasgow, Glasgow, UK

Alon Itai Technion, Haifa, Israel

Giuseppe F. Italiano Department of Computer and Systems Science, University of Rome, Rome, Italy

Department of Information and Computer Systems, University of Rome, Rome, Italy

Kazuo Iwama Computer Engineering, Kyoto University, Sakyo, Kyoto, Japan

School of Informatics, Kyoto University, Sakyo, Kyoto, Japan

Jeffrey C. Jackson Department of Mathematics and Computer Science, Duquesne University, Pittsburgh, PA, USA

Ronald Jackups Department of Pediatrics, Washington University, St. Louis, MO, USA

Riko Jacob Institute of Computer Science, Technical University of Munich, Munich, Germany

IT University of Copenhagen, Copenhagen, Denmark

Rahul Jain Department of Computer Science, Center for Quantum Technologies, National University of Singapore, Singapore, Singapore

Klaus Jansen Department of Computer Science, University of Kiel, Kiel, Germany

Jesper Jansson Laboratory of Mathematical Bioinformatics, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto, Japan

Stacey Jeffery David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Madhav Jha Sandia National Laboratories, Livermore, CA, USA

Zenefits, San Francisco, CA, USA

David S. Johnson Department of Computer Science, Columbia University, New York, NY, USA

AT&T Laboratories, Algorithms and Optimization Research Department, Florham Park, NJ, USA

Mark Jones Department of Computer Science, Royal Holloway, University of London, Egham, UK

Tomasz Jurdziński Institute of Computer Science, University of Wrocław, Wrocław, Poland

Yoji Kajitani Department of Information and Media Sciences, The University of Kitakyushu, Kitakyushu, Japan

Shahin Kamali David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Andrew Kane David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Mamadou Moustapha Kanté Clermont-Université, Université Blaise Pascal, LIMOS, CNRS, Aubière, France

Ming-Yang Kao Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA

Alexis Kaporis Department of Information and Communication Systems Engineering, University of the Aegean, Karlovasi, Samos, Greece

George Karakostas Department of Computing and Software, McMaster University, Hamilton, ON, Canada

Juha Kärkkäinen Department of Computer Science, University of Helsinki, Helsinki, Finland

Petteri Kaski Department of Computer Science, School of Science, Aalto University, Helsinki, Finland

Helsinki Institute for Information Technology (HIIT), Helsinki, Finland

Hans Kellerer Department of Statistics and Operations Research, University of Graz, Graz, Austria

Andrew A. Kennings Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada

Kurt Keutzer Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA

Mohammad Reza Khani University of Maryland, College Park, MD, USA

Samir Khuller Computer Science Department, University of Maryland, College Park, MD, USA

Donghyun Kim Department of Mathematics and Physics, North Carolina Central University, Durham, NC, USA

Jin Wook Kim HM Research, Seoul, Korea

Yoo-Ah Kim Computer Science and Engineering Department, University of Connecticut, Storrs, CT, USA

Valerie King Department of Computer Science, University of Victoria, Victoria, BC, Canada

Zoltán Király Department of Computer Science, Eötvös Loránd University, Budapest, Hungary

Egerváry Research Group (MTA-ELTE), Eötvös Loránd University, Budapest, Hungary

Lefteris Kirousis Department of Computer Engineering and Informatics, University of Patras, Patras, Greece

Jyrki Kivinen Department of Computer Science, University of Helsinki, Helsinki, Finland

Masashi Kiyomi International College of Arts and Sciences, Yokohama City University, Yokohama, Kanagawa, Japan

Kim-Manuel Klein University Kiel, Kiel, Germany

Rolf Klein Institute for Computer Science, University of Bonn, Bonn, Germany

Adam Klivans Department of Computer Science, University of Texas, Austin, TX, USA

Koji M. Kobayashi National Institute of Informatics, Chiyoda-ku, Tokyo, Japan

Stephen Kobourov Department of Computer Science, University of Arizona, Tucson, AZ, USA

Kirill Kogan IMDEA Networks, Madrid, Spain

Christian Komusiewicz Institute of Software Engineering and Theoretical Computer Science, Technical University of Berlin, Berlin, Germany

Goran Konjevod Department of Computer Science and Engineering, Arizona State University, Tempe, AZ, USA

Spyros Kontogiannis Department of Computer Science, University of Ioannina, Ioannina, Greece

Matias Korman Graduate School of Information Sciences, Tohoku University, Miyagi, Japan

Guy Kortsarz Department of Computer Science, Rutgers University, Camden, NJ, USA

Nitish Korula Google Research, New York, NY, USA

Robin Kothari Center for Theoretical Physics, Massachusetts Institute of Technology, Cambridge, MA, USA

David R. Cheriton School of Computer Science, Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada

Ioannis Koutis Computer Science Department, University of Puerto Rico-Rio Piedras, San Juan, PR, USA

Dariusz R. Kowalski Department of Computer Science, University of Liverpool, Liverpool, UK

Evangelos Kranakis Department of Computer Science, Carleton, Ottawa, ON, Canada

Dieter Kratsch UFM MIM – LITA, Université de Lorraine, Metz, France

Stefan Kratsch Department of Software Engineering and Theoretical Computer Science, Technical University Berlin, Berlin, Germany

Robert Krauthgamer Weizmann Institute of Science, Rehovot, Israel

IBM Almaden Research Center, San Jose, CA, USA

Stephan Kreutzer Chair for Logic and Semantics, Technical University, Berlin, Germany

Sebastian Krinninger Faculty of Computer Science, University of Vienna, Vienna, Austria

Ravishankar Krishnaswamy Computer Science Department, Princeton University, Princeton, NJ, USA

Danny Krizanc Department of Computer Science, Wesleyan University, Middletown, CT, USA

Piotr Krysta Department of Computer Science, University of Liverpool, Liverpool, UK

Gregory Kucherov CNRS/LIGM, Université Paris-Est, Marne-la-Vallée, France

Fabian Kuhn Department of Computer Science, ETH Zurich, Zurich, Switzerland

V.S. Anil Kumar Virginia Bioinformatics Institute, Virginia Tech, Blacksburg, VA, USA

Tak-Wah Lam Department of Computer Science, University of Hong Kong, Hong Kong, China

Giuseppe Lancia Department of Mathematics and Computer Science, University of Udine, Udine, Italy

Gad M. Landau Department of Computer Science, University of Haifa, Haifa, Israel

Zeph Landau Department of Computer Science, University of California, Berkelely, CA, USA

Michael Langberg Department of Electrical Engineering, The State University of New York, Buffalo, NY, USA

Department of Mathematics and Computer Science, The Open University of Israel, Raanana, Israel

Elmar Langetepe Department of Computer Science, University of Bonn, Bonn, Germany

Ron Lavi Faculty of Industrial Engineering and Management, Technion, Haifa, Israel

Thierry Lecroq Computer Science Department and LITIS Faculty of Science, Université de Rouen, Rouen, France

James R. Lee Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA

Stefano Leonardi Department of Information and Computer Systems, University of Rome, Rome, Italy

Pierre Leone Informatics Department, University of Geneva, Geneva, Switzerland

Henry Leung Department of Computer Science, The University of Hong Kong, Hong Kong, China

Christos Levcopoulos Department of Computer Science, Lund University, Lund, Sweden

Asaf Levin Faculty of Industrial Engineering and Management, The Technion, Haifa, Israel

Moshe Lewenstein Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Li (Erran) Li Bell Laboratories, Alcatel-Lucent, Murray Hill, NJ, USA

Mengling Li Division of Mathematical Sciences, Nanyang Technological University, Singapore, Singapore

Ming Li David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Ming Min Li Computer Science and Technology, Tsinghua University, Beijing, China

Xiang-Yang Li Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA

Vahid Liaghat Department of Computer Science, University of Maryland, College Park, MD, USA

Jie Liang Department of Bioengineering, University of Illinois, Chicago, IL, USA

Andrzej Lingas Department of Computer Science, Lund University, Lund, Sweden

Maarten Löffler Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands

Daniel Lokshtanov Department of Informatics, University of Bergen, Bergen, Norway

Alejandro López-Ortiz David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Chin Lung Lu Institute of Bioinformatics and Department of Biological Science and Technology, National Chiao Tung University, Hsinchu, Taiwan

Pinyan Lu Microsoft Research Asia, Shanghai, China

Zaixin Lu Department of Mathematics and Computer Science, Marywood University, Scranton, PA, USA

Feng Luo Department of Mathematics, Rutgers University, Piscataway, NJ, USA

Haiming Luo Department of Computer Science and Technology, Shenzhen Graduate School, Harbin Institute of Technology, Shenzhen, China

Rune B. Lyngsø Department of Statistics, Oxford University, Oxford, UK

Winton Capital Management, Oxford, UK

Bin Ma David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Department of Computer Science, University of Western Ontario, London, ON, Canada

Mohammad Mahdian Yahoo! Research, Santa Clara, CA, USA

Hamid Mahini Department of Computer Science, University of Maryland, College Park, MD, USA

Veli Mäkinen Department of Computer Science, Helsinki Institute for Information Technology (HIIT), University of Helsinki, Helsinki, Finland

Dahlia Malkhi Microsoft, Silicon Valley Campus, Mountain View, CA, USA

Mark S. Manasse Microsoft Research, Mountain View, CA, USA

David F. Manlove School of Computing Science, University of Glasgow, Glasgow, UK

Giovanni Manzini Department of Computer Science, University of Eastern Piedmont, Alessandria, Italy

Department of Science and Technological Innovation, University of Piemonte Orientale, Alessandria, Italy

Madha V. Marathe IBM T.J. Watson Research Center, Hawthorne, NY, USA

Alberto Marchetti-Spaccamela Department of Information and Computer Systems, University of Rome, Rome, Italy

Igor L. Markov Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

Alexander Matveev Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA

Eric McDermid Cedar Park, TX, USA

Catherine C. McGeoch Department of Mathematics and Computer Science, Amherst College, Amherst, MA, USA

Lyle A. McGeoch Department of Mathematics and Computer Science, Amherst College, Amherst, MA, USA

Andrew McGregor School of Computer Science, University of Massachusetts, Amherst, MA, USA

Brendan D. McKay Department of Computer Science, Australian National University, Canberra, ACT, Australia

Nicole Megow Institut für Mathematik, Technische Universität Berlin, Berlin, Germany

Manor Mendel Department of Mathematics and Computer Science, The Open University of Israel, Raanana, Israel

George B. Mertzios School of Engineering and Computing Sciences, Durham University, Durham, UK

Julián Mestre Department of Computer Science, University of Maryland, College Park, MD, USA

School of Information Technologies, The University of Sydney, Sydney, NSW, Australia

Pierre-Étienne Meunier Le Laboratoire d'Informatique Fondamentale de Marseille (LIF), Aix-Marseille Université, Marseille, France

Ulrich Meyer Department of Computer Science, Goethe University Fankfurt am Main, Frankfurt, Germany

Daniele Micciancio Department of Computer Science, University of California, San Diego, La Jolla, CA, USA

István Miklós Department of Plant Taxonomy and Ecology, Eötvös Loránd University, Budapest, Hungary

Shin-ichi Minato Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan

Vahab S. Mirrokni Theory Group, Microsoft Research, Redmond, WA, USA

Neeldhara Misra Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India

Joseph S.B. Mitchell Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY, USA

Shuichi Miyazaki Academic Center for Computing and Media Studies, Kyoto University, Kyoto, Japan

Alistair Moffat Department of Computing and Information Systems, The University of Melbourne, Melbourne, VIC, Australia

Mark Moir Sun Microsystems Laboratories, Burlington, MA, USA

Ashley Montanaro Department of Computer Science, University of Bristol, Bristol, UK

Tal Mor Department of Computer Science, Technion – Israel Institute of Technology, Haifa, Israel

Michele Mosca Canadian Institute for Advanced Research, Toronto, ON, Canada

Combinatorics and Optimization/Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada

Perimeter Institute for Theoretical Physics, Waterloo, ON, Canada

Thomas Moscibroda Systems and Networking Research Group, Microsoft Research, Redmond, WA, USA

Yoram Moses Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa, Israel

Shay Mozes Efi Arazi School of Computer Science, The Interdisciplinary Center (IDC), Herzliya, Israel

Marcin Mucha Faculty of Mathematics, Informatics and Mechanics, Institute of Informatics, Warsaw, Poland

Priyanka Mukhopadhyay Center for Quantum Technologies, National University of Singapore, Singapore, Singapore

Kamesh Munagala Levine Science Research Center, Duke University, Durham, NC, USA

J. Ian Munro David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Joong Chae Na Department of Computer Science and Engineering, Sejong University, Seoul, Korea

Viswanath Nagarajan University of Michigan, Ann Arbor, MI, USA

Shin-ichi Nakano Department of Computer Science, Gunma University, Kiryu, Japan

Danupon Nanongkai School of Computer Science and Communication, KTH Royal Institute of Technology, Stockholm, Sweden

Giri Narasimhan Department of Computer Science, Florida International University, Miami, FL, USA

School of Computing and Information Sciences, Florida International University, Miami, FL, USA

Gonzalo Navarro Department of Computer Science, University of Chile, Santiago, Chile

Ashwin Nayak Department of Combinatorics and Optimization, and Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada

Amir Nayyeri Department of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA

Jesper Nederlof Technical University of Eindhoven, Eindhoven, The Netherlands

Ofer Neiman Department of Computer Science, Ben-Gurion University of the Negev, Beer Sheva, Israel

Yakov Nekrich David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Jelani Nelson Harvard John A. Paulson School of Engineering and Applied Sciences, Cambridge, MA, USA

Ragnar Nevries Computer Science Department, University of Rostock, Rostock, Germany

Alantha Newman CNRS-Université Grenoble Alpes and G-SCOP, Grenoble, France

Hung Q. Ngo Computer Science and Engineering, The State University of New York, Buffalo, NY, USA

Patrick K. Nicholson Department D1: Algorithms and Complexity, Max Planck Institut für Informatik, Saarbrücken, Germany

Rolf Niedermeier Department of Mathematics and Computer Science, University of Jena, Jena, Germany

Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Berlin, Germany

Sergey I. Nikolenko Laboratory of Mathematical Logic, Steklov Institute of Mathematics, St. Petersburg, Russia

Sotiris Nikoletseas Computer Engineering and Informatics Department, University of Patras, Patras, Greece

Computer Technology Institute and Press "Diophantus", Patras, Greece

Aleksandar Nikolov Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Nikola S. Nikolov Department of Computer Science and Information Systems, University of Limerick, Limerick, Republic of Ireland

Kobbi Nisim Department of Computer Science, Ben-Gurion University, Beer Sheva, Israel

Lhouari Nourine Clermont-Université, Université Blaise Pascal, LIMOS, CNRS, Aubière, France

Yoshio Okamoto Department of Information and Computer Sciences, Toyohashi University of Technology, Toyohashi, Japan

Michael Okun Weizmann Institute of Science, Rehovot, Israel

Rasmus Pagh Theoretical Computer Science, IT University of Copenhagen, Copenhagen, Denmark

David Z. Pan Department of Electrical and Computer Engineering, University of Texas, Austin, TX, USA

Peichen Pan Xilinx, Inc., San Jose, CA, USA

Debmalya Panigrahi Department of Computer Science, Duke University, Durham, NC, USA

Fahad Panolan Institute of Mathematical Sciences, Chennai, India

Vicky Papadopoulou Department of Computer Science, University of Cyprus, Nicosia, Cyprus

Fabio Pardi Institut de Biologie Computationnelle, Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), CNRS and Université de Montpellier, Montpellier cedex 5, France

Kunsoo Park School of Computer Science and Engineering, Seoul National University, Seoul, Korea

Srinivasan Parthasarathy IBM T.J. Watson Research Center, Hawthorne, NY, USA

Apoorva D. Patel Centre for High Energy Physics, Indian Institute of Science, Bangalore, India

Matthew J. Patitz Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR, USA

Mihai Pătraşcu Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology (MIT), Cambridge, MA, USA

Maurizio Patrignani Engineering Department, Roma Tre University, Rome, Italy

Boaz Patt-Shamir Department of Electrical Engineering, Tel-Aviv University, Tel-Aviv, Israel

Ramamohan Paturi Department of Computer Science and Engineering, University of California at San Diego, San Diego, CA, USA

Christophe Paul CNRS, Laboratoire d'Informatique Robotique et Microélectronique de Montpellier, Université Montpellier 2, Montpellier, France

Andrzej Pelc Department of Computer Science, University of Québec-Ottawa, Gatineau, QC, Canada

Jean-Marc Petit Université de Lyon, CNRS, INSA Lyon, LIRIS, Lyon, France

Seth Pettie Electrical Engineering and Computer Science (EECS) Department, University of Michigan, Ann Arbor, MI, USA

Marcin Pilipczuk Institute of Informatics, University of Bergen, Bergen, Norway

Institute of Informatics, University of Warsaw, Warsaw, Poland

Michał Pilipczuk Institute of Informatics, University of Warsaw, Warsaw, Poland

Institute of Informatics, University of Bergen, Bergen, Norway

Yuri Pirola Università degli Studi di Milano-Bicocca, Milan, Italy

Olivier Powell Informatics Department, University of Geneva, Geneva, Switzerland

Amit Prakash Microsoft, MSN, Redmond, WA, USA

Eric Price Department of Computer Science, The University of Texas, Austin, TX, USA

Kirk Pruhs Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

Teresa M. Przytycka Computational Biology Branch, NCBI, NIH, Bethesda, MD, USA

Pavel Pudlák Academy of Science of the Czech Republic, Mathematical Institute, Prague, Czech Republic

Simon J. Puglisi Department of Computer Science, University of Helsinki, Helsinki, Finland

Balaji Raghavachari Computer Science Department, The University of Texas at Dallas, Richardson, TX, USA

Md. Saidur Rahman Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh

Naila Rahman University of Hertfordshire, Hertfordshire, UK

Rajmohan Rajaraman Department of Computer Science, Northeastern University, Boston, MA, USA

Sergio Rajsbaum Instituto de Matemáticas, Universidad Nacional Autónoma de México (UNAM), México City, México

Vijaya Ramachandran Computer Science, University of Texas, Austin, TX, USA

Rajeev Raman Department of Computer Science, University of Leicester, Leicester, UK

M.S. Ramanujan Department of Informatics, University of Bergen, Bergen, Norway

Edgar Ramos School of Mathematics, National University of Colombia, Medellín, Colombia

Satish Rao Department of Computer Science, University of California, Berkeley, CA, USA

Christoforos L. Raptopoulos Computer Science Department, University of Geneva, Geneva, Switzerland

Computer Technology Institute and Press "Diophantus", Patras, Greece

Research Academic Computer Technology Institute, Greece and Computer Engineering and Informatics Department, University of Patras, Patras, Greece

Sofya Raskhodnikova Computer Science and Engineering Department, Pennsylvania State University, University Park, PA, USA

Rajeev Rastogi Amazon, Seattle, WA, USA

Joel Ratsaby Department of Electrical and Electronics Engineering, Ariel University of Samaria, Ariel, Israel

Kaushik Ravindran National Instruments, Berkeley, CA, USA

Michel Raynal Institut Universitaire de France and IRISA, Université de Rennes, Rennes, France

Ben W. Reichardt Electrical Engineering Department, University of Southern California (USC), Los Angeles, CA, USA

Renato Renner Institute for Theoretical Physics, Zurich, Switzerland

Elisa Ricci Department of Electronic and Information Engineering, University of Perugia, Perugia, Italy

Andréa W. Richa School of Computing, Informatics, and Decision Systems Engineering, Ira A. Fulton Schools of Engineering, Arizona State University, Tempe, AZ, USA

Peter C. Richter Department of Combinatorics and Optimization, and Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada

Department of Computer Science, Rutgers, The State University of New Jersey, New Brunswick, NJ, USA

Liam Roditty Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Marcel Roeloffzen Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Martin Roetteler Microsoft Research, Redmond, WA, USA

Heiko Röglin Department of Computer Science, University of Bonn, Bonn, Germany

José Rolim Informatics Department, University of Geneva, Geneva, Switzerland

Dana Ron School of Electrical Engineering, Tel-Aviv University, Ramat-Aviv, Israel

Frances Rosamond Parameterized Complexity Research Unit, University of Newcastle, Callaghan, NSW, Australia

Jarek Rossignac Georgia Institute of Technology, Atlanta, GA, USA

Matthieu Roy Laboratory of Analysis and Architecture of Systems (LAAS), Centre National de la Recherche Scientifique (CNRS), Université Toulouse, Toulouse, France

Ronitt Rubinfeld Massachusetts Institute of Technology (MIT), Cambridge, MA, USA

Tel Aviv University, Tel Aviv-Yafo, Israel

Atri Rudra Department of Computer Science and Engineering, State University of New York, Buffalo, NY, USA

Eric Ruppert Department of Computer Science and Engineering, York University, Toronto, ON, Canada

Frank Ruskey Department of Computer Science, University of Victoria, Victoria, BC, Canada

Luís M.S. Russo Departamento de Informática, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

INESC-ID, Lisboa, Portugal

Wojciech Rytter Institute of Informatics, Warsaw University, Warsaw, Poland

Kunihiko Sadakane Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan

S. Cenk Sahinalp Laboratory for Computational Biology, Simon Fraser University, Burnaby, BC, USA

Michael Saks Department of Mathematics, Rutgers, State University of New Jersey, Piscataway, NJ, USA

Alejandro Salinger Department of Computer Science, Saarland University, Saarbücken, Germany

Sachin S. Sapatnekar Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA

Shubhangi Saraf Department of Mathematics and Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Srinivasa Rao Satti Department of Computer Science and Engineering, Seoul National University, Seoul, South Korea

Saket Saurabh Institute of Mathematical Sciences, Chennai, India

University of Bergen, Bergen, Norway

Guido Schäfer Institute for Mathematics and Computer Science, Technical University of Berlin, Berlin, Germany

Dominik Scheder Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

Institute for Computer Science, Shanghai Jiaotong University, Shanghai, China

Christian Scheideler Department of Computer Science, University of Paderborn, Paderborn, Germany

André Schiper EPFL, Lausanne, Switzerland

Christiane Schmidt The Selim and Rachel Benin School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem, Israel

Markus Schmidt Institute for Computer Science, University of Freiburg, Freiburg, Germany

Dominik Schultes Institute for Computer Science, University of Karlsruhe, Karlsruhe, Germany

Robert Schweller Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, TX, USA

Shinnosuke Seki Department of Computer Science, Helsinki Institute for Information Technology (HIIT), Aalto University, Aalto, Finland

Pranab Sen School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai, India

Sandeep Sen Indian Institute of Technology (IIT) Delhi, Hauz Khas, New Delhi, India

Maria Serna Department of Language and System Information, Technical University of Catalonia, Barcelona, Spain

Rocco A. Servedio Computer Science, Columbia University, New York, NY, USA

Comandur Seshadhri Sandia National Laboratories, Livermore, CA, USA

Department of Computer Science, University of California, Santa Cruz, CA, USA

Jay Sethuraman Industrial Engineering and Operations Research, Columbia University, New York, NY, USA

Jiří Sgall Computer Science Institute, Charles University, Prague, Czech Republic

Rahul Shah Department of Computer Science, Louisiana State University, Baton Rouge, LA, USA

Shai Shalev-Shwartz School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel

Vikram Sharma Department of Computer Science, New York University, New York, NY, USA

Nir Shavit Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA

School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel

Yaoyun Shi Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

Ayumi Shinohara Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Eugene Shragowitz Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA

René A. Sitters Department of Econometrics and Operations Research, VU University, Amsterdam, The Netherlands

Balasubramanian Sivan Microsoft Research, Redmond, WA, USA

Daniel Sleator Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

Michiel Smid School of Computer Science, Carleton University, Ottawa, ON, Canada

Adam Smith Computer Science and Engineering Department, Pennsylvania State University, University Park, State College, PA, USA

Dina Sokol Department of Computer and Information Science, Brooklyn College of CUNY, Brooklyn, NY, USA

Rolando D. Somma Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM, USA

Wen-Zhan Song School of Engineering and Computer Science, Washington State University, Vancouver, WA, USA

Bettina Speckmann Department of Mathematics and Computer Science, Technical University of Eindhoven, Eindhoven, The Netherlands

Paul (Pavlos) Spirakis Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece

Computer Science, University of Liverpool, Liverpool, UK

Computer Technology Institute (CTI), Patras, Greece

Aravind Srinivasan Department of Computer Science, University of Maryland, College Park, MD, USA

Venkatesh Srinivasan Department of Computer Science, University of Victoria, Victoria, BC, Canada

Gerth Stølting Department of Computer Science, University of Aarhus, Århus, Denmark

Jens Stoye Faculty of Technology, Genome Informatics, Bielefeld University, Bielefeld, Germany

Scott M. Summers Department of Computer Science, University of Wisconsin – Oshkosh, Oshkosh, WI, USA

Aries Wei Sun Department of Computer Science, City University of Hong Kong, Hong Kong, China

Vijay Sundararajan Broadcom Corp, Fremont, CA, USA

Wing-Kin Sung Department of Computer Science, National University of Singapore, Singapore, Singapore

Mario Szegedy Department of Combinatorics and Optimization, and Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada

Stefan Szeider Department of Computer Science, Durham University, Durham, UK

Tadao Takaoka Department of Computer Science and Software Engineering, University of Canterbury, Christchurch, New Zealand

Masayuki Takeda Department of Informatics, Kyushu University, Fukuoka, Japan

Kunal Talwar Microsoft Research, Silicon Valley Campus, Mountain View, CA, USA

Christino Tamon Department of Computer Science, Clarkson University, Potsdam, NY, USA

Akihisa Tamura Department of Mathematics, Keio University, Yokohama, Japan

Tiow-Seng Tan School of Computing, National University of Singapore, Singapore, Singapore

Shin-ichi Tanigawa Research Institute for Mathematical Sciences (RIMS), Kyoto University, Kyoto, Japan

Eric Tannier LBBE Biometry and Evolutionary Biology, INRIA Grenoble Rhône-Alpes, University of Lyon, Lyon, France

Alain Tapp Université de Montréal, Montréal, QC, Canada

Stephen R. Tate Department of Computer Science, University of North Carolina, Greensboro, NC, USA

Gadi Taubenfeld Department of Computer Science, Interdiciplinary Center Herzlia, Herzliya, Israel

Kavitha Telikepalli CSA Department, Indian Institute of Science, Bangalore, India

Barbara M. Terhal JARA Institute for Quantum Information, RWTH Aachen University, Aachen, Germany

Alexandre Termier IRISA, University of Rennes, 1, Rennes, France

My T. Thai Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, USA

Abhradeep Thakurta Department of Computer Science, Stanford University, Stanford, CA, USA

Microsoft Research, CA, USA

Justin Thaler Yahoo! Labs, New York, NY, USA

Sharma V. Thankachan School of CSE, Georgia Institute of Technology, Atlanta, USA

Dimitrios Thilikos AlGCo Project-Team, CNRS, LIRMM, France

Department of Mathematics, National and Kapodistrian University of Athens, Athens, Greece

Haitong Tian Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA

Ioan Todinca INSA Centre Val de Loire, Universite d'Orleans, Orléans, France

Alade O. Tokuta Department of Mathematics and Physics, North Carolina Central University, Durham, NC, USA

Laura Toma Department of Computer Science, Bowdoin College, Brunswick, ME, USA

Etsuji Tomita The Advanced Algorithms Research Laboratory, The University of Electro-Communications, Chofu, Tokyo, Japan

Csaba D. Tóth Department of Computer Science, Tufts University, Medford, MA, USA

Department of Mathematics, California State University Northridge, Los Angeles, CA, USA

Luca Trevisan Department of Computer Science, University of California, Berkeley, CA, USA

John Tromp CWI, Amsterdam, The Netherlands

Nicolas Trotignon Laboratoire de l'Informatique du Parallélisme (LIP), CNRS, ENS de Lyon, Lyon, France

Jakub Truszkowski Cancer Research UK Cambridge Institute, University of Cambridge, Cambridge, UK

European Molecular Biology Laboratory, European Bioinformatics Institute (EMBL-EBI), Wellcome Trust Genome Campus, Hinxton, Cambridge, UK

Esko Ukkonen Department of Computer Science, Helsinki Institute for Information Technology (HIIT), University of Helsinki, Helsinki, Finland

Jonathan Ullman Department of Computer Science, Columbia University, New York, NY, USA

Takeaki Uno National Institute of Informatics, Chiyoda, Tokyo, Japan

Ruth Urner Department of Machine Learning, Carnegie Mellon University, Pittsburgh, USA

Jan Vahrenhold Department of Computer Science, Westfälische Wilhelms-Universität Münster, Münster, Germany

Daniel Valenzuela Department of Computer Science, Helsinki Institute for Information Technology (HIIT), University of Helsinki, Helsinki, Finland

Marc van Kreveld Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands

Rob van Stee University of Leicester, Leicester, UK

Stefano Varricchio Department of Computer Science, University of Roma, Rome, Italy

José Verschae Departamento de Matemáticas and Departamento de Ingeniería Industrial y de Sistemas, Pontificia Universidad Católica de Chile, Santiago, Chile

Stéphane Vialette IGM-LabInfo, University of Paris-East, Descartes, France

Sebastiano Vigna Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy

Yngve Villanger Department of Informatics, University of Bergen, Bergen, Norway

Paul Vitányi Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands

Jeffrey Scott Vitter University of Kansas, Lawrence, KS, USA

Berthold Vöcking Department of Computer Science, RWTH Aachen University, Aachen, Germany

Tjark Vredeveld Department of Quantitative Economics, Maastricht University, Maastricht, The Netherlands

Magnus Wahlström Department of Computer Science, Royal Holloway, University of London, Egham, UK

Peng-Jun Wan Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA

Chengwen Chris Wang Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

Feng Wang Mathematical Science and Applied Computing, Arizona State University at the West Campus, Phoenix, AZ, USA

Huijuan Wang Shandong University, Jinan, China

Joshua R. Wang Department of Computer Science, Stanford University, Stanford, CA, USA

Lusheng Wang Department of Computer Science, City University of Hong Kong, Hong Kong, Hong Kong

Wei Wang School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an, Shaanxi, China

Weizhao Wang Google Inc., Irvine, CA, USA

Yu Wang Department of Computer Science, University of North Carolina, Charlotte, NC, USA

Takashi Washio The Institute of Scientific and Industrial Research, Osaka University, Ibaraki, Osaka, Japan

Matthew Weinberg Computer Science, Princeton University, Princeton, NJ, USA

Tobias Weinzierl School of Engineering and Computing Sciences, Durham University, Durham, UK

Renato F. Werneck Microsoft Research Silicon Valley, La Avenida, CA, USA

Matthias Westermann Department of Computer Science, TU Dortmund University, Dortmund, Germany

Tim A.C. Willemse Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Ryan Williams Department of Computer Science, Stanford University, Stanford, CA, USA

Tyson Williams Computer Sciences Department, University of Wisconsin–Madison, Madison, WI, USA

Andrew Winslow Department of Computer Science, Tufts University, Medford, MA, USA

Paul Wollan Department of Computer Science, University of Rome La Sapienza, Rome, Italy

Martin D.F. Wong Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA

Prudence W.H. Wong University of Liverpool, Liverpool, UK

David R. Wood School of Mathematical Sciences, Monash University, Melbourne, VIC, Australia

Damien Woods Computer Science, California Institute of Technology, Pasadena, CA, USA

Lidong Wu Department of Computer Science, The University of Texas, Tyler, TX, USA

Weili Wu College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi Province, China

Department of Computer Science, California State University, Los Angeles, CA, USA

Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Christian Wulff-Nilsen Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

Mingji Xia The State Key Laboratory of Computer Science, Chinese Academy of Sciences, Beijing, China

David Xiao CNRS, Université Paris 7, Paris, France

Dong Xu Bond Life Sciences Center, University of Missouri, Columbia, MO, USA

Wen Xu Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Katsuhisa Yamanaka Department of Electrical Engineering and Computer Science, Iwate University, Iwate, Japan

Hiroki Yanagisawa IBM Research – Tokyo, Tokyo, Japan

Honghua Hannah Yang Strategic CAD Laboratories, Intel Corporation, Hillsboro, OR, USA

Qiuming Yao University of Missouri, Columbia, MO, USA

Chee K. Yap Department of Computer Science, New York University, New York, NY, USA

Yinyu Ye Department of Management Science and Engineering, Stanford University, Stanford, CA, USA

Anders Yeo Engineering Systems and Design, Singapore University of Technology and Design, Singapore, Singapore

Department of Mathematics, University of Johannesburg, Auckland Park, South Africa

Chih-Wei Yi Department of Computer Science, National Chiao Tung University, Hsinchu City, Taiwan

Ke Yi Hong Kong University of Science and Technology, Hong Kong, China

Yitong Yin Nanjing University, Jiangsu, Nanjing, Gulou, China

S.M. Yiu Department of Computer Science, University of Hong Kong, Hong Kong, China

Makoto Yokoo Department of Information Science and Electrical Engineering, Kyushu University, Nishi-ku, Fukuoka, Japan

Evangeline F.Y. Young Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China

Neal E. Young Department of Computer Science and Engineering, University of California, Riverside, CA, USA

Bei Yu Department of Electrical and Computer Engineering, University of Texas, Austin, TX, USA

Yaoliang Yu Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA, USA

Raphael Yuster Department of Mathematics, University of Haifa, Haifa, Israel

Morteza Zadimoghaddam Google Research, New York, NY, USA

Francis Zane Lucent Technologies, Bell Laboraties, Murray Hill, NJ, USA

Christos Zaroliagis Department of Computer Engineering and Informatics, University of Patras, Patras, Greece

Norbert Zeh Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada

Li Zhang Microsoft Research, Mountain View, CA, USA

Louxin Zhang Department of Mathematics, National University of Singapore, Singapore, Singapore

Shengyu Zhang The Chinese University of Hong Kong, Hong Kong, China

Zhang Zhao College of Mathematics Physics and Information Engineering, Zhejiang Normal University, Zhejiang, Jinhua, China

Hai Zhou Electrical Engineering and Computer Science (EECS) Department, Northwestern University, Evanston, IL, USA

Yuqing Zhu Department of Computer Science, California State University, Los Angeles, CA, USA

Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Sandra Zilles Department of Computer Science, University of Regina, Regina, SK, Canada

Aaron Zollinger Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA

Uri Zwick Department of Mathematics and Computer Science, Tel-Aviv University, Tel-Aviv, Israel

A

Abelian Hidden Subgroup Problem

Michele Mosca
Canadian Institute for Advanced Research, Toronto, ON, Canada
Combinatorics and Optimization/Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada
Perimeter Institute for Theoretical Physics, Waterloo, ON, Canada

Keywords

Abelian hidden subgroup problem; Abelian stabilizer problem; Quantum algorithms; Quantum complexity; Quantum computing

Years and Authors of Summarized Original Work

1995; Kitaev
2008; Mosca

Problem Definition

The Abelian hidden subgroup problem is the problem of finding generators for a subgroup K of an Abelian group G, where this subgroup is defined implicitly by a function $f : G \to X$, for some finite set X. In particular, f has the property that $f(v) = f(w)$ if and only if the cosets (we are assuming additive notation for the group operation here.) $v+K$ and $w+K$ are equal. In other words, f is constant on the cosets of the subgroup K and distinct on each coset.

It is assumed that the group G is finitely generated and that the elements of G and X have unique binary encodings. The binary assumption is only for convenience, but it is important to have unique encodings (e.g., in [22] Watrous uses a quantum state as the unique encoding of group elements). When using variables g and h (possibly with subscripts), multiplicative notation is used for the group operations. Variables x and y (possibly with subscripts) will denote integers with addition. The boldface versions $\mathbf{x}$ and $\mathbf{y}$ will denote *tuples* of integers or binary strings.

By assumption, there is computational means of computing the function f, typically a circuit or "black box" that maps the encoding of a value g to the encoding of $f(g)$. The theory of reversible computation implies that one can turn a circuit for computing $f(g)$ into a reversible circuit for computing $f(g)$ with a modest increase in the size of the circuit. Thus, it will be assumed that there is a reversible circuit or black box that maps $(g, \mathbf{z}) \mapsto (g, \mathbf{z} \oplus f(g))$, where $\oplus$ denotes the bit-wise XOR (sum modulo 2), and $\mathbf{z}$ is any binary string of the same length as the encoding of $f(g)$.

Quantum mechanics implies that any reversible gate can be extended linearly to a unitary operation that can be implemented in the model of quantum computation. Thus, it is assumed that there is a quantum circuit or

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

black box that implements the unitary map $U_f : |g\rangle|\mathbf{z}\rangle \mapsto |g\rangle|\mathbf{z} \oplus f(g)\rangle$.

Although special cases of this problem have been considered in classical computer science, the general formulation as the hidden subgroup problem seems to have appeared in the context of quantum computing, since it neatly encapsulates a family of black-box problems for which quantum algorithms offer an exponential speedup (in terms of query complexity) over classical algorithms. For some explicit problems (i.e., where the black box is replaced with a specific function, such as exponentiation modulo N), there is a conjectured exponential speedup.

Abelian Hidden Subgroup Problem:

Input: Elements $g_1, g_2, \ldots, g_n \in G$ that generate the Abelian group G. A black box that implements $U_f : |m_1, m_2, \ldots, m_n\rangle|\mathbf{y}\rangle \mapsto |m_1, m_2, \ldots, m_n\rangle|f(g) \oplus \mathbf{y}\rangle$ where $g = g_1^{m_1} g_2^{m_2} \ldots g_n^{m_n}$ and K is the hidden subgroup corresponding to f.

Output: Elements $h_1, h_2, \ldots, h_l \in G$ that generate K.

Here we use multiplicative notation for the group G in order to be consistent with Kitaev's formulation of the Abelian stabilizer problem. Most of the applications of interest typically use additive notation for the group G.

It is hard to trace the precise origin of this general formulation of the problem, which simultaneously generalizes "Simon's problem" [20], the order-finding problem (which is the quantum part of the quantum factoring algorithm [18]), and the discrete logarithm problem.

One of the earliest generalizations of Simon's problem, order-finding problem, and discrete logarithm problem, which captures the essence of the Abelian hidden subgroup problem, is the *Abelian stabilizer problem* which was solved by Kitaev using a quantum algorithm in his 1995 paper [14] (and also appears in [15, 16]).

Let G be a group acting on a finite set X. That is, each element of G acts as a map from X to X in such a way that for any two elements $g, h \in G$, $g(h(z)) = (gh)(z)$ for all $z \in X$. For a particular element $z \in X$, the set of elements that fix z (i.e., the elements $g \in G$ such that $g(z) = z$) form a subgroup. This subgroup is called the stabilizer of z in G, denoted $St_G(z)$.

Abelian Stabilizer Problem:

Input: Elements $g_1, g_2, \ldots, g_n \in G$ that generate the group G. An element $z \in X$. A black box that implements $U_{(G,X)} : |m_1, m_2, \ldots, m_n\rangle|z\rangle \mapsto |m_1, m_2, \ldots, m_n\rangle|g(z)\rangle$ where $g = g_1^{m_1} g_2^{m_2} \ldots g_n^{m_n}$.

Output: Elements $h_1, h_2, \ldots, h_l \in G$ that generate $St_G(z)$.

Let f_z denote the function from G to X that maps $g \in G$ to $g(z)$. One can implement U_{f_z} using $U_{(G,X)}$. The hidden subgroup corresponding to f_z is $St_G(z)$. Thus, the Abelian stabilizer problem is a special case of the Abelian hidden subgroup problem.

One of the subtle differences (discussed in Appendix 6 of [12]) between the above formulation of the Abelian stabilizer problem and the Abelian hidden subgroup problem is that Kitaev's formulation gives a black box that for any $g, h \in G$ maps $|m_1, \ldots, m_n\rangle|f_z(h)\rangle \mapsto |m_1, \ldots, m_n\rangle|f_z(hg)\rangle$, where $g = g_1^{m_1} g_2^{m_2} \ldots g_n^{m_n}$. The algorithm given by Kitaev is essentially one that estimates eigenvalues of shift operations of the form $|f_z(h)\rangle \mapsto |f_z(hg)\rangle$. In general, these shift operators are not explicitly needed, and it suffices to be able to compute a map of the form $|\mathbf{y}\rangle \mapsto |f_z(h) \oplus \mathbf{y}\rangle$ for any binary string $\mathbf{y}$.

Generalizations of this form have been known since shortly after Shor presented his factoring and discrete logarithm algorithms (e.g., [23] presents the hidden subgroup problem for a large class of finite Abelian groups or more generally in [11] for finite Abelian groups presented as a product of finite cyclic groups. In [17] the natural Abelian hidden subgroup algorithm is related to eigenvalue estimation.)

Other problems which can be formulated in this way include:

Deutsch's Problem:

Input: A black box that implements $U_f : |x\rangle|b\rangle \mapsto |x\rangle|b \oplus f(x)\rangle$, for some function f that maps $\mathbb{Z}_2 = \{0, 1\}$ to $\{0, 1\}$.

Output: "constant" if $f(0) = f(1)$ and "balanced" if $f(0) \neq f(1)$.

Note that $f(x) = f(y)$ if and only if $x - y \in K$, where K is either $\{0\}$ or $\mathbb{Z}_2 = \{0, 1\}$. If $K = \{0\}$, then f is $1-1$ or "balanced," and if $K = \mathbb{Z}_2$, then f is constant [5,6].

Simon's Problem:

Input: A black box that implements $U_f : |\mathbf{x}\rangle|\mathbf{b}\rangle \mapsto |\mathbf{x}\rangle|\mathbf{b} \oplus f(\mathbf{x})\rangle$ for some function f from $\mathbb{Z}_2^n$ to some set X (which is assumed to consist of binary strings of some fixed length) with the property that $f(\mathbf{x}) = f(\mathbf{y})$ if and only if $\mathbf{x} - \mathbf{y} \in K = \{\mathbf{0}, \mathbf{s}\}$ for some $\mathbf{s} \in \mathbb{Z}_2^n$.
Output: The "hidden" string $\mathbf{s}$.

The decision version allows $K = \{\mathbf{0}\}$ and asks whether K is trivial. Simon [20] presents an efficient algorithm for solving this problem and an exponential lower bound on the query complexity. The solution to the Abelian hidden subgroup problem is a generalization of Simon's algorithm.

Key Results

Theorem (ASP) There exists a quantum algorithm that, given an instance of the Abelian stabilizer problem, makes $n + O(1)$ queries to $U_{(G,X)}$ and uses poly(n) other elementary quantum and classical operations, with probability at least $\frac{2}{3}$ output values $h_1, h_2, \ldots, h_l$ such that $St_G(z) = \langle h_1\rangle \oplus \langle h_2\rangle \oplus \cdots \langle h_l\rangle$.

Kitaev first solved this problem (with a slightly higher query complexity, because his eigenvalue estimation procedure was not optimal). An eigenvalue estimation procedure based on the Quantum Fourier Transform achieves the $n + O(1)$ query complexity [5].

Theorem (AHSP) There exists a quantum algorithm that, given an instance of the Abelian hidden subgroup problem, makes $n+O(1)$ queries to U_f and uses poly(n) other elementary quantum and classical operations, with probability at least $\frac{2}{3}$ output values $h_1, h_2, \ldots, h_l$ such that $K = \langle h_1\rangle \oplus \langle h_2\rangle \oplus \cdots \langle h_l\rangle$.

In some cases, the success probability can be made 1 with the same complexity, and in general the success probability can be made $1 - \epsilon$ using $n + O(\log(1/\epsilon))$ queries and poly$(n, \log(1/\epsilon))$ other elementary quantum and classical operations.

Applications

Most of these applications in fact were known before the Abelian stabilizer problem or hidden subgroup problem were formulated.

Finding the order of an element in a group: Let a be an element of a group H (which does *not* need to be Abelian), and let r be the smallest positive integer so that $a^r = 1$.

Consider the function f from $G = \mathbb{Z}$ to the group H where $f(x) = a^x$ for some element a of H. Then $f(x) = f(y)$ if and only if $x - y \in r\mathbb{Z}$. The hidden subgroup is $K = r\mathbb{Z}$ and a generator for K gives the order r of a.

Finding the period of a periodic function: Consider a function f from $G = \mathbb{Z}$ to a set X with the property that for some positive integer r, we have $f(x) = f(y)$ if and only if $x - y \in r\mathbb{Z}$. The hidden subgroup of f is $K = r\mathbb{Z}$ and a generator for K gives the period r.

Order finding is a special case of period finding and was also solved by Shor's algorithm [18].

Discrete Logarithms: Let a be an element of a group H (which does *not* need to be Abelian), with $a^r = 1$, and suppose $b = a^k$ from some unknown k. The integer k is called the *discrete logarithm of b to the base a*. Consider the function f from $G = \mathbb{Z}_r \times \mathbb{Z}_r$ to H satisfying $f(x_1, x_2) = a^{x_1} b^{x_2}$. Then $f(x_1, x_2) = f(y_1, y_2)$ if and only if $(x_1, x_2) - (y_1, y_2) \in \{(t, -tk), t = 0, 1, \ldots, r - 1\}$ which is the subgroup $\langle(1, -k)\rangle$ of $\mathbb{Z}_r \times \mathbb{Z}_r$. Thus, finding a generator for the hidden subgroup K will give

the discrete logarithm k. Note that this algorithm works for H equal to the multiplicative group of a finite field, or the additive group of points on an elliptic curve, which are groups that are used in public-key cryptography.

Recently, Childs and Ivanyos [3] presented an efficient quantum algorithm for finding **discrete logarithms in semigroups**. Their algorithm makes use of the quantum algorithms for period finding and discrete logarithms as subroutines.

Hidden Linear Functions: Let σ be some permutation of $\mathbb{Z}_N$ for some integer N. Let h be a function from $G = \mathbb{Z} \times \mathbb{Z}$ to $\mathbb{Z}_N$, $h(x, y) = x + ay \mod N$. Let $f = \sigma \circ h$. The hidden subgroup of f is $\langle(-a, 1)\rangle$. Boneh and Lipton [1] showed that even if the linear structure of h is hidden (by σ), one can efficiently recover the parameter a with a quantum algorithm.

Self-Shift-Equivalent Polynomials: Given a polynomial P in l variables $X_1, X_2, \ldots, X_l$ over $\mathbb{F}_q$, the function f that maps $(a_1, a_2, \ldots, a_l) \in \mathbb{F}_q^l$ to $P(X_1 - a_1, X_2 - a_2, \ldots, X_l - a_l)$ is constant on cosets of a subgroup K of $\mathbb{F}_q^l$. This subgroup K is the set of shift-self-equivalences of the polynomial P. Grigoriev [10] showed how to compute this subgroup.

Decomposition of a Finitely Generated Abelian Group: Let G be a group with a unique binary representation for each element of G, and assume that the group operation, and recognizing if a binary string represents an element of G or not, can be done efficiently.

Given a set of generators $g_1, g_2, \ldots, g_n$ for a group G, output a set of elements $h_1, h_2, \ldots, h_l$, $l \leq n$, from the group G such that $G = \langle g_1\rangle \oplus \langle g_2\rangle \oplus \ldots \oplus \langle g_l\rangle$. Such a generating set can be found efficiently [2] from generators of the hidden subgroup of the function that maps $(m_1, m_2, \ldots, m_n) \mapsto g_1^{m_1} g_2^{m_2} \ldots g_n^{m_n}$.

This simple algorithm directly leads to an algorithm for computing the class group and class number of a quadratic number field, as pointed out by Watrous [22] in his paper that shows how to compute the order of solvable groups. Computing the class group of a more general number field is a much more difficult task: this and related problems have been successfully tackled in a series of elegant work summarized in ▸ Quantum Algorithms for Class Group of a Number Field.

Such a decomposition of Abelian groups was also applied by Friedl, Ivanyos, and Santha [9] to test if a finite set with a binary operation is an Abelian group, by Kedlaya [13] to compute the zeta function of a genus g curve over a finite field $\mathbb{F}_q$ in time polynomial in g and q, and by Childs, Jao, and Soukharev [4] in order to construct elliptic curve isogenies in subexponential time.

Discussion: What About Non-Abelian Groups?

The great success of quantum algorithms for solving the Abelian hidden subgroup problem leads to the natural question of whether it can solve the hidden subgroup problem for non-Abelian groups. It has been shown that a polynomial number of queries suffice [8]; however, in general there is no bound on the overall computational complexity (which includes other elementary quantum or classical operations).

This question has been studied by many researchers, and efficient quantum algorithms can be found for some non-Abelian groups. However, at present, there is no efficient algorithm for most non-Abelian groups. For example, solving the HSP for the symmetric group would directly solve the graph automorphism problem.

Cross-References

- ▸ Quantum Algorithm for Factoring
- ▸ Quantum Algorithm for Solving Pell's Equation
- ▸ Quantum Algorithms for Class Group of a Number Field

Recommended Reading

1. Boneh D, Lipton R (1995) Quantum cryptanalysis of hidden linear functions (extended abstract). In: Proceedings of 15th Annual International Cryptology Conference (CRYPTO'95), Santa Barbara, pp 424–437

2. Cheung K, Mosca M (2001) Decomposing finite Abelian groups. Quantum Inf Comput 1(2):26–32
3. Childs AM, Ivanyos G (2014) Quantum computation of discrete logarithms in semigroups. J Math Cryptol 8(4):405–416
4. Childs AM, Jao D, Soukharev V (2010) Constructing elliptic curve isogenies in quantum subexponential time. preprint. arXiv:1012.4019
5. Cleve R, Ekert A, Macchiavello C, Mosca M (1998) Quantum algorithms revisited. Proc R Soc Lond A 454:339–354
6. Deutsch D (1985) Quantum theory, the Church-Turing principle and the universal quantum computer. Proc R Soc Lond A 400:97–117
7. Deutsch D, Jozsa R (1992) Rapid solutions of problems by quantum computation. Proc R Soc Lond Ser A 439:553–558
8. Ettinger M, Høyer P, Knill E (2004) The quantum query complexity of the hidden subgroup problem is polynomial. Inf Process Lett 91:43–48
9. Friedl K, Ivanyos G, Santha M (2005) Efficient testing of groups. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC'05), Baltimore, pp 157–166
10. Grigoriev D (1997) Testing shift-equivalence of polynomials by deterministic, probabilistic and quantum machines. Theor Comput Sci 180: 217–228
11. Høyer P (1999) Conjugated operators in quantum algorithms. Phys Rev A 59(5):3280–3289
12. Kaye P, Laflamme R, Mosca M (2007) An introduction to quantum computation. Oxford University Press, Oxford, UK
13. Kedlaya KS (2006) Quantum computation of zeta functions of curves. Comput Complex 15:1–19
14. Kitaev A (1995) Quantum measurements and the Abelian stabilizer problem. quant-ph/9511026
15. Kitaev A (1996) Quantum measurements and the Abelian stabilizer problem. In: Electronic Colloquium on Computational Complexity (ECCC), vol 3. http://eccc.hpi-web.de/report/1996/003/
16. Kitaev A. Yu (1997) Quantum computations: algorithms and error correction. Russ Math Surv 52(6):1191–1249
17. Mosca M, Ekert A (1998) The hidden subgroup problem and eigenvalue estimation on a quantum computer. In: Proceedings 1st NASA International Conference on Quantum Computing & Quantum Communications, Palm Springs. Lecture notes in computer science, vol 1509. Springer, Berlin, pp 174–188
18. Shor P (1994) Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS'94), Santa Fe, pp 124–134
19. Shor P (1997) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J Comput 26:1484–1509
20. Simon D (1994) On the power of quantum computation. In: Proceedings of the 35th IEEE Symposium on the Foundations of Computer Science (FOCS'94), Santa Fe, pp 116–123
21. Simon D (1997) On the power of quantum computation. SIAM J Comput 26:1474–1483
22. Watrous J (2000) Quantum algorithms for solvable groups. In: Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC'00), Portland, pp 60–67
23. Vazirani U (1997) Berkeley lecture notes. Fall. Lecture 8. http://www.cs.berkeley.edu/~vazirani/qc.html

A

Abstract Voronoi Diagrams

Rolf Klein
Institute for Computer Science, University of Bonn, Bonn, Germany

Keywords

Abstract Voronoi diagram; Bisector; Computational geometry; Metric; Voronoi diagram

Years and Authors of Summarized Original Work

2009; Klein, Langetepe, Nilforoushan

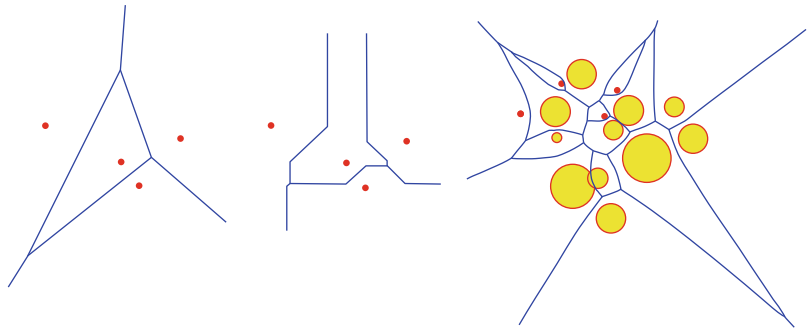
Problem Definition

Concrete Voronoi diagrams are usually defined for a set S of sites p that exert influence over the points z of a surrounding space M. Often, influence is measured by distance functions $d_p(z)$ that are associated with the sites. For each p, its *Voronoi region* is given by

$$\mathrm{VR}(p,S) = \{z \in M;\ d_p(z) < d_q(z) \text{ for all } q \in S \setminus \{p\}\},$$

and the *Voronoi diagram* $V(S)$ of S is the decomposition of M into Voronoi regions; compare the entry ▸ Voronoi Diagrams and Delaunay Triangulations of this Encyclopedia.

Quite different Voronoi diagrams result depending on the particular choices of space, sites, and distance measures; see Fig. 1. A great number of other types of Voronoi diagrams can be found in the monographs [1] and [14]. In each



Abstract Voronoi Diagrams, Fig. 1 Voronoi diagrams of points in the Euclidean and Manhattan metric and of disks (or additively weighted points) in the Euclidean plane

case, one wants to quickly compute the Voronoi diagram, because it contains a lot of distance information about the sites. However, the classical algorithms for the standard case of point sites in the Euclidean plane do not apply to more general situations.

To free us from designing individual algorithms for each and every special case, we would like to find a unifying concept that provides structural results and efficient algorithms for generalized Voronoi diagrams. One possible approach studied in [5,6] is to construct the lower envelope of the 3-dimensional graphs of the distance functions $d_p(z)$, whose projection to the XY-plane equals the Voronoi diagram.

Key Results

A different approach is given by *abstract Voronoi diagrams* that are not based on the notions of sites and distance measures (as their definitions vary anyway). Instead, AVDs are built from *bisecting curves* as primary objects [7].

Let $S = \{p, q, r, \ldots\}$ be a set of n indices, and for $p \neq q \in S$, let $J(p,q) = J(q,p)$ denote an unbounded curve that bisects the plane into two unbounded open domains $D(p,q)$ and $D(q,p)$. We require that each $J(p,q)$ is mapped to a closed Jordan curve through the north pole, under stereographic projection to the sphere. Now we define Voronoi regions by

$$\mathrm{VR}(p,S) := \bigcap_{q \in S \setminus \{p\}} D(p,q)$$

and the abstract Voronoi diagram by

$$V(S) := \mathbf{R}^2 \setminus \bigcup_{p \in S} \mathrm{VR}(p,S).$$

The system $\mathbf{J}$ of the curves $J(p,q)$ is called *admissible* if the following axioms are fulfilled for every subset T of S of size three.

A1. Each Voronoi region $\mathrm{VR}(p,T)$ *is pathwise connected.*

A2. Each point of $\mathbf{R}^2$ *lies in the closure of a Voronoi region* $\mathrm{VR}(p,T)$.

These combinatorial properties should not be too hard to check in a concrete situation because only triplets of sites need to be inspected. Yet, they form a strong foundation, as was shown in [8]. The following fact is crucial for the proof of Theorem 1. It also shows that AVDs can be seen as lower envelopes of surfaces in dimension 3.

Lemma 1 *For all* p,q,r *in* S, *we have* $D(p,q) \cap D(q,r) \subset D(p,r)$. *Consequently, for each point* $z \in \mathbf{R}^2$ *not contained in any curve of* $\mathbf{J}$, *the relation*

$$p <_z q \;:\Leftrightarrow\; z \in D(p,q)$$

is an ordering of the sites in S *at* z.

Theorem 1 *If* $\mathbf{J}$ *is admissible, then axioms A1 and A2 hold for all subsets* T *of* S. *Moreover, the abstract Voronoi diagram* $V(S)$ *is a planar graph of size* $O(n)$.

Of the classical algorithm for constructing Voronoi diagrams, the randomized incremental construction method works best for abstract Voronoi diagrams [8, 10].

Theorem 2 *If* **J** *is admissible, then* $V(S)$ *can be constructed in an expected number of* $O(n \log n)$ *many steps and in expected linear space.*

Here basic operations like computing an intersection of two bisecting curves are counted as one step.

Applications

To show that a concrete type of Voronoi diagram is under the roof of abstract Voronoi diagrams, one needs to prove that its bisector system is admissible.

Let d be a metric in the plane that enjoys the following properties. Each d-disk contains a Euclidean disk and vice versa; for any two points a, b, there exists a point c different from a and b such that $d(a,b) = d(a,c) + d(c,b)$ holds; for any two points a, b, their metric bisector

$$B_d(a,b) = \{z \in \mathbf{R}^2;\ d(a,z) = d(b,z)\}$$

is itself a curve that maps to a closed Jordan curve through the north pole by stereographic projection to the sphere, or, in case $B_d(p,q)$ contains 2-dimensional pieces, its boundary consists of two such curves.

The first two properties ensure that any two points can be connected by a d-straight path along which d-distances add up. The third condition ensures that we can choose from $B_d(p,q)$ suitable bisecting curves. Let us call metric d *very nice* if also a fourth condition is fulfilled. Given three points a, p_0, p_1, there exist d-straight paths from a to p_0 and from a to p_1 that have only point a in common, or each d-straight path from a to p_i contains p_{1-i} for $i = 0$ or $i = 1$. All *convex distance functions (gauges)* are very nice.

Theorem 3 *Very nice metrics have admissible point bisector curves.*

Other applications of AVDs include points with additive weights, both the regular and the Hausdorff Voronoi diagram of disjoint convex sites with respect to a convex distance function, and some types of city Voronoi diagrams; see [1] for further details.

Generalizations

How to *dynamize* abstract Voronoi diagrams has been studied in [12]. Special cases of *3-dimensional* abstract Voronoi diagrams have been discussed in [11]; they include all convex distance functions whose unit spheres are ellipsoids. It is well known that for the vertices of a *convex polygon*, the Voronoi diagram can be constructed in linear time. This result has been generalized to AVDs in [9] and [4]. In [3] the *path-connectedness* of abstract Voronoi regions (axiom A1) has been relaxed. If a region of three sites can have up to s connected components, the abstract Voronoi diagram can still be constructed in expected time $O(s^2 n \sum_{j=3}^{n} m_j/j)$, where m_j denotes the average number of faces per region in any subdiagram of j sites from S.

In an *order-k Voronoi diagram*, all points of space M are placed in one region that shares the same k nearest sites in S. For $k = n-1$, this concept has been generalized to furthest site abstract Voronoi diagrams in [13]. Here the furthest (or inverse) region of $p \in S$ is the intersection of all domains $D(q, p)$, where $q \in S \setminus \{p\}$. If all regular Voronoi regions are nonempty, then the furthest site AVD is a tree of size $O(n)$, even though some regions may be disconnected.

General order-k abstract Voronoi diagrams have been studied in [2]. If all regular Voronoi regions are nonempty and if bisecting curves are in general position, a tight upper complexity bound of $2k(n-k)$ can be shown. Fortunately, the nonemptiness of the regular regions need only be tested for all subsets of S of size 4.

Cross-References

- Voronoi Diagrams and Delaunay Triangulations

Recommended Reading

1. Aurenhammer F, Klein R, Lee DT (2013) Voronoi diagrams and Delaunay triangulations. World Scientific, Singapore
2. Bohler C, Cheilaris P, Klein R, Liu CH, Papadopoulou E, Zavershynskyi M (2013) On the complexity of higher order abstract Voronoi diagrams. In: Proceedings of the 40th international colloquium on automata languages and programming, Riga. Lecture notes in computer science, vol 7965, pp 208–219
3. Bohler C, Klein R (2013) Abstract Voronoi diagrams with disconnected regions. In: Proceedings of the 24th international symposium on algorithms and computation, Hong Kong. Lecture notes in computer science, vol 8283, pp 306–316
4. Bohler C, Klein R, Liu CH (2014) Forest-like abstract Voronoi diagrams in linear time. In: 26th Canadian conference on computational geometry, Halifax
5. Boissonnat JD, Wormser C, Yvinec M (2006) Curved Voronoi diagrams. In: Boissonnat JD, Teillaud M (eds) Effective computational geometry for curves and surfaces. Mathematics and visualization. Springer, Berlin/New York, pp 67–116
6. Edelsbrunner H, Seidel R (1986) Voronoi diagrams and arrangements. Discret Comput Geom 1:387–421
7. Klein R (1989) Concrete and abstract Voronoi diagrams. Lecture notes in computer science, vol 400. Springer, Berlin/New York
8. Klein R, Langetepe E, Nilforoushan Z (2009) Abstract Voronoi diagrams revisited. Comput Geom Theory Appl 42(9):885–902
9. Klein R, Lingas A (1994) Hamiltonian abstract Voronoi diagrams in linear time. In: Proceedings of the 5th international symposium on algorithms and computation, Beijing. Lecture notes in computer science, vol 834, pp 11–19
10. Klein R, Mehlhorn K, Meiser S (1993) Randomized incremental construction of abstract Voronoi diagrams. Comput Geom Theory Appl 3:157–184
11. Lê NM (1995) Randomized incremental construction of simple abstract Voronoi diagrams in 3-space. In: Proceedings of the 10th international conference on fundamentals of computation theory, Dresden. Lecture notes in computer science, vol 965, pp 333–342
12. Malinauskas KK (2008) Dynamic construction of abstract Voronoi diagrams. J Math Sci 154(2): 214–222
13. Mehlhorn K, Meiser S, Rasch R (2009) Furthest site abstract Voronoi diagrams. Int J Comput Geom Appl 11:583–616
14. Okabe A, Boots B, Sugihara K, Chiu SN (2000) Spatial tessellations: concepts and applications of Voronoi diagrams. Wiley, Chichester

Active Learning – Modern Learning Theory

Maria-Florina Balcan and Ruth Urner
Department of Machine Learning, Carnegie Mellon University, Pittsburgh, PA, USA

Keywords

Active learning; Computational complexity; Learning theory; Sample complexity

Years and Authors of Summarized Original Work

2006; Balcan, Beygelzimer, Langford
2007; Balcan, Broder, Zhang
2007; Hanneke
2013; Urner, Wulff, Ben-David
2014; Awashti, Balcan, Long

Problem Definition

Most classic machine learning methods depend on the assumption that humans can annotate all the data available for training. However, many modern machine learning applications (including image and video classification, protein sequence classification, and speech processing) have massive amounts of unannotated or unlabeled data. As a consequence, there has been tremendous interest both in machine learning and its application areas in designing algorithms that most efficiently utilize the available data while minimizing the need for human intervention. An extensively used and studied technique is *active learning*, where the algorithm is presented with a large pool of unlabeled examples (such as all images available on the web) and can interactively ask for the

labels of examples of its own choosing from the pool, with the goal to drastically reduce labeling effort.

Formal Setup

We consider *classification problems* (such as classifying images by who is in them or classifying emails as spam or not), where the goal is to predict a label y based on its corresponding input vector x. In the standard machine learning formulation, we assume that the data points (x, y) are drawn from an unknown underlying distribution D_{XY} over $X \times Y$; X is called the feature (instance) space and $Y = \{0, 1\}$ is the label space. The goal is to output a hypothesis function h of small error (or small 0/1 loss), where $\text{err}(h) = \mathbb{P}_{(x,y)\sim D_{XY}}[h(x) \neq y]$. In the passive learning setting, the learning algorithm is given a set of labeled examples $(x_1, y_1), \ldots, (x_m, y_m)$ drawn i.i.d. from D_{XY} and the goal is to output a hypothesis of small error by using only a polynomial number of labeled examples. In the *realizable* case [10] (PAC learning), we assume that the true label of any example is determined by a deterministic function of the features (the so-called target function) that belongs to a known concept class C (e.g., the class of linear separators, decision trees, etc.). In the *agnostic* case [10, 13], we do not make the assumption that there is a perfect classifier in C, but instead we aim to compete with the best function in C (i.e., we aim to identify a classifier whose error is not much worse than $opt(C)$, the error of the best classifier in C). Both in the realizable and agnostic settings, there is a well-developed theory of Sample Complexity [13], quantifying in terms of the so-called *VC-dimension* (a measure of complexity of a concept class) how many training examples we need in order to be confident that a rule that does well on training data is a good rule for future data as well.

In the active learning setting, a set of labeled examples $(x_1, y_1), \ldots, (x_m, y_m)$ is also drawn i.i.d. from D_{XY}; the learning algorithm is permitted direct access to the sequence of x_i values (unlabeled data points), but has to make a label request to obtain the label y_i of example x_i. The hope is that we can output a classifier of small error by using many fewer label requests than in passive learning by actively directing the queries to informative examples (while keeping the number of unlabeled examples polynomial).

It has been long known that, in the realizable case, active learning can sometimes provide an exponential improvement in label complexity over passive learning. The canonical example [6] is learning threshold classifiers ($X = [0, 1]$ and $C = \{\mathbf{1}_{[0,a]} \mid a \in [0, 1]\}$). Here we can actively learn with only $\tilde{O}(\log(1/\epsilon))$ label requests by using a simple binary search-like algorithm as follows: we first draw $N = \tilde{O}((1/\epsilon)\log(1/\delta))$ unlabeled examples, then do binary search to find the transition from label 1 to label 0, and with only $O(\log(N))$ queries we can correctly infer the labels of all our examples; we finally output a classifier from C consistent with all the inferred labels. By standard VC-dimension based bounds for supervised learning [13], we are guaranteed to output an ϵ-accurate classifier. On the other hand, for passive learning, we provably need $\Omega(1/\epsilon)$ labels to output a classifier of error at most ϵ with constant probability, yielding the exponential reduction in label complexity.

Key Results

While in the simple threshold concept class described above active learning always provides huge improvements over passive learning, things are more delicate in more general scenarios. In particular, both in the realizable and in the agnostic case, it has been shown that for more general concept spaces, in the worst case over all data-generating distributions, the label complexity of active learning equals that of passive learning. Thus, much of the literature was focused on identifying non-worst case, natural conditions about the relationship between the data distribution and the target, under which active learning provides improvements over passive. Below, we discuss three approaches, under which active learning has been shown to reduce the label complexity: disagreement-based

techniques, margin-based techniques and cluster-based techniques.

Disagreement-Based Active Learning

Disagreement-based active learning was the first method to demonstrate the feasibility of *agnostic active learning* for general concept classes. The general algorithmic framework of disagreement-based active learning in the presence of noise was introduced with the A^2 algorithm by Balcan et al. [2]. Subsequently, several researchers have proposed related disagreement-based algorithms with improved sample complexity, e.g., [5, 8, 11].

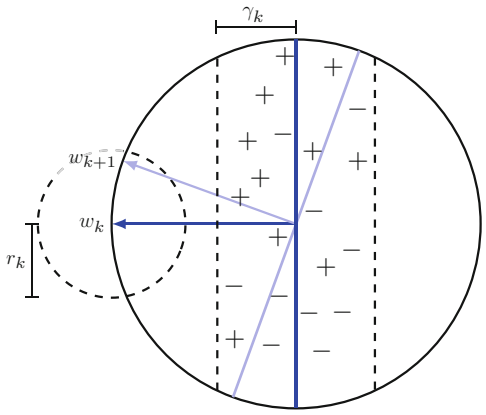
At a high level, A^2 operates in rounds. It maintains a set of candidate classifiers from the concept class C and in each round queries labels aiming to efficiently reduce this set to only few high-quality candidates. More precisely, in round i, A^2 considers the set of surviving classifiers $C_i \subseteq C$, and asks for the labels of a few random points that fall in the *region of disagreement* of C_i. Formally, the region of disagreement of a set of classifiers C_i is $\mathrm{DIS}(C_i) = \{x \in X \mid \exists f, g \in C_i : f(x) \neq g(x)\}$. Based on these queried labels from $\mathrm{DIS}(C_i)$, to obtain C_{i+1}, the algorithm then throws out hypotheses that are suboptimal. The key ingredient is that A^2 *only* throws out hypotheses, for which it is *statistically confident* that they are suboptimal.

Balcan et al. [2] show that A^2 provides exponential improvements in the label sample complexity in terms of the $1/\epsilon$-parameter when the noise rate η is sufficiently small, both for learning thresholds and for learning homogeneous linear separators in R^d, one of the most widely used and studied classes in machine learning. Following up on this, Hanneke [9] provided a generic analysis of the A^2 algorithm that applies to *any concept class*. This analysis quantifies the label complexity of A^2 in terms of the so-called *disagreement coefficient* of the class C. The disagreement coefficient is a distribution-dependent sample complexity measure that quantifies how fast the region of disagreement of the set of classifiers at distance r of the optimal classifier collapses as a function r. In particular, [9] showed that the label complexity of the A^2 algorithm is $O\left(\theta^2\left(\frac{\nu^2}{\epsilon^2}+1\right)(d\log(1/\epsilon)+\log(1/\delta))\log(1/\epsilon)\right)$, where ν is the best error rate of a classifier in C, d is the VC-dimension of C, and θ is the disagreement-coefficient. As an example, for homogeneous linear separators, we have $\theta = \theta(\sqrt{d})$ under uniform marginal over the unit ball. Here, the disagreement-based analysis yields a label complexity of $\tilde{O}\left(d^2\frac{\nu^2}{\epsilon^2}\log(1/\epsilon)\right)$ in the agnostic case and $\tilde{O}\left(d^{3/2}\log(1/\epsilon)\right)$ in the realizable case.

Margin-Based Active Learning

While the disagreement-based active learning line of work provided the first general understanding of the sample complexity benefits with active learning for arbitrary concept classes, it suffers from two main drawbacks: (1) methods and analyses developed in this context are often suboptimal in terms of label complexity, since they take a conservative approach and query even points on which there is only a small amount of uncertainty, (2) the methods are computationally inefficient. *Margin-based* active learning is a technique that overcomes both the above drawbacks for learning homogeneous linear separators under log-concave distributions. The technique was first introduced by Balcan et al. [3] and further developed by Balcan et al. [4], and Awasthi et al. [1].

At a high level, like disagreement-based methods, the margin-based active learning algorithm operates in rounds, in which a number of labels are queried in some subspace of the domain and a set of candidate classifiers for the next round is identified. The crucial idea to reduce the label complexity is to design a *more aggressive querying strategy* by carefully choosing where to query instead of querying in all of the current disagreement region. Concretely, in round k the algorithm has a *current hypothesis* w_k, and the set of candidate classifiers for the next round consists of all homogeneous halfspaces that lie in a *ball of radius* r_k *around* w_k (in terms of their angle with w_k). The algorithm then queries points for labels near the decision boundary of w_k; that is, it only queries points



Active Learning – Modern Learning Theory, Fig. 1 The margin-based active learning algorithm after iteration k. The algorithm samples points within margin γ_k of the current weight vector w_k and then minimizes the hinge loss over this sample subject to the constraint that the new weight vector w_{k+1} is within distance r_k from w_k

that are within a *margin* γ_k of w_k; see Fig. 1. To obtain w_{k+1}, the algorithm finds a loss minimizer among the current set of candidates with respect to the queried examples of round k. In the realizable case, this is done by 0/1-loss minimization. In the presence of noise, to obtain a computationally efficient procedure, the margin-based technique minimizes a convex surrogate loss.

Balcan et al. [3] and Balcan and Long [4] showed that by localizing aggressively, namely by setting the margin parameter to $\gamma_k = \Theta(\frac{1}{2^k})$, one can actively learn with only $\tilde{O}(d\log(1/\epsilon))$ label requests in the realizable case, when the underlying distribution is isotropic log-concave. A key idea of their analysis is to decompose, in round k, the error of a candidate classifier w as its error outside margin γ_k of the current separator plus its error inside margin γ_k, and to prove that for the above parameters, a small constant error inside the margin suffices to reduce the overall error by a constant factor. For the constant error inside the margin only $\theta(d)$ labels need to be queried, and since in each round the overall error gets reduced by a constant factor, $O(\log(1/\epsilon))$ rounds suffice to reduce the error to ϵ, yielding the label complexity of $\tilde{O}(d\log(1/\epsilon))$. Passive learning here provably requires $\Omega(d/\epsilon)$ labeled examples. Thus, the dependence on $1/\epsilon$ is exponentially improved, but without increasing the dependence on d (as in the disagreement-based method for this case, see above).

Building on this work, [1] gave the first *polynomial-time* active learning algorithm for learning linear separators to error ϵ in the presence of agnostic noise (of rate $O(\epsilon)$) when the underlying distribution is an isotropic log-concave distribution in R^d. They proposed to use a normalized hinge loss minimization (with normalization factor τ_k) for selecting the next classifier w_{k+1} in round k. Awasthi et al. [1] show that by setting the parameters appropriately (namely, $\tau_k = \Theta(1/2^k)$ and $r_k = \Theta(1/2^k)$), the algorithm again achieves error ϵ using only $O(\log(1/\epsilon))$ rounds, with $O(d^2)$ label requests per round. This yields a query complexity of $\text{poly}(d, \log 1/\epsilon)$. The key ingredient for the analysis of this computationally efficient version in the noisy setting is proving that by constraining the search for w_{k+1} to vectors within a ball of radius r_k around w_k, the hinge-loss acts as a sufficiently faithful proxy for the 0/1-loss.

A recent work [14] proposes an elegant generalization of [3, 4] to more general concept spaces and shows an analysis that is always tighter than disagreement-based active learning (though their results are not computationally efficient).

Cluster-Based Active Learning

The methods described above (disagreement-based and margin-based active learning) use active label queries to efficiently identify a classifier from the concept class C with low error. An alternative approach to agnostic active learning is to design active querying methods that efficiently find a (approximately) correct labeling of the unlabeled input sample. Here, "correct labeling" refers to the hidden labels y_i in the sample $(x_1, y_1), \ldots, (x_m, y_m)$ from the distribution D_{XY} (as defined in the formal setup section). The so labeled sample can then be used as input to a passive learning algorithm to learn an arbitrary concept class.

Cluster-based active learning is a method for the latter approach and was introduced by Dasgupta and Hsu [7]. The idea is to use

a hierarchical clustering (cluster tree) of the unlabeled data, and check the clusters for label homogeneity by starting at the root of the tree (the whole data set) and working towards the leaves (single data points). The label homogeneity of a cluster is estimated by choosing data points for label query uniformly at random from the cluster. If a cluster is considered label homogeneous (with sufficiently high confidence), all remaining unlabeled points in that cluster are labeled with the majority label. If a cluster is detected to be label heterogeneous, it is split into its children in the cluster tree and processed later. The key insight in [7] is that since the cluster tree is fixed before any labels were seen, the induced labeled subsample of a child cluster can be considered a sample that was chosen uniformly at random from the points in that child-cluster. Thus, the algorithm can reuse labels from the parent cluster without introducing any sampling bias. The label efficiency of this paradigm crucially depends on the quality of input hierarchical clustering. Intuitively, if the cluster tree has a small pruning with label homogeneous clusters, the procedure will make only few label queries.

Urner et al. [12] proved label complexity reductions with this paradigm under a distributional assumption. They analyze a version (PLAL) of the above paradigm that uses hierarchical clusterings induced by *spatial trees* on the domain $[0,1]^d$ and provide label query bounds in terms of the *Probabilistic Lipschitzness* of the underlying data-generating distribution. Probabilistic Lipschitzness quantifies a marginal-label relatedness in the sense of close points being likely to have the same label. For a distribution with deterministic labels ($\Pr[Y = 1 \mid X = x] \in \{0,1\}$ for all x), the Probabilistic Lipschitzness is a function ϕ that bounds, as a function of λ, the mass of points x for which both labels 0 and 1 occur in the ball $B_\lambda(x)$.

Urner et al. [12] show that, independently of the any data assumptions, (with probability $1-\delta$) PLAL labels a $(1-\epsilon)$-fraction of the input points correctly. They further show that using PLAL as a preprocedure, if the data-generating distribution has deterministic labels and its Probabilistic Lipschitzness is bounded by $\phi(\lambda) = \lambda^n$ for some $n \in \mathbb{N}$, then classes C of bounded VC-dimension on domain $X = [0,1]^d$ can be learned with $\tilde{O}\left(\left(\frac{1}{\epsilon}\right)^{\frac{n+2d}{n+d}}\right)$ many labels, while any passive proper learner (i.e., a passive learner that outputs a function from C) requires to see $\Omega(1/\epsilon^2)$ many labels. Further, [12] show that PLAL can be used to reduce the number of labels needed for nearest neighbor classification (i.e., labeling a test point by the label of its nearest point in the sample) from $\Omega\left(\left(\frac{1}{\epsilon}\right)^{1+\frac{d-1}{n}}\right)$ to $\tilde{O}\left(\left(\frac{1}{\epsilon}\right)^{1+\frac{d^2}{n(n+d)}}\right)$.

Cross-References

▸ PAC Learning

Recommended Reading

1. Awasthi P, Balcan M-F, Long PM (2014) The power of localization for efficiently learning linear separators with noise. In: Proceedings of the 46th annual symposium on the theory of computing (STOC), New York
2. Balcan MF, Beygelzimer A, Langford J (2006) Agnostic active learning. In: Proceedings of the 23rd international conference on machine learning (ICML), Pittsburgh
3. Balcan M-F, Broder A, Zhang T (2007) Margin based active learning. In: Proceedings of the 20th annual conference on computational learning theory (COLT), San Diego
4. Balcan M-F, Long PM (2013) Active and passive learning of linear separators under log-concave distributions. In: Proceedings of the 26th conference on learning theory (COLT), Princeton
5. Beygelzimer A, Hsu D, Langford J, Zhang T (2010) Agnostic active learning without constraints. In: Advances in neural information processing systems (NIPS), Vancouver
6. Cohn D, Atlas L, Ladner R (1994) Improving generalization with active learning. In: Proceedings of the 11th international conference on machine learning (ICML), New Brunswick
7. Dasgupta S, Hsu D (2008) Hierarchical sampling for active learning. In: Proceedings of the 25th international conference on machine learning (ICML), Helsinki
8. Dasgupta S, Hsu DJ, Monteleoni C (2007) A general agnostic active learning algorithm. In: Advances in neural information processing systems (NIPS), Vancouver
9. Hanneke S (2007) A bound on the label complexity of agnostic active learning. In: Proceedings of the

24th international conference on machine learning (ICML), Corvallis
10. Kearns MJ, Vazirani UV (1994) An introduction to computational learning theory. MIT, Cambridge
11. Koltchinskii V (2010) Rademacher complexities and bounding the excess risk in active learning. J Mach Learn 11:2457–2485
12. Urner R, Wullf S, Ben-David S (2013) Plal: cluster-based active learning. In: Proceedings of the 26th conference on learning theory (COLT), Princeton
13. Vapnik VN (1998) Statistical learning theory. Wiley, New York
14. Zhang C, Chaudhuri K (2014) Beyond disagreement-based agnostic active learning. In: Advances in neural information processing systems (NIPS), Montreal

Active Self-Assembly and Molecular Robotics with Nubots

Damien Woods
Computer Science, California Institute of Technology, Pasadena, CA, USA

Keywords

Molecular robotics; Rigid-body motion; Self-assembly

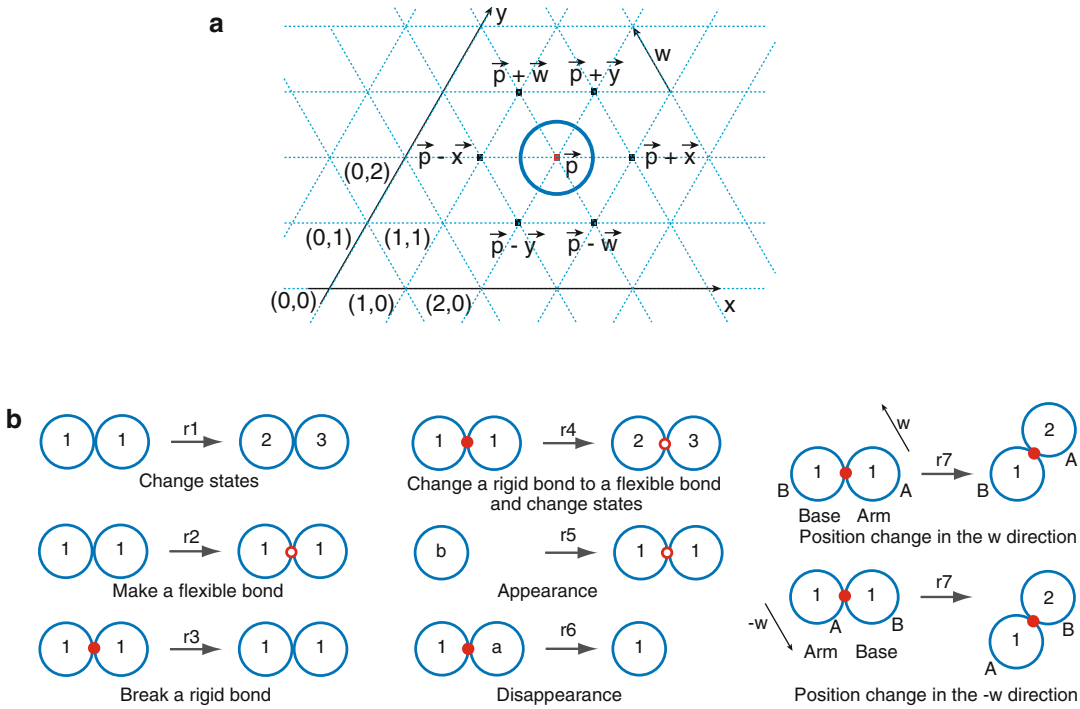
Years and Authors of Summarized Original Work

2013; Woods, Chen, Goodfriend, Dabby, Winfree, Yin
2013; Chen, Xin, Woods
2014; Chen, Doty, Holden, Thachuk, Woods, Yang

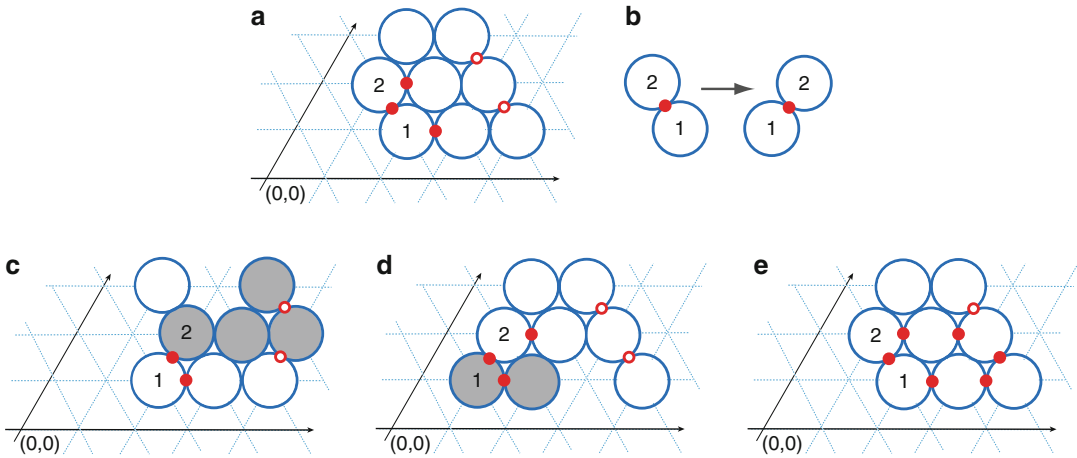
Problem Definition

In the theory of molecular-scale self-assembly, large numbers of simple interacting components are designed to come together to build complicated shapes and patterns. Many models of self-assembly, such as the abstract Tile Assembly Model [6], are cellular automata-like crystal growth models. Indeed such models have given rise to a rich theory of self-assembly as described elsewhere in this encyclopedia. In biological organisms we frequently see much more sophisticated growth processes, where self-assembly is combined with active molecular components that change internal state and even molecular motors that have the ability to push and pull large structures around. Molecular engineers are now beginning to design and build molecular-scale DNA motors and active self-assembly systems [2]. We wish to understand, at a high level of abstraction, the ultimate computational capabilities and limitations of such molecular-scale rearrangement and growth. The nubot model, put forward in [8], is akin to an asynchronous nondeterministic cellular automaton augmented with nonlocal rigid-body movement. Unit-sized monomers are placed on a 2D triangular grid. Monomers undergo state changes, appear, and disappear using local rules, as shown in Fig. 1. However, there is also a nonlocal aspect to the model: rigid-body movement that comes in two forms, movement rules and random agitations.

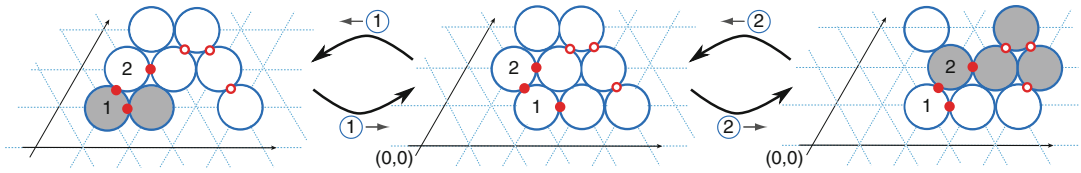
A *movement rule* r, consisting of a pair of monomer states A, B and two unit vectors, is a programmatic way to specify unit-distance translation of a set of monomers in one step. See Fig. 2 for an example. If A and B are in a prescribed orientation, one is nondeterministically chosen to move unit distance in a prescribed direction. The rule r is applied in a rigid-body fashion: roughly speaking, if A is to move right, it pushes anything immediately to its right and pulls any monomers that are bound to its left which in turn push and pull other monomers, all in one step. The rule may not be applicable if it is blocked (i.e., if movement of A would force B to also move), which is analogous to the fact that an arm cannot push its own shoulder. The other, somewhat related, form of movement is called *agitation*: at every point in time, every monomer on the grid may move unit distance in any of the six directions, at unit rate for each (monomer, direction) pair. An agitating monomer will push or pull any monomers that it is adjacent to, in a way that preserves rigid-body structure and all in



Active Self-Assembly and Molecular Robotics with Nubots, Fig. 1 Overview of the nubot model. (**a**) A nubot configuration showing a single nubot monomer on the triangular grid. (**b**) Examples of nubot monomer rules. Rules r1–r6 are local cellular automaton-like rules, whereas r7 effects a nonlocal movement that may translate other monomers as shown in Fig. 2. Monomers continuously undergo agitation, as shown in Fig. 3. A flexible bond is depicted as an empty *red circle* and a rigid bond is depicted as a *solid red disk* (from [8])



Active Self-Assembly and Molecular Robotics with Nubots, Fig. 2 Movement rule. (**a**) Initial configuration. (**b**) Movement rule with one of two results depending on the choice of arm or base. (**c**) Result if the monomer with state 2 is the arm or (**d**) monomer with state 1 is the arm. The shaded monomers are the movable set. The affect on rigid (*filled red disks*), flexible (*hollow red circles*), and null bonds is shown. (**e**) A configuration for which the movement rule is blocked: movement of 1 or 2 would force the other to move; hence the rule is not applicable (from [3])



Active Self-Assembly and Molecular Robotics with Nubots, Fig. 3 Example agitations. Starting from the centre configuration, there are 48 possible agitations (8 monomers, 6 directions each) each with equal probability. The right configuration is the result of agitation of the monomer in state 2 in direction →, the left is the result of the agitation of the monomer in state 1 in direction ←. The shaded monomers are the agitation set – monomers that are moved by the agitation (from [4])

one step as shown in Fig. 3. Unlike movement, agitations are never blocked. Rules are applied asynchronously and in parallel. Taking its time model from stochastic chemical kinetics, a nubot system evolves as a continuous time Markov process.

For intuition, we describe motion in terms of pushing and pulling. However movement and agitation are actually intended to model a nanoscale environment with diffusion, Brownian motion, convection, turbulent flow, cytoplasmic streaming, and other uncontrolled inputs of energy that interact monomers in all directions, moving large molecular assemblies in a random fashion (i.e., agitation) and allowing motors to simply latch and unlatch large assemblies into position (i.e., the movement rule).

Key Results

Assembling simple structures, namely, lines and squares, has proven to be a fruitful way to explore the power of the nubot model for a few reasons. Firstly, it helps us develop a number of techniques and intuitions for the model. Secondly, lines and squares get used again and again in more general results that show the full power of the model. Thirdly, the efficiency of assembling simple shapes has been a de facto benchmark problem for a number of self assembly models (although this benchmark often does not give the full story). In a variety of models, such as the abstract Tile Assembly Model, cellular automata, and some robotics models, it takes time $\Omega(n)$ to assemble a length n line. In the nubot model this is achieved in merely $O(\log n)$ expected time and $O(\log n)$ states.

Theorem 1 ([8]) *For each $n \in \mathbb{N}$, there is a set of nubot rules $\mathcal{N}_n^{\text{line}}$ such that starting from a single monomer $\mathcal{N}_n^{\text{line}}$ assembles a length n line in $O(\log n)$ expected time, $n \times 2$ space, and $O(\log n)$ states.*

One can trade time for states by giving a slightly slower method with fewer states:

Theorem 2 ([3]) *There is a set of nubot rules $\mathcal{N}^{\text{line}}$ such that for each $n \in \mathbb{N}$, from a line of $O(\log n)$ "binary" monomers (each in state 0 or 1), $\mathcal{N}^{\text{line}}$ assembles a length n line in $O(\log^2 n)$ expected time, $n \times O(1)$ space, and $O(1)$ states.*

An $n \times n$ square can be built by growing a horizontal line and then n vertical lines, showing that assembly of squares with nubots is exponentially faster than the $\Theta(n)$ expected time seen in the abstract Tile Assembly Model [1]:

Theorem 3 ([8]) *For each $n \in \mathbb{N}$, there is a set of nubot rules $\mathcal{N}_n^{\text{square}}$ such that starting from a single monomer, $\mathcal{N}_n^{\text{square}}$ assembles a $n \times n$ square in $O(\log n)$ expected time, $n \times n$ space, and $O(\log n)$ states.*

The results above, and all of those in [3, 8], crucially make use of the rigid-body movement rule: the ability for a single monomer to control the movement of large objects quickly and at a time and place of the programmer's choosing. However, in a molecular-scale environment, molecular motion is happening in a largely uncontrolled and fundamentally random manner, all of the time. The *agitation nubot* model does

not have the movement rule, but instead permits such uncontrolled random agitation (movement). Although this form of movement is challenging to control in a precise manner, the following result shows we can use it to achieve sublinear expected time growth of a length n line in only $O(n)$ space:

Theorem 4 ([4]) *There is a set of nubot rules $\mathcal{N}_{\text{line}}$, such that $\forall n \in \mathbb{N}$, starting from a line of $\lfloor \log_2 n \rfloor + 1$ monomers, each in state 0 or 1, $\mathcal{N}_{\text{line}}$ in the agitation nubot model assembles an $n \times 1$ line in $O(n^{1/3} \log n)$ expected time, $n \times 5$ space, and $O(1)$ monomer states.*

For a square we can do much better, achieving polylogarithmic expected time:

Theorem 5 ([4]) *There is a set of nubot rules $\mathcal{N}_{\text{square}}$, such that $\forall n \in \mathbb{N}$, starting from a line of $\lfloor \log_2 n \rfloor + 1$ monomers, each in state 0 or 1, $\mathcal{N}_{\text{square}}$ in the agitation nubot model assembles an $n \times n$ square in $O(\log^2 n)$ expected time, $n \times n$ space, and $O(1)$ monomer states.*

This section concludes with three results on general-purpose computation and shape construction with the nubot model. First we have a computability-theoretic result: any finite computable connected shape can be quickly self-assembled.

Theorem 6 ([8]) *An arbitrary connected computable 2D shape of size $\leq \sqrt{n} \times \sqrt{n}$ can be assembled in expected time $O(\log^2 n + t(|n|))$ using $O(s + \log n)$ states. Here, $t(|n|)$ is the time required for a program-size s Turing machine to compute, given a pixel index as a binary string of length $|n| = \lfloor \log_2 n \rfloor + 1$, whether or not the pixel is present in the shape.*

For complicated computable shapes the construction for Theorem 6 necessarily requires computation workspace outside of the shape's bounding box. The next result is of a more resource-bounded style and, roughly speaking, states that 2D patterns with efficiently computable pixel colors can be assembled using nubots in merely polylogarithmic expected time while staying inside the pattern's bounding box.

Theorem 7 ([8]) *An arbitrary finite computable 2D pattern of size $\leq n \times n$, where $n = 2^p$, $p \in \mathbb{N}$, with pixels whose color is computable on a polynomial time $O(|n|^\ell)$ (inputs are binary strings of length $|n| = O(\log n)$), linear space $O(|n|)$, program-size s Turing machine, can be assembled in expected time $O(\log^{\ell+1} n)$, with $O(s + \log n)$ monomer states and without growing outside the pattern borders.*

The results cited so far can be used to compare the nubot model to other models of self-assembly and tell us that nubots build shapes and patterns in a fast parallel manner. The next result quantifies this parallelism in terms of a well-known parallel model from computational complexity theory: NC is the class of problems solved by uniform polylogarithmic depth and polynomial-size Boolean circuits.

Theorem 8 ([3]) *For each language $L \in$ NC, there is a set of nubot rules $\mathcal{N}_L$ that decides L in polylogarithmic expected time, constant number of monomer states, and polynomial space in the length of the input string of binary monomers (in state 0 or 1). The output is a single binary monomer.*

This result stands in contrast to sequential machines like Turing machines, that cannot read all of an n-bit input string in polylogarithmic time, and "somewhat parallel" models like cellular automata and the abstract Tile Assembly Model, that cannot have all of n bits influence a single output bit decision in polylogarithmic time [5]. Thus, adding the nubot rigid-body movement primitive to an asynchronous nondeterministic cellular automaton drastically increases its parallel processing abilities.

Open Problems

Some future research directions are discussed here and in [3, 4, 8]. It remains as future work to look at other topics such as fault tolerance, self-healing, dynamical tasks, or systems that continuously respond to the environment.

The Complexity of Assembling Lines

Theorem 1 states that a line can be grown in expected time $O(\log n)$, space $O(n) \times O(1)$, and $O(\log n)$ states, and Theorem 2 trades time for states to get expected time $O(\log^2 n)$, space $O(n) \times O(1)$, and $O(1)$ states. What is the complexity (expected time $\times$ states) of assembling a line in the nubot model? Is it possible to meet the lower bound of expected time $\times$ states $= \Omega(\log n)$? In this problem, the input should be a set of monomers with space $\times$ states $= O(\log n)$.

Computational Power

Theorem 8 gives a lower bound on the computational complexity of the nubot model. What is the exact power of polylogarithmic expected time nubots? The answer may differ on whether we begin from a small collection of monomers (as in Theorem 8) or a large prebuilt structure. One challenge, for the upper bound, involves finding better Turing machine space, or circuit depth, bounds on computing multiple applications of the movable set on a large nubots grid.

Synchronization and Composition of Nubot Algorithms

Synchronization is a method to quickly send signals using nonlocal rigid-body motion [3, 8]. The nubot model is asynchronous, but synchronization can be used to set discrete stages, or checkpoints, during a complicated construction. This in turn facilitates composition of nubot algorithms (run algorithm 1, synchronize, run algorithm 2, synchronize, etc.) and many of the results cited here use it for exactly that reason. However, synchronization-less constructions often exhibit a kind of independence where growth proceeds everywhere in parallel, without waiting on signals from distant components. Such systems are highly distributed, easy to analyze, and perhaps more amenable to laboratory implementation. Intuitively, this seems like the right way to program molecules. The proof of Theorem 7 does not use synchronization which shows that without it a very general class of (efficiently) computable patterns can be grown and indeed the proof gives methods to compose nubot algorithms without resorting to synchronization. It remains as future work to formalize both this notion of synchronization-less "independence" and what we mean by "composition" of nubot algorithms. What conditions are necessary and sufficient for composition of nubot algorithms? What classes of shapes and patterns can be assembled using without synchronization or other forms of rapid long-range communication?

Agitation Versus the Movement Rule

Is it possible to simulate the movement rule using agitation? More formally, is it the case that for each nubot program $\mathcal{N}$, there is an agitation nubot program $\mathcal{A}_{\mathcal{N}}$, that acts just like $\mathcal{N}$ but with some $m \times m$ scale-up in space, and a k factor slowdown in time, where m and k are (constants) independent of $\mathcal{N}$ and its input? As motivation, note that every self-assembled molecular-scale structure was made under conditions where random jiggling of monomers is a dominant source of movement! Our question asks if we can *programmably exploit* this random molecular motion to build structures quicker than without it.

Intrinsic Universality and Simulation

Is the nubot model intrinsically universal? Specifically, does there exist a set of monomer rules U, such that any nubot system $\mathcal{N}$ can be simulated by "seeding" U with a suitable initial configuration? Here the simulation should have a spatial scale factor m that is a function of the number of states in the simulated system $\mathcal{N}$. Is the agitation nubot model intrinsically universal? Our hope would be that simulation could be used to tease apart the power of different notions of movement (e.g., to understand if nubot-style movement is weaker or stronger than other notions of robotic movement), in the way it has been used to characterize and separate the power of other self-assembly models [7].

Brownian Nubots

With nubots, under agitation, or multiple parallel movement rules, larger objects move faster. This is intended to model an environment with uncontrolled and rapid fluid flows. But in Brownian motion, larger objects move slower: what is the power of nubots with such a rate model, for

example, with rate equal to object size? Although assembly in such a model may be slower than with the usual model, many of the same programming principles should apply, and indeed it will still be possible to assemble objects in a parallel distributed fashion.

Cross-References

- Combinatorial Optimization and Verification in Self-Assembly
- Intrinsic Universality in Self-Assembly
- Patterned Self-Assembly Tile Set Synthesis
- Randomized Self-Assembly
- Robustness in Self-Assembly
- Self-Assembly at Temperature 1
- Self-Assembly of Fractals
- Self-Assembly of Squares and Scaled Shapes
- Self-Assembly with General Shaped Tiles
- Temperature Programming in Self-Assembly

Acknowledgments A warm thanks to all of my coauthors on this topic and especially to Erik Winfree and Chris Thachuk for their helpful comments. The author is supported by NSF grants 0832824, 1317694, CCF-1219274, and CCF-1162589.

Recommended Reading

1. Adleman LM, Cheng Q, Goel A, Huang MD (2001) Running time and program size for self-assembled squares. In: STOC 2001: proceedings of the 33rd annual ACM symposium on theory of computing, Hersonissos. ACM, pp 740–748
2. Bath J, Turberfield A (2007) DNA nanomachines. Nat Nanotechnol 2:275–284
3. Chen M, Xin D, Woods D (2013) Parallel computation using active self-assembly. In: DNA19: the 19th international conference on DNA computing and molecular programming. LNCS, vol 8141. Springer, pp 16–30. arxiv preprint arXiv:1405.0527
4. Chen HL, Doty D, Holden D, Thachuk C, Woods D, Yang CT (2014) Fast algorithmic self-assembly of simple shapes using random agitation. In: DNA20: the 20th international conference on DNA computing and molecular programming. LNCS, vol 8727. Springer, pp 20–36. arxiv preprint: arXiv:1409.4828
5. Keenan A, Schweller R, Sherman M, Zhong X (2014) Fast arithmetic in algorithmic self-assembly. In: UCNC: the 13th international conference on unconventional computation and natural computation. LNCS, vol 8553. Springer, pp 242–253. arxiv preprint arXiv:1303.2416 [cs.DS]
6. Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology
7. Woods D (2015) Intrinsic universality and the computational power of self-assembly. Philos Trans R Soc A: Math Phys Eng Sci 373(2046). doi:10.1098/rsta.2014.0214, ISBN:1471-2962, ISSN:1364-503X
8. Woods D, Chen HL, Goodfriend S, Dabby N, Winfree E, Yin P (2013) Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In: ITCS'13: proceedings of the 4th conference on innovations in theoretical computer science. ACM, pp 353–354. Full version: arXiv:1301.2626 [cs.DS]

Adaptive Partitions

Ping Deng[1], Weili Wu[1,4,5], Eugene Shragowitz[2], and Ding-Zhu Du[1,3]
[1]Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA
[2]Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA
[3]Computer Science, University of Minnesota, Minneapolis, MN, USA
[4]College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi Province, China
[5]Department of Computer Science, California State University, Los Angeles, CA, USA

Keywords

Technique for constructing approximation

Years and Authors of Summarized Original Work

1986; Du, Pan, Shing

Problem Definition

Adaptive partition is one of major techniques to design polynomial-time approximation algorithms, especially polynomial-time approximation schemes for geometric optimization problems. The framework of this technique is to put the input data into a rectangle and partition this rectangle into smaller rectangles by a sequence of cuts so that the problem is also partitioned into smaller ones. Associated with each adaptive partition, a feasible solution can be constructed recursively from solutions in smallest rectangles to bigger rectangles. With dynamic programming, an optimal adaptive partition is computed in polynomial time.

Historical Note

The adaptive partition was first introduced to the design of an approximation algorithm by Du et al. [4] with a guillotine cut while they studied the minimum edge-length rectangular partition (MELRP) problem. They found that if the partition is performed by a sequence of guillotine cuts, then an optimal solution can be computed in polynomial time with dynamic programming. Moreover, this optimal solution can be used as a pretty good approximation solution for the original rectangular partition problem. Both Arora [1] and Mitchell et al. [12,15] found that the cut does not need to be completely guillotine. In other words, the dynamic programming can still run in polynomial time if subproblems have some relations but the number of relations is small. As the number of relations goes up, the approximation solution obtained approaches the optimal one, while the run time, of course, goes up. They also found that this technique can be applied to many geometric optimization problems to obtain polynomial-time approximation schemes.

Key Results

The MELRP was proposed by Lingas et al. [10] as follows: Given a rectilinear polygon possibly with some rectangular holes, partition it into rectangles with minimum total edge length. Each hole may be degenerated into a line segment or a point.

There are several applications mentioned in [10] for the background of the problem: process control (stock cutting), automatic layout systems for integrated circuit (channel definition), and architecture (internal partitioning into offices). The *minimum edge-length* partition is a natural goal for these problems since there is a certain amount of waste (e.g., sawdust) or expense incurred (e.g., for dividing walls in the office) which is proportional to the sum of edge lengths drawn. For very large-scale integration (VLSI) design, this criterion is used in the MIT placement and interconnect (PI) system to divide the routing region up into channels – one finds that this produces large "natural-looking" channels with a minimum of channel-to-channel interaction to consider.

They showed that while the MELRP in general is nondeterministic polynomial-time (NP)-hard, it can be solved in time $O(n^4)$ in the hole-free case, where n is the number of vertices in the input rectilinear polygon. The polynomial algorithm is essentially a dynamic programming based on the fact that there always exists an optimal solution satisfying the property that every cut line passes through a vertex of the input polygon or holes (namely, every maximal cut segment is incident to a vertex of input or holes).

A naive idea to design an approximation algorithm for the general case is to use a forest connecting all holes to the boundary and then to solve the resulting hole-free case in $O(n^4)$ time. With this idea, Lingas [9] gave the first constant-bounded approximation; its performance ratio is 41.

Motivated by a work of Du et al. [6] on application of dynamic programming to optimal routing trees, Du et al. [4] initiated an idea of adaptive partition. They used a sequence of guillotine cuts to do rectangular partition; each guillotine cut breaks a connected area into at least two parts. With dynamic programming, they were able to show that a minimum-length guillotine rectangular partition (i.e., one with minimum total length among all guillotine partitions) can

be computed in $O(n^5)$ time. Therefore, they suggested using the minimum-length guillotine rectangular partition to approximate the MELRP and tried to analyze the performance ratio. Unfortunately, they failed to get a constant ratio in general and only obtained an upper bound of 2 for the performance ratio in an NP-hard special case [7]. In this special case, the input is a rectangle with some points inside. Those points are holes. The following is a simple version of the proof obtained by Du et al. [5].

Theorem 1 *The minimum-length guillotine rectangular partition is an approximation with performance ratio 2 for the MELRP.*

Proof Consider a rectangular partition P. Let $proj_x(P)$ denote the total length of segments on a horizontal line covered by vertical projection of the partition P.

A rectangular partition is said to be covered by a guillotine partition if each segment in the rectangular partition is covered by a guillotine cut of the latter. Let $guil(P)$ denote the minimum length of the guillotine partition covering P and $length(P)$ denote the total length of rectangular partition P. It will be proved by induction on the number k of segments in P that

$$guil(P) \le 2 \cdot length(P) - proj_x(P).$$

For $k = 1$, one has $guil(P) = length(P)$. If the segment is horizontal, then one has $proj_x(P) = length(P)$ and hence

$$guil(P) = 2 \cdot length(P) - proj_x(P).$$

If the segment is vertical, then $proj_x(P) = 0$ and hence

$$guil(P) < 2 \cdot length(P) - proj_x(P).$$

Now, consider $k \ge 2$. Suppose that the initial rectangle has each vertical edge of length a and each horizontal edge of length b. Consider two cases.

Case 1. There exists a vertical segment s having length greater than or equal to $0.5a$. Apply a guillotine cut along this segment s. Then the remainder of P is divided into two parts, P_1 and P_2, which form rectangular partition of two resulting small rectangles, respectively. By induction hypothesis,

$$guil(P_i) \le 2 \cdot length(P_i) - proj_x(P_i)$$

for $i = 1, 2$. Note that

$$\begin{aligned} guil(P) &\le guil(P_1) + guil(P_2) + a, \\ length(P) &= length(P_1) + length(P_2) \\ &\quad + length(s), \\ proj_x(P) &= proj_x(P_1) + proj_x(P_2). \end{aligned}$$

Therefore,

$$guil(P) \le 2 \cdot length(P) - proj_x(P).$$

Case 2. No vertical segment in P has length greater than or equal to $0.5a$. Choose a horizontal guillotine cut which partitions the rectangle into two equal parts. Let P_1 and P_2 denote rectangle partitions of the two parts, obtained from P. By induction hypothesis,

$$guil(P_i) \le 2 \cdot length(P_i) - proj_x(P_i)$$

for $i = 1, 2$. Note that

$$\begin{aligned} guil(P) &= guil(P_1) + guil(P_2) + b, \\ length(P) &\ge length(P_1) + length(P_2), \\ proj_x(P) &= proj_x(P_1) = proj_x(P_2) = b. \end{aligned}$$

Therefore,

$$guil(P) \le 2 \cdot length(P) - proj_x(P).$$

Gonzalez and Zheng [8] improved this upper bound to 1.75 and conjectured that the performance ratio in this case is 1.5. □

Applications

In 1996, Arora [1] and Mitchell et al. [12, 14, 15] found that the cut does not necessarily have to be completely guillotine in order to have a polynomial-time computable optimal solution for such a sequence of cuts. Of course, the number of connections left by an incomplete guillotine cut should be limited. While Mitchell et al. developed the m-guillotine subdivision technique, Arora employed a "portal" technique. They also found that their techniques can be used for not only the MELRP, but also for many geometric optimization problems [1–3, 12–15].

Open Problems

One current important submicron step of technology evolution in electronics interconnects has become the dominating factor in determining VLSI performance and reliability. Historically a problem of interconnects design in VLSI has been very tightly intertwined with the classical problem in computational geometry: Steiner minimum tree generation. Some essential characteristics of VLSI are roughly proportional to the length of the interconnects. Such characteristics include chip area, yield, power consumption, reliability, and timing. For example, the area occupied by interconnects is proportional to their combined length and directly impacts the chip size. Larger chip size results in reduction of yield and increase in manufacturing cost. The costs of other components required for manufacturing also increase with the increase of the wire length. From the performance angle, longer interconnects cause an increase in power dissipation, degradation of timing, and other undesirable consequences. That is why finding the minimum length of interconnects consistent with other goals and constraints is such an important problem at this stage of VLSI technology.

The combined length of the interconnects on a chip is the sum of the lengths of individual signal nets. Each signal net is a set of electrically connected terminals, where one terminal acts as a driver and other terminals are receivers of electrical signals. Historically, for the purpose of finding an optimal configuration of interconnects, terminals were considered as points on the plane, and a routing problem for individual nets was formulated as a classical Steiner minimum tree problem. For a variety of reasons, VLSI technology implements only rectilinear wiring on the set of parallel planes, and, consequently, with few exceptions, only a rectilinear version of the Steiner tree is being considered in the VLSI domain. This problem is known as the RSMT.

Further progress in VLSI technology resulted in more factors than just length of interconnects gaining importance in selection of routing topologies. For example, the presence of obstacles led to reexamination of techniques used in studies of the rectilinear Steiner tree, since many classical techniques do not work in this new environment. To clarify the statement made above, we will consider the construction of a rectilinear Steiner minimum tree in the presence of obstacles.

Let us start with a rectilinear plane with obstacles defined as rectilinear polygons. Given n points on the plane, the objective is to find the shortest rectilinear Steiner tree that interconnects them. One already knows that a polynomial-time approximation scheme for RSMT without obstacles exists and can be constructed by adaptive partition with application of either the portal or the m-guillotine subdivision technique. However, both the m-guillotine cut and the portal techniques do not work in the case that obstacles exist. The portal technique is not applicable because obstacles may block the movement of the line that crosses the cut at a portal. The m-guillotine cut could not be constructed either, because obstacles may break down the cut segment that makes the Steiner tree connected.

In spite of the facts stated above, the RSMT with obstacles may still have polynomial-time approximation schemes. Strong evidence was given by Min et al. [11]. They constructed a polynomial-time approximation scheme for the problem with obstacles under the condition that the ratio of the longest edge and the shortest edge of the minimum spanning tree is bounded by a constant. This design is based on the classical nonadaptive partition approach. All of the above

make us believe that a new adaptive technique can be found for the case with obstacles.

Cross-References

- Metric TSP
- Rectilinear Steiner Tree
- Steiner Trees

Recommended Reading

1. Arora S (1996) Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. In: Proceedings of the 37th IEEE symposium on foundations of computer science, pp 2–12
2. Arora S (1997) Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In: Proceedings of the 38th IEEE symposium on foundations of computer science, pp 554–563
3. Arora S (1998) Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. J ACM 45:753–782
4. Du D-Z, Pan, L-Q, Shing, M-T (1986) Minimum edge length guillotine rectangular partition. Technical report 0241886, Math. Sci. Res. Inst., Univ. California, Berkeley
5. Du D-Z, Hsu DF, Xu K-J (1987) Bounds on guillotine ratio. Congressus Numerantium 58:313–318
6. Du DZ, Hwang FK, Shing MT, Witbold T (1988) Optimal routing trees. IEEE Trans Circuits 35:1335–1337
7. Gonzalez T, Zheng SQ (1985) Bounds for partitioning rectilinear polygons. In: Proceedings of the 1st symposium on computational geometry
8. Gonzalez T, Zheng SQ (1989) Improved bounds for rectangular and guillotine partitions. J Symb Comput 7:591–610
9. Lingas A (1983) Heuristics for minimum edge length rectangular partitions of rectilinear figures. In: Proceedings of the 6th GI-conference, Dortmund. Springer
10. Lingas A, Pinter RY, Rivest RL, Shamir A (1982) Minimum edge length partitioning of rectilinear polygons. In: Proceedings of the 20th Allerton conference on communication, control, and computing, Illinois
11. Min M, Huang SC-H, Liu J, Shragowitz E, Wu W, Zhao Y, Zhao Y (2003) An approximation scheme for the rectilinear Steiner minimum tree in presence of obstructions. Fields Inst Commun 37:155–164
12. Mitchell JSB (1996) Guillotine subdivisions approximate polygonal subdivisions: a simple new method for the geometric k-MST problem. In: Proceedings of the 7th ACM-SIAM symposium on discrete algorithms, pp 402–408
13. Mitchell JSB (1997) Guillotine subdivisions approximate polygonal subdivisions: part III – faster polynomial-time approximation scheme for geometric network optimization, manuscript, State University of New York, Stony Brook
14. Mitchell JSB (1999) Guillotine subdivisions approximate polygonal subdivisions: part II – a simple polynomial-time approximation scheme for geometric k-MST, TSP, and related problem. SIAM J Comput 29(2):515–544
15. Mitchell JSB, Blum A, Chalasani P, Vempala S (1999) A constant-factor approximation algorithm for the geometric k-MST problem in the plane. SIAM J Comput 28(3):771–781

Additive Spanners

Shiri Chechik
Department of Computer Science, Tel Aviv University, Tel Aviv, Israel

Keywords

Approximate shortest paths; Spanners; Stretch factor

Years and Authors of Summarized Original Work

2013; Chechik

Problem Definition

A spanner is a subgraph of a given graph that faithfully preserves the pairwise distances of that graph. Formally, an (α, β) spanner of a graph $G = (V, E)$ is a subgraph H of G such that for any pair of nodes x, y, $\mathbf{dist}(x, y, H) \leq \alpha \cdot \mathbf{dist}(x, y, G) + \beta$, where $\mathbf{dist}(x, y, H')$ for a subgraph H' is the distance of the shortest path from s to t in H'. We say that the spanner is *additive* if $\alpha = 1$, and if in addition $\beta = O(1)$, we say that the spanner is *purely additive*. If $\beta = 0$, we say that the spanner is *multiplicative*; otherwise, we say that the spanner is *mixed*.

Key Results

This section presents a survey on spanners with a special focus on additive spanners.

Graph spanners were first introduced in [12, 13] in the late 1980s and have been extensively studied since then.

Spanners are used as a key ingredient in many distributed applications, e.g., synchronizers [13], compact routing schemes [6, 14, 17], broadcasting [11], etc.

Much of the work on spanners considers multiplicative spanners. A well-known theorem on multiplicative spanners is that one can efficiently construct a $(2k-1,0)$ spanner with $O\left(n^{1+1/k}\right)$ edges [2]. Based in the girth conjecture of Erdős [10], this size-stretch ratio is conjectured to be optimal.

The problem of additive spanners was also extensively studied, but yet several key questions remain open. The girth conjecture does not contradict the existence of $(1,2k-2)$ spanners of size $O\left(n^{1+1/k}\right)$, or in fact any (α,β) spanners of size $O\left(n^{1+1/k}\right)$ such that $\alpha+\beta=2k-1$ with $\alpha\geq 1$ and $\beta>0$.

The first construction for purely additive spanners was introduced by Aingworth et al. [1]. It was shown in [1] how to efficiently construct a $(1,2)$ spanner, or a 2-*additive* spanner for short, with $O\left(n^{3/2}\right)$ edges (see also [8, 9, 16, 19] for further follow-up). Later, Baswana et al. [3, 4] presented an efficient construction for 6-additive spanners with $O\left(n^{4/3}\right)$ edges. Woodruff [21] further presented another construction for 6-additive spanners with $\tilde{O}\left(n^{4/3}\right)$ edges with improved construction time. Chechik [7] recently presented a new algorithm for $(1,4)$-additive spanners with $\tilde{O}\left(n^{7/5}\right)$ edges. These are the only three constructions known for purely additive spanners. Interestingly, Woodruff [20] presented lower bounds for additive spanners that match the girth conjecture bounds. These lower bounds do not rely on the correctness of the conjecture. More precisely, Woodruff [20] showed the existence of graphs for which any spanner of size $O\left(k^{-1}n^{1+1/k}\right)$ must have an additive stretch of at least $2k-1$.

In the absence of additional purely additive spanners, attempts were made to seek sparser spanners with nonconstant additive stretch. Bollobás et al. [5] showed how to efficiently construct a $\left(1,n^{1-2\delta}\right)$ spanner with $O\left(2^{1/\delta}n^{1+\delta}\right)$ edges for any $\delta>0$. Later, Baswana et al. in [3,4] improved this additive stretch to $\left(1,n^{1-3\delta}\right)$, and in addition, Pettie [15] improved the stretch to $\left(1,n^{9/16-7\delta/8}\right)$ (the latter is better than the former for every $\delta<7/34$).

Chechik [7] recently further improved the stretch for a specific range of δ. More specifically, Chechik presented a construction for additive spanners with $\tilde{O}\left(n^{1+\delta}\right)$ edges and $\tilde{O}\left(n^{1/2-3\delta/2}\right)$ additive stretch for any $3/17\leq\delta<1/3$. Namely, [7] decreased the stretch for this range to the root of the best previously known additive stretch.

Sublinear additive spanners, that is, spanners with additive stretch that is sublinear in the distances, were also studied. Thorup and Zwick [19] showed a construction of a $O\left(kn^{1+1/k}\right)$ size spanner such that for every pair of nodes s and t, the additive stretch is $O\left(d^{1-1/k}+2^k\right)$, where $d=\mathbf{dist}(s,t)$ is the distance between s and t. This was later improved by Pettie [15] who presented an efficient spanner construction with $O\left(kn^{1+\frac{(3/4)^{k-2}}{7-2(3/4)^{k-2}}}\right)$ size and $O\left(kd^{1-1/k}+k^k\right)$ additive stretch, where $d=\mathbf{dist}(s,t)$. Specifically, for $k=2$, the size of the spanner is $O\left(n^{6/5}\right)$ and the additive stretch is $O\left(\sqrt{d}\right)$.

Chechik [7] slightly improved the size of Pettie's [15] sublinear additive spanner with additive stretch $O(\sqrt{d})$ from $O\left(n^{1+1/5}\right)$ to $\tilde{O}\left(n^{1+3/17}\right)$.

Open Problems

A major open problem in the area of additive spanners is on the existence of purely additive spanners with $O\left(n^{1+\delta}\right)$ for any $\delta>0$. In particular, proving or disproving the existence of a spanner of size $O\left(n^{4/3-\epsilon}\right)$ for some constant ϵ with constant or even polylog additive stretch would be a major breakthrough.

Recommended Reading

1. Aingworth D, Chekuri C, Indyk P, Motwani R (1999) Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM J Comput 28(4):1167–1181
2. Althöfer I, Das G, Dobkin D, Joseph D, Soares J (1993) On sparse spanners of weighted graphs. Discret Comput Geom 9:81–100
3. Baswana S, Kavitha T, Mehlhorn K, Pettie S (2005) New constructions of (α, β)-spanners and purely additive spanners. In: Proceedings of the 16th symposium on discrete algorithms (SODA), Vancouver, pp 672–681
4. Baswana S, Kavitha T, Mehlhorn K, Pettie S (2010) Additive spanners and (α, β)-spanners. ACM Trans Algorithm 7:1–26. A.5
5. Bollobás B, Coppersmith D, Elkin M (2003) Sparse distance preservers and additive spanners. In: Proceedings of the 14th ACM-SIAM symposium on discrete algorithms (SODA), Baltimore, pp 414–423
6. Chechik S (2013) Compact routing schemes with improved stretch. In: Proceedings of the 32nd ACM symposium on principles of distributed computing (PODC), Montreal, pp 33–41
7. Chechik S (2013) New additive spanners. In: Proceedings of the 24th symposium on discrete algorithms (SODA), New Orleans, pp 498–512
8. Dor D, Halperin S, Zwick U (2000) All-pairs almost shortest paths. SIAM J Comput 29(5):1740–1759
9. Elkin M, Peleg D (2004) $(1 + \epsilon, \beta)$-spanner constructions for general graphs. SIAM J Comput 33(3):608–631
10. Erdős P (1964) Extremal problems in graph theory. In: Theory of graphs and its applications. Methuen, London, pp 29–36
11. Farley AM, Proskurowski A, Zappala D, Windisch K (2004) Spanners and message distribution in networks. Discret Appl Math 137(2):159–171
12. Peleg D, Schäffer AA (1989) Graph spanners. J Graph Theory 13:99–116
13. Peleg D, Ullman JD (1989) An optimal synchronizer for the hypercube. SIAM J Comput 18(4):740–747
14. Peleg D, Upfal E (1989) A trade-off between space and efficiency for routing tables. J ACM 36(3):510–530
15. Pettie S (2009) Low distortion spanners. ACM Trans Algorithms 6(1):1–22
16. Roditty L, Thorup M, Zwick U (2005) Deterministic constructions of approximate distance oracles and spanners. In: Proceedings of the 32nd international colloquium on automata, languages and programming (ICALP), Lisbon, pp 261–272
17. Thorup M, Zwick U (2001) Compact routing schemes. In: Proceedings of the 13th ACM symposium on parallel algorithms and architectures (SPAA), Heraklion, pp 1–10
18. Thorup M, Zwick U (2005) Approximate distance oracles. J ACM 52(1):1–24
19. Thorup M, Zwick U (2006) Spanners and emulators with sublinear distance errors. In: Proceedings of the 17th ACM-SIAM symposium on discrete algorithms (SODA), Miami, pp 802–809
20. Woodruff DP (2006) Lower bounds for additive spanners, emulators, and more. In: Proceedings of the 47th IEEE symposium on foundations of computer science (FOCS), Berkeley, pp 389–398
21. Woodruff DP (2010) Additive spanners in nearly quadratic time. In: Proceedings of the 37th international colloquium on automata, languages and programming (ICALP), Bordeaux, pp 463–474

Adwords Pricing

Tian-Ming Bu
Software Engineering Institute, East China Normal University, Shanghai, China

Keywords

Adword auction; Convergence; Dynamics; Equilibrium; Mechanism design

Years and Authors of Summarized Original Work

2007; Bu, Deng, Qi

Problem Definition

The model studied here is the same as that which is first presented in [10] by Varian. For some keyword, $\mathcal{N} = \{1, 2, \ldots, N\}$ advertisers bid $\mathcal{K} = \{1, 2, \ldots, K\}$ advertisement slots ($K < N$) which will be displayed on the search result page from top to bottom. The higher the advertisement is positioned, the more conspicuous it is and the more clicks it receives. Thus for any two slots $k_1, k_2 \in \mathcal{K}$, if $k_1 < k_2$, then slot k_1's click-through rate (CTR) c_{k_1} is larger than c_{k_2}. That is, $c_1 > c_2 > \cdots > c_K$, from top to bottom, respectively. Moreover, each bidder $i \in \mathcal{N}$ has privately known information, v^i, which represents the expected return of per click to bidder i.

According to each bidder i's submitted bid b^i, the auctioneer then decides how to distribute the advertisement slots among the bidders and how much they should pay for per click. In particular, the auctioneer first sorts the bidders in decreasing order according to their submitted bids. Then the highest slot is allocated to the first bidder, the second highest slot is allocated to the second bidder, and so on. The last $N - K$ bidders would lose and get nothing. Finally, each winner would be charged on a per-click basis for the next bid in the descending bid queue. The losers would pay nothing.

Let b_k denote the kth highest bid in the descending bid queue and v_k the true value of the kth bidder in the descending queue. Thus if bidder i got slot k, i's payment would be $b_{k+1} \cdot c_k$. Otherwise, his payment would be zero. Hence, for any bidder $i \in \mathcal{N}$, if i were on slot $k \in \mathcal{K}$, his/her utility (payoff) could be represented as

$$u_k^i = (v^i - b_{k+1}) \cdot c_k$$

Unlike one-round sealed-bid auctions where each bidder has only one chance to bid, the adword auction allows bidders to change their bids any time. Once bids are changed, the system refreshes the ranking automatically and instantaneously. Accordingly, all bidders' payment and utility are also recalculated. As a result, other bidders could then have an incentive to change their bids to increase their utility and so on.

Definition 1 (Adword Pricing)

INPUT: The CTR for each slot, each bidder's expected return per click on his/her advertising
OUTPUT: The stable states of this auction and whether any of these stable states can be reached from any initial states

Key Results

Let $\mathbf{b}$ represent the bid vector $(b^1, b^2, \ldots, b^N)$. $\forall i \in \mathcal{N}$, $\mathcal{O}^i(\mathbf{b})$ denotes bidder i's place in the descending bid queue. Let $\mathbf{b}^{-i} = (b^1, \ldots, b^{i-1}, b^{i+1}, \ldots, b^N)$ denote the bids of all other bidders except i. $\mathcal{M}^i(\mathbf{b}^{-i})$ returns a set defined as

$$\mathcal{M}^i(\mathbf{b}^{-i}) = \arg \max_{b^i \in [0, v^i]} \left\{ u^i_{\mathcal{O}^i(b^i, \mathbf{b}^{-i})} \right\} \quad (1)$$

Definition 2 (Forward-Looking Best-Response Function) Given $\mathbf{b}^{-i}$, suppose $\mathcal{O}^i(\mathcal{M}^i(\mathbf{b}^{-i}), \mathbf{b}^{-i}) = k$, then bidder i's forward-looking response function $\mathcal{F}^i(\mathbf{b}^{-i})$ is defined as

$$\mathcal{F}^i(\mathbf{b}^{-i}) = \begin{cases} v^i - \frac{c_k}{c_{k-1}}(v^i - b_{k+1}) & 2 \le k \le K \\ v^i & k = 1 \text{ or } k > K \end{cases} \quad (2)$$

Definition 3 (Forwarding-Looking Equilibrium) A forward-looking best-response function-based equilibrium is a strategy profile $\hat{\mathbf{b}}$ such that

$$\forall i \in \mathcal{N}, \hat{\mathbf{b}}^i \in \mathcal{F}^i(\hat{\mathbf{b}}^{-i})$$

Definition 4 (Output Truthful (Kao et al., 2006, Output truthful versus input truthful: a new concept for algorithmic mechanism design, unpublished) [7]) For any instance of adword auction and the corresponding equilibrium set $\mathcal{E}$, if $\forall \mathbf{e} \in \mathcal{E}$ and $\forall i \in \mathcal{N}$, $\mathcal{O}^i(\mathbf{e}) = \mathcal{O}^i(v^1, \ldots, v^N)$, then the adword auction is *output truthful* on $\mathcal{E}$.

Theorem 1 *An adword auction is output truthful on* $\mathcal{E}_{forward-looking}$.

Corollary 1 *An Adword auction has a unique forward-looking Nash equilibrium.*

Corollary 2 *Any bidder's payment under the forward-looking Nash equilibrium is equal to his/her payment under the VCG mechanism for the auction.*

Corollary 3 *For adword auctions, the auctioneer's revenue in a forward-looking Nash equilibrium is equal to his/her revenue under the VCG mechanism for the auction.*

Definition 5 (Simultaneous Readjustment Scheme) In a simultaneous readjustment scheme, all bidders participating in the auction

will use forward-looking best-response function $\mathcal{F}$ to update their current bids simultaneously, which turns the current stage into a new stage. Then based on the new stage, all bidders may update their bids again.

Theorem 2 *An adword auction may not always converge to a forward-looking Nash equilibrium under the simultaneous readjustment scheme even when the number of slots is 3. But the protocol converges when the number of slots is 2.*

Definition 6 (Round-Robin Readjustment Scheme) In the round-robin readjustment scheme, bidders update their biddings one after the other, according to the order of the bidder's number or the order of the slots.

Theorem 3 *An adword auction may not always converge to a forward-looking Nash equilibrium under the round-robin readjustment scheme even when the number of slots is 4. But the protocol converges when the number of slots is 2 or 3.*

1 Readjustment Scheme: Lowest-First$(K, j, b_1, b_2, \cdots, b_N)$

if $(j = 0)$ **then**
  exit
end if
Let i be the ID of the bidder whose current bid is b_j (and equivalently, b^i).
Let $h = \mathcal{O}^i(\mathcal{M}^i(\mathbf{b}^{-i}), \mathbf{b}^{-i})$.
Let $\mathcal{F}^i(\mathbf{b}^{-i})$ be the best response function value for Bidder i.
Re-sort the bid sequence. (So h is the slot of the new bid $\mathcal{F}^i(\mathbf{b}^{-i})$ of Bidder i.)
if $(h < j)$ **then**
  call Lowest-First $(K,j,b_1,b_2,\cdots,b_N)$,
else
  call Lowest-First$(K,h-1,b_1,b_2,\cdots,b_N)$
end if

Theorem 4 *Adword auctions converge to a forward-looking Nash equilibrium in finite steps with a* lowest-first *adjustment scheme.*

Theorem 5 *Adword auctions converge to a forward-looking Nash equilibrium with probability 1 under a randomized readjustment scheme.*

Applications

Online adword auctions are the fastest growing form of advertising. Many search engine companies such as Google and Yahoo! make huge profits on this kind of auction. Because advertisers can change their bids anytime, such auctions can reduce the advertisers' risk. Further, because the advertisement is only displayed to those people who are really interested in it, such auctions can reduce the advertisers' investment and increase their return on investment.

For the same model, Varian [10] focuses on a subset of Nash equilibria, called *Symmetric Nash Equilibria*, which can be formulated nicely and dealt with easily. Edelman et al. [8] study *locally envy-free* equilibria, where no player can improve his/her payoff by exchanging bids with the player ranked one position above him/her. Coincidently, locally envy-free equilibrium is equal to symmetric Nash equilibrium proposed in [10]. Further, the revenue under the forward-looking Nash equilibrium is the same as the lower bound under Varian's symmetric Nash equilibria and the lower bound under Edelman et al.'s locally envy-free equilibria. In [6], Cary et al. also study the dynamic model's equilibria and convergence based on the *balanced bidding strategy* which is actually the same as the *forward-looking best-response function* in [4]. Cary et al. explore the convergence properties under two models, a *synchronous* model which is the same as *simultaneous readjustment scheme* in [4] and an *asynchronous* model which is

the same as *randomized readjustment scheme* in [4].

In addition, there are other models for adword auctions. Abrams [1] and Bu et al. [5] study the model under which each bidder could submit his/her daily budget, even the maximum number of clicks per day, in addition to the price per click. Both [9] and [3] study bidders' behavior of bidding on several keywords. Aggarwal et al. [2] studies the model where the advertiser not only submits a bid but additionally submits which positions he/she is going to bid for.

Open Problems

The speed of convergence still remains open. Does the dynamic model converge in polynomial time under randomized readjustment scheme? Even more, are there other readjustment scheme that converge in polynomial time?

Cross-References

- Multiple Unit Auctions with Budget Constraint
- Position Auction

Recommended Reading

1. Abrams Z (2006) Revenue maximization when bidders have budgets. In: Proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithms (SODA-06), Miami, pp 1074–1082
2. Aggarwal G, Feldman J, Muthukrishnan S (2006) Bidding to the top: VCG and equilibria of position-based auctions. In: Proceedings of the 4th international workshop on approximation and online algorithms (WAOA-2006), Zurich, pp 15–28
3. Borgs C, Chayes J, Etesami O, Immorlica N, Jain K, Mahdian M (2006) Bid optimization in online advertisement auctions. In: 2nd workshop on sponsored search auctions (SSA2006), in conjunction with the ACM conference on electronic commerce (EC-06), Ann Arbor
4. Bu TM, Deng X, Qi Q (2008) Forward looking nash equilibrium for keyword auction. Inf Process Lett 105(2):41–46
5. Bu TM, Qi Q, Sun AW (2008) Unconditional competitive auctions with copy and budget constraints. Theor Comput Sci 393(1–3):1–13
6. Cary M, Das A, Edelman B, Giotis I, Heimerl K, Karlin AR, Mathieu C, Schwarz M (2007) Greedy bidding strategies for keyword auctions. In: Proceedings of the 8th ACM conference on electronic commerce (EC-2007), San Diego, pp 262–271
7. Chen X, Deng X, Liu BJ (2006) On incentive compatible competitive selection protocol. In: Proceedings of the 12th annual international computing and combinatorics conference (COCOON06), Taipei, pp 13–22
8. Edelman B, Ostrovsky M, Schwarz M (2007) Internet advertising and the generalized second price auction: selling billions of dollars worth of keywords. Am Econ Rev 97(1):242–259
9. Kitts B, Leblanc B (2004) Optimal bidding on keyword auctions. Electron Mark Spec Issue Innov Auction Mark 14(3):186–201
10. Varian HR (2007) Position auctions. Int J Ind Organ 25(6):1163–1178

A

Algorithm DC-TREE for *k*-Servers on Trees

Marek Chrobak
Computer Science, University of California, Riverside, CA, USA

Keywords

Competitive analysis; K-server problem; On-line algorithms; Trees

Years and Authors of Summarized Original Work

1991; Chrobak, Larmore

Problem Definition

In the *k-Server Problem*, one wishes to schedule the movement of k-servers in a metric space $\mathbb{M}$, in response to a sequence $\varrho = r_1, r_2, \ldots, r_n$ of *requests*, where $r_i \in \mathbb{M}$ for each i. Initially, all the servers are located at some initial configuration $X_0 \subseteq \mathbb{M}$ of k points. After each request r_i is issued, one of the k-servers must move

to r_i. A *schedule* specifies which server moves to each request. The *cost* of a schedule is the total distance traveled by the servers, and our objective is to find a schedule with minimum cost.

In the *online* version of the k-Server Problem, the decision as to which server to move to each request r_i must be made before the next request r_{i+1} is issued. In other words, the choice of this server is a function of requests $r_1, r_2, \ldots, r_i$. It is quite easy to see that in this online scenario, it is not possible to compute an optimal schedule for each request sequence, raising the question of how to measure the accuracy of such online algorithms. A standard approach to doing this is based on competitive analysis. If $\mathcal{A}$ is an online k-server algorithm denote by $cost_{\mathcal{A}}(\varrho)$ the cost of the schedule produced by $\mathcal{A}$ on a request sequence ϱ, and by $opt(\varrho)$ the cost of the optimal schedule. $\mathcal{A}$ is called *R-competitive* if $cost_{\mathcal{A}}(\varrho) \leq R \cdot opt(\varrho) + B$, where B is a constant that may depend on $\mathbb{M}$ and X_0. The smallest such R is called the *competitive ratio* of $\mathcal{A}$.

The k-Server Problem was introduced by Manasse, McGeoch, and Sleator [7, 8], who proved that there is no online R-competitive algorithm for $R < k$, for any metric space with at least $k + 1$ points. They also gave a 2-competitive algorithm for $k = 2$ and formulated what is now known as the *k-Server Conjecture*, which postulates that there exists a k-competitive online algorithm for all k. Koutsoupias and Papadimitriou [5, 6] proved that the so-called *Work-Function Algorithm* has competitive ratio at most $2k - 1$, which to date remains the best upper bound known.

Efforts to prove the k-Server Conjecture led to discoveries of k-competitive algorithms for some restricted classes of metric spaces, including Algorithm DC-TREE for trees [3] presented in this entry. (See [1, 2, 4] for other examples.) A *tree* is a metric space defined by a connected acyclic graph whose edges are treated as line segments of arbitrary positive lengths. This metric space includes both the tree's vertices and the points on the edges, and the distances are measured along the (unique) shortest paths.

Key Results

Let $\mathbb{T}$ be a tree, as defined above. Given the current server configuration $S = \{s_1, \ldots, s_k\}$, where s_j denotes the location of server j, and a request point r, the algorithm will move several servers, with one of them ending up on r. For two points $x, y \in \mathbb{T}$, let $[x, y]$ be the unique path from x to y in $\mathbb{T}$. A server j is called *active* if there is no other server in $[s_j, r] - \{s_j\}$, and j is the minimum-index server located on s_j (the last condition is needed only to break ties).

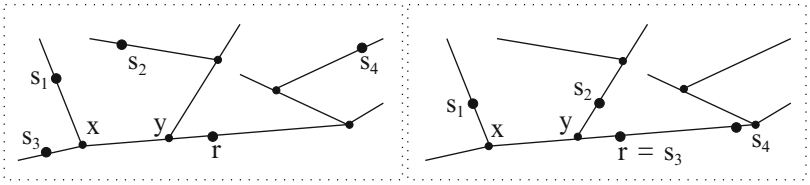
Algorithm DC-TREE. On a request r, move all active servers, continuously and with the same speed, towards r, until one of them reaches the request. Note that during this process some active servers may become inactive, in which case they halt. Clearly, the server that will arrive at r is the one that was closest to r at the time when r was issued.

More formally, denoting by s_j the variable representing the current position of server j, the algorithm serves r as follows:

while $s_j \neq r$ for all j **do**
  let $\delta = \frac{1}{2} \min_{i<j} \{d(s_i, s_j) + d(s_i, r) - d(s_j, r)\}$
  move each active server s_j by distance δ towards r

The example below shows how DC-TREE serves a request r (Fig. 1).

The competitive analysis of Algorithm DC-TREE is based on a potential argument. The cost of Algorithm DC-TREE is compared to that of an adversary who serves the requests with her own servers. Denoting by A the configuration of the adversary servers at a given step, define the potential by $\Phi = k \cdot D(S, A) + \sum_{i<j} d(s_i, s_j)$, where $D(S, A)$ is the cost of the minimum matching between S and A. At each step, the adversary first moves one of her servers to r. In this substep the potential increases by at most k times the increase of the adversary's cost. Then, Algorithm DC-TREE serves the request. One can show that then the sum of Φ and DC-TREE's cost does not increase. These two facts, by amortization over



Algorithm DC-TREE for *k*-Servers on Trees, Fig. 1 Algorithm DC-TREE serving a request on r. The configuration before r is issued is on the left; the configuration after the service is completed is on the right. At first, all servers are active. When server 3 reaches point x, server 1 becomes inactive. When server 3 reaches point y, server 2 becomes inactive

the whole request sequence, imply the following result [3]:

Theorem 1 ([3]) *Algorithm* DC-TREE *is* k*-competitive on trees.*

Applications

The k-Server Problem is an abstraction of various scheduling problems, including emergency crew scheduling, caching in multilevel memory systems, or scheduling head movement in 2-headed disks. Nevertheless, due to its abstract nature, the k-server problem is mainly of theoretical interest.

Algorithm DC-TREE can be applied to other spaces by "embedding" them into trees. For example, a uniform metric space (with all distances equal 1) can be represented by a star with arms of length $1/2$, and thus Algorithm DC-TREE can be applied to those spaces. This also immediately gives a k-competitive algorithm for the *caching problem*, where the objective is to manage a two-level memory system consisting of a large main memory and a cache that can store up to k memory items. If an item is in the cache, it can be accessed at cost 0, otherwise it costs 1 to read it from the main memory. This caching problem can be thought of as the k-server problem in a uniform metric space where the server positions represent the items residing in the cache. This idea can be extended further to the *weighted caching* [4], which is a generalization of the caching problem where different items may have different costs. In fact, if one can embed a metric space $\mathbb{M}$ into a tree with distortion bounded by δ, then Algorithm DC-TREE yields a δk-competitive algorithm for $\mathbb{M}$.

Open Problems

The k-Server Conjecture – whether there is a k-competitive algorithm for k-servers in any metric space – remains open. It would be of interest to prove it for some natural special cases, for example the plane, either with the Euclidean or Manhattan metric. (A k-competitive algorithm for the Manhattan plane for $k = 2, 3$ servers is known [1], but not for $k \geq 4$).

Very little is known about online *randomized* algorithms for k-servers. In fact, even for $k = 2$ it is not known if there is a randomized algorithm with competitive ratio smaller than 2.

Cross-References

- Deterministic Searching on the Line
- Generalized Two-Server Problem
- Metrical Task Systems
- Online Paging and Caching
- Work-Function Algorithm for k-Servers

Recommended Reading

1. Bein W, Chrobak M, Larmore LL (2002) The 3-server problem in the plane. Theor Comput Sci 287: 387–391
2. Borodin A, El-Yaniv R (1998) Online computation and competitive analysis. Cambridge University Press, Cambridge
3. Chrobak M, Larmore LL (1991) An optimal online algorithm for k servers on trees. SIAM J Comput 20:144–148
4. Chrobak M, Karloff H, Payne TH, Vishwanathan S (1991) New results on server problems. SIAM J Discret Math 4:172–181

5. Koutsoupias E, Papadimitriou C (1994) On the k-server conjecture. In: Proceedings of the 26th symposium on theory of computing (STOC). ACM, Montreal, pp 507–511
6. Koutsoupias E, Papadimitriou C (1995) On the k-server conjecture. J ACM 42:971–983
7. Manasse M, McGeoch LA, Sleator D (1988) Competitive algorithms for online problems. In: Proceedings of the 20th symposium on theory of computing (STOC). ACM, Chicago, pp 322–333
8. Manasse M, McGeoch LA, Sleator D (1990) Competitive algorithms for server problems. J Algorithms 11:208–230

Algorithmic Cooling

Tal Mor
Department of Computer Science, Technion – Israel Institute of Technology, Haifa, Israel

Keywords

Cooling; Data compression; Nuclear magnetic resonance; Quantum computing; Spin cooling; State initialization

Years and Authors of Summarized Original Work

1999; Schulman, Vazirani
2002; Boykin, Mor, Roychowdhury, Vatan, Vrijen

Problem Definition

The fusion of concepts taken from the fields of quantum computation, data compression, and thermodynamics has recently yielded novel algorithms that resolve problems in nuclear magnetic resonance and potentially in other areas as well, algorithms that "cool down" physical systems.

- A leading candidate technology for the construction of quantum computers is nuclear magnetic resonance (NMR). This technology has the advantage of being well established for other purposes, such as chemistry and medicine. Hence, it does not require new and exotic equipment, in contrast to ion traps and optical lattices, to name a few. However, when using standard NMR techniques, (not only for quantum computing purposes) one has to live with the fact that the state can only be initialized in a very noisy manner: The particles' spins point in mostly random directions, with only a tiny bias towards the desired state.

  The key idea of Schulman and Vazirani [27] is to combine the tools of both data compression and quantum computation, to suggest a *scalable* state initialization process, a "molecular-scale heat engine." Based on Schulman and Vazirani's method, Boykin, Mor, Roychowdhury, Vatan, and Vrijen [4] then developed a new process, "heat-bath algorithmic cooling," to significantly improve the state initialization process, by opening the system to the environment. Strikingly, this offered a way to put to good use the phenomenon of decoherence, which is usually considered to be the villain in quantum computation. These two methods are now sometimes called "closed-system" (or "reversible"), algorithmic cooling, and "open-system" algorithmic cooling, respectively.
- The far-reaching consequence of this research lies in the possibility of reaching beyond the potential implementation of remote-future quantum computing devices. An efficient technique to generate ensembles of spins that are highly polarized by external magnetic fields is considered to be a Holy Grail in NMR spectroscopy. Spin-half nuclei have steady-state polarization biases that increase inversely with temperature; therefore, spins exhibiting polarization biases above their thermal-equilibrium biases are considered *cool*. Such cooled spins present an improved signal-to-noise ratio if used in NMR spectroscopy or imaging.

  Existing spin-cooling techniques are limited in their efficiency and usefulness. Algorithmic cooling is a promising new spin-cooling approach that employs data compression methods in *open systems*.

It reduces the entropy of spins to a point far beyond Shannon's entropy bound on reversible entropy manipulations, thus increasing their polarization biases. As a result, it is conceivable that the open-system algorithmic cooling technique could be harnessed to improve on *current uses of NMR* in areas such as chemistry, material science, and even medicine, since NMR is at the basis of MRI – magnetic resonance imaging.

Basic Concepts

Loss-Less In-Place Data Compression

Given a bit string of length n, such that the probability distribution is known and far enough from the uniform distribution, one can use data compression to generate a shorter string, say of m bits, such that the entropy of each bit is much closer to one. As a simple example, consider a four-bit string which is distributed as follows: $p_{0001} = p_{0010} = p_{0100} = p_{1000} = 1/4$, with p_i the probability of the string i. The probability of any other string value is exactly zero, so the probabilities sum up to one. Then, the bit string can be compressed, via a lossy compression algorithm, into a 2-bit string that holds the binary description of the location of "1" in the above four strings. One can also envision a similar process that generates an output which is of the same length n as the input, but such that the entropy is compressed via a loss-less, in-place, data compression into the last two bits. For instance, logical gates that operate on the bits can perform the permutation $0001 \rightarrow 0000$, $0010 \rightarrow 0001$, $0100 \rightarrow 0010$, and $1000 \rightarrow 0011$, while the other input strings transform to output strings in which the two most significant bits are not zero; for instance, $1100 \rightarrow 1010$. One can easily see that the entropy is now fully concentrated on the two least significant bits, which are useful in data compression, while the two most significant bits have zero entropy.

In order to gain some intuition about the design of logical gates that perform entropy manipulations, one can look at a closely related scenario which was first considered by von Neumann. He showed a method to extract fair coin flips, given a biased coin; he suggested taking a pair of biased coin flips, with results a and b, and using the value of a conditioned on $a \neq b$. A simple calculation shows that $a = 0$ and $a = 1$ are now obtained with equal probabilities, and therefore, the entropy of coin a is increased in this case to 1. The opposite case, the probability distribution of a given that $a = b$, results in a highly determined coin flip, namely, a (conditioned) coin flip with a higher bias or lower entropy. A gate that flips the value of b if (and only if) $a = 1$ is called a controlled NOT gate. If after applying such a gate $b = 1$ is obtained, this means that $a \neq b$ prior to the gate operation; thus, now the entropy of a is 1. If, on the other hand, after applying such a gate $b = 0$ is obtained, this means that $a = b$ prior to the gate operation; thus, the entropy of a is now lower than its initial value.

Spin Temperature, Polarization Bias, and Effective Cooling

In physics, two-level systems, namely, systems that possess only binary values, are useful in many ways. Often it is important to initialize such systems to a pure state "0" or to a probability distribution which is as close as possible to a pure state "0." In these physical two-level systems, a data compression process that brings some of them closer to a pure state can be considered as "cooling." For quantum two-level systems, there is a simple connection between temperature, entropy, and population probability. The population difference between these two levels is known as the polarization bias, ϵ. Consider a single spin-half particle – for instance, a hydrogen nucleus – in a constant magnetic field. At equilibrium with a thermal heat-bath, the probability of this spin to be up or down (i.e., parallel or antiparallel to the field direction) is given by $p_\uparrow = \frac{1+\epsilon}{2}$ and $p_\downarrow = \frac{1-\epsilon}{2}$. The entropy H of the spin is $H(\text{single}-\text{bit}) = H(1/2 + \epsilon/2)$ with $H(P) \equiv -P \log_2 P - (1-P)\log_2(1-P)$ measured in bits. The two pure states of a spin-half nucleus are commonly written as $|\uparrow\rangle \equiv$"0" and $|\downarrow\rangle \equiv$"1"; the $|\rangle$ notation will be clarified elsewhere. (Quantum Computing entries in this encyclopedia.) The polarization bias of the spin at

thermal equilibrium is given by $\epsilon = p_\uparrow - p_\downarrow$. For such a physical System, the bias is obtained via a quantum statistical mechanics argument, $\epsilon = \tanh\left(\frac{\hbar\gamma B}{2K_B T}\right)$, where $\hbar$ is Planck's constant, B is the magnetic field, γ is the particle-dependent gyromagnetic constant,(This constant, γ, is thus responsible for the difference in equilibrium polarization bias [e.g., a hydrogen nucleus is 4 times more polarized than a carbon isotope ^{13}C nucleus, but about 10^3 less polarized than an electron spin].) K_B is Boltzman's coefficient, and T is the thermal heat-bath temperature. For high temperatures or small biases $\epsilon \approx \frac{\hbar\gamma B}{2K_B T}$; thus, the bias is inversely proportional to the temperature. Typical values of ϵ for spin-half nuclei at room temperature (and magnetic field of $\sim$10 T) are 10^{-5}–10^{-6}, and therefore, most of the analysis here is done under the assumption that $\epsilon \ll 1$. The spin temperature at equilibrium is thus $T = \frac{\text{Const}}{\epsilon}$, and its (Shannon) entropy is $H = 1 - (\epsilon^2/\ln 4)$.

A spin temperature out of thermal equilibrium is still defined via the same formulas. Therefore, when a system is moved away from thermal equilibrium, achieving a greater polarization bias is equivalent to cooling the spins, *without cooling the system*, and to decreasing their entropy. The process of increasing the bias (reducing the entropy) without increasing the temperature of the thermal bath is known as "effective cooling." After a typical period of time, termed the thermalization time or relaxation time, the bias will gradually revert to its thermal equilibrium value; yet during this process, typically in the order of seconds, the effectively cooled spin may be used for various purposes as described in section "Applications."

Consider a molecule that contains n adjacent spin-half nuclei arranged in a line; these form the bits of the string. These spins are initially at thermal equilibrium due to their interaction with the environment. At room temperature, the bits at thermal equilibrium are not correlated to their neighbors on the same string: more precisely, the correlation is very small and can be ignored. Furthermore, in a liquid state one can also neglect the interaction between strings (between molecules). It is convenient to write the probability distribution of a single spin at thermal equilibrium using the "density-matrix" notation

$$\rho_\epsilon = \begin{pmatrix} p_\uparrow & 0 \\ 0 & p_\downarrow \end{pmatrix} = \begin{pmatrix} (1+\epsilon)/2 & 0 \\ 0 & (1-\epsilon)/2 \end{pmatrix}, \tag{1}$$

since these two-level systems are of a quantum nature (namely, these are quantum bits – qubits) and, in general, can also have states other than just a classical probability distribution over "0" and "1." The classical case will now be considered, where ρ contains only diagonal elements, and these describe a conventional probability distribution. At thermal equilibrium, the state of $n = 2$ uncorrelated qubits that have the same polarization bias is described by the density matrix $\rho_{\text{init}}^{\{n=2\}} = \rho_\epsilon \otimes \rho_\epsilon$, where $\otimes$ means tensor product. The probability of the state "00," for instance, is then $(1+\epsilon)/2 \times (1+\epsilon)/2 = (1+\epsilon)^2/4$, etc. Similarly, the initial state of an n-qubit system of this type, at thermal equilibrium, is

$$\rho_{\text{init}}^{\{n\}} = \rho_\epsilon \otimes \rho_\epsilon \otimes \cdots \otimes \rho_\epsilon. \tag{2}$$

This state represents a thermal probability distribution, such that the probability of the classical state "000…0" is $P_{000\ldots0} = (1+\epsilon_0)^n/2^n$, etc. In reality, the initial bias is not the same on each qubit,(Furthermore, individual addressing of each spin during the algorithm requires a slightly different bias for each.) but as long as the differences between these biases are small (e.g., all qubits are of the same nucleus), these differences can be ignored in a discussion of an idealized scenario.

Key Results

Molecular-Scale Heat Engines

Schulman and Vazirani (SV) [27] identified the importance of in-place loss-less data compression and of the low-entropy bits created in that process: physical two-level systems (e.g., spin-half nuclei) may be similarly cooled by data compression algorithms. SV analyzed the cooling of such a system using various tools of data compression. A loss-less compression of an n-bit binary string distributed according to the thermal

equilibrium distribution, Eq. 2, is readily analyzed using information-theoretical tools: In an ideal compression scheme (not necessarily realizable), with sufficiently large n, all randomness – and hence all the entropy – of the bit string is transferred to $n - m$ bits; the remaining m bits are thus left, with extremely high probability, at a known deterministic state, say the string "000...0." The entropy H of the entire system is $H(\text{system}) = nH(\text{single} - \text{bit}) = nH(1/2 + \epsilon/2)$. Any compression scheme cannot decrease this entropy; hence, Shannon's source coding entropy bound yields $m \leq n[1 - H(1/2 + \epsilon/2)]$. A simple leading-order calculation shows that m is bounded by (approximately) $\frac{\epsilon^2}{2\ln 2}n$ for small values of the initial bias ϵ. Therefore, with typical $\epsilon \sim 10^{-5}$, molecules containing an order of magnitude of 10^{10} spins are required to cool a single spin close to zero temperature.

Conventional methods for NMR quantum computing are based on unscalable state initialization schemes [7, 14] (e.g., the "pseudo-pure-state" approach) in which the signal-to-noise ratio falls exponentially with n, the number of spins. Consequently, these methods are deemed inappropriate for future NMR quantum computers. SV [27] were first to employ tools of information theory to address the scaling problem; they presented a compression scheme in which the number of cooled spins scales well (namely, a constant times n). SV also demonstrated a scheme approaching Shannon's entropy bound, for very large n. They provided detailed analyses of three cooling algorithms, each useful for a different regime of ϵ values.

Some ideas of SV were already explored a few years earlier by Sørensen [29], a physical chemist who analyzed effective cooling of spins. He considered the entropy of several spin systems and the limits imposed on cooling these systems by polarization transfer and more general polarization manipulations. Furthermore, he considered spin-cooling processes in which only unitary operations were used, wherein unitary matrices are applied to the density matrices; such operations are realizable, at least from a conceptual point of view. Sørensen derived a stricter bound on unitary cooling, which today bears his name. Yet, unlike SV, he did not infer the connection to data compression or advocate compression algorithms.

SV named their concept "molecular-scale heat engine." When combined with conventional polarization transfer (which is partially similar to a SWAP gate between two qubits), the term "reversible polarization compression (RPC)" is more descriptive.

Heat-Bath Algorithmic Cooling

The next significant development came when Boykin, Mor, Roychowdhury, Vatan, and Vrijen (hereinafter referred to as BMRVV), invented a new spin-cooling technique, which they named *Algorithmic cooling* [4] or more specifically heat-bath algorithmic cooling in which the use of controlled interactions with a heat bath enhances the cooling techniques much further. Algorithmic cooling (AC) expands the effective cooling techniques by exploiting entropy manipulations in *open systems*. It combines RPC steps (When the entire process is RPC, namely, any of the processes that follow SV ideas, one can refer to it as reversible AC or closed-system AC, rather than as RPC.) with fast relaxation (namely, thermalization) of the *hotter spins*, as a way of pumping entropy outside the system and cooling the system *much beyond Shannon's entropy bound*. In order to pump entropy out of the system, AC employs regular spins (here called computation spins) together with rapidly relaxing spins. The latter are auxiliary spins that return to their thermal equilibrium state very rapidly. These spins have been termed "reset spins," or, equivalently, reset bits. The controlled interactions with the heat bath are generated by polarization transfer or by standard algorithmic techniques (of data compression) that transfer the entropy onto the reset spins which then lose this excess entropy into the environment.

The ratio $R_{\text{relax}-\text{times}}$, between the relaxation time of the computation spins and the relaxation time of the reset spins, must satisfy $R_{\text{relax}-\text{times}} \gg 1$. This condition is vital if one wishes to perform many cooling steps on the system to obtain significant cooling.

In a pure information-theoretical point of view, it is legitimate to assume that the only restriction on ideal RPC steps is Shannon's entropy bound; then the equivalent of Shannon's entropy bound, when an ideal open-system AC is used, is that all computation spins can be cooled down to zero temperature, that is to $\epsilon = 1$. Proof: repeat the following till the entropy of all computation spins is exactly zero: (i) push entropy from computation spins into reset spins and (ii) let the reset spins cool back to room temperature. Clearly, each application of step (i), except the last one, pushes the same amount of entropy onto the reset spins, and then this entropy is removed from the system in step (ii). Of course, a realistic scenario must take other parameters into account such as finite relaxation-time ratios, realistic environment, and physical operations on the spins. Once this is done, cooling to zero temperature is no longer attainable. While finite relaxation times and a realistic environment are system dependent, the constraint of using physical operations is conceptual.

BMRVV therefore pursued an algorithm that follows some physical rules; it is performed by unitary operations and reset steps and still bypass Shannon's entropy bound, by far. The BMRVV cooling algorithm obtains significant cooling beyond that entropy bound by making use of very long molecules bearing hundreds or even thousands of spins, because its analysis relies on the law of large numbers.

Practicable Algorithmic Cooling

The concept of algorithmic cooling then led to practicable algorithms [13] for cooling *small molecules*. In order to see the impact of practicable algorithmic cooling, it is best to use a different variant of the entropy bound. Consider a system containing n spin-half particles with total entropy higher than $n - 1$, so that there is no way to cool even one spin to zero temperature. In this case, the entropy bound is a result of the compression of the entropy into $n - 1$ fully random spins, so that the remaining entropy on the last spin is minimal. The entropy of the remaining single spin satisfies $H(\text{single}) \geq 1 - n\epsilon^2/\ln 4$; thus, at most, its polarization can be improved to

$$\epsilon_{\text{final}} \leq \epsilon\sqrt{n}\,. \tag{3}$$

The practicable algorithmic cooling (PAC), suggested by Fernandez, Lloyd, Mor, and Roychowdhury in [13], indicated potential for a near-future application to NMR spectroscopy. In particular, it presented an algorithm named PAC2 which uses any (odd) number of spins n, such that one of them is a reset spin, and $(n-1)$ are computation spins. PAC2 cools the spins such that the coldest one can (approximately) reach a bias amplification by a factor of $(3/2)^{(n-1)/2}$. The approximation is valid as long as the final bias $(3/2)^{(n-1)/2}\ \epsilon$ is much smaller than 1. Otherwise, a more precise treatment must be done. This proves an exponential advantage of AC over the best possible reversible AC, as these reversible cooling techniques, e.g., of [27, 29], are limited to improve the bias by no more than a factor of $\sqrt{n}$. PAC can be applied for small n (e.g., in the range of 10–20), and therefore, it is potentially suitable for near-future applications [9, 13, 19] in chemical and biomedical usages of NMR spectroscopy.

It is important to note that in typical scenarios, the initial polarization bias of a reset spin is higher than that of a computation spin. In this case, the bias amplification factor of $(3/2)^{(n-1)/2}$ is relative to the larger bias, that of the reset spin.

Exhaustive Algorithmic Cooling

Next, AC was analyzed, wherein the cooling steps (reset and RPC) are repeated an arbitrary number of times. This is actually an idealization where an unbounded number of reset and logic steps can be applied without error or decoherence, while the computation qubits do not lose their polarization biases. Fernandez [12] considered two computation spins and a single reset spin (the least significant bit, namely, the qubit at the right in the tensor-product density-matrix notation) and analyzed optimal cooling of this system. By repeating the reset and compression

exhaustively, he realized that the bound on the final biases of the three spins is approximately $\{2, 1, 1\}$ in units of ϵ, the polarization bias of the reset spin.

Mor and Weinstein generalized this analysis further and found that $n-1$ computation spins and a single reset spin can be cooled (approximately) to biases according to the Fibonacci series: $\{\ldots 34, 21, 13, 8, 5, 3, 2, 1, 1\}$. The computation spin that is further away from the reset spin can be cooled up to the relevant Fibonacci number F_n. That approximation is valid as long as the largest term times ϵ is still much smaller than 1. Schulman then suggested the "partner pairing algorithm" (PPA) and proved the optimality of the PPA among all *classical and quantum* algorithms. These two algorithms, the Fibonacci AC and the PPA, led to two joint papers [25, 26], where upper and lower bounds on AC were also obtained. The PPA is defined as follows: repeat these two steps until cooling sufficiently close to the limit: (a) RESET, applied to a reset spin in a system containing $n-1$ computation spins and a single (the LSB) reset spin, and (b) SORT, a permutation that sorts the 2^n diagonal elements of the density matrix by decreasing order, so that the MSB spin becomes the coldest. Two important theorems proven in [26] are:

(1) Lower bound: When $\epsilon 2^n \gg 1$ (namely, for long enough molecules), Theorem 3 in [26] promises that $n - \log(1/\epsilon)$ cold qubits can be extracted. This case is relevant for scalable NMR quantum computing.
(2) Upper bound: Section 4.2 in [26] proves the following theorem: No algorithmic cooling method can increase the probability of any basis state to above $\min\{2^{-n}e^{2^n\epsilon}, 1\}$, wherein the initial configuration is the completely mixed state (the same is true if the initial state is a thermal state).

More recently, Elias, Fernandez, Mor, and Weinstein [9] analyzed more closely the case of $n < 15$ (at room temperature), where the coldest spin (at all stages) still has a polarization bias much smaller than 1. This case is most relevant for near-future applications in NMR spectroscopy. They generalized the Fibonacci-AC to algorithms yielding higher-term Fibonacci series, such as the tribonacci (also known as 3-term Fibonacci series), $\{\ldots 81, 44, 24, 13, 7, 4, 2, 1, 1\}$, etc. The ultimate limit of these multi-term Fibonacci series is obtained when each term in the series is the sum of all previous terms. The resulting series is precisely the exponential series $\{\ldots 128, 64, 32, 16, 8, 4, 2, 1, 1\}$, so the coldest spin is cooled by a factor of 2^{n-2}. Furthermore, a leading-order analysis of the upper bound mentioned above (Section 4.2 in Ref. [26]) shows that no spin can be cooled beyond a factor of 2^{n-1}; see Corollary 1 in [9].

Other Results

For several other theoretical results dealing with relevant algorithms and with the connection to thermodynamics, see [11, 15, 17, 21]. For several popular "News and Views" discussions of AC in Nature, see [18, 22, 24].

Applications

The two major far-future and near-future applications are already described in section "Problem Definition." It is important to add here that although the specific algorithms analyzed so far for AC are usually classical, their practical implementation via an NMR spectrometer must be done through analysis of universal quantum computation, using the specific gates allowed in such systems. Therefore, AC could yield the first near-future application of quantum computing devices.

AC may also be useful for cooling various other physical systems; for several examples (theoretical and experimental), see [2, 16, 28, 30, 31], since state initialization is a common problem in physics in general and in quantum computation in particular.

Open Problems

A main open problem in practical AC is technological; can the ratio of relaxation

times be increased so that many cooling steps may be applied onto relevant NMR systems? Other methods, for instance, a spin-diffusion mechanism [3, 23], may also be useful for various applications.

Another interesting open problem is whether the ideas developed during the design of AC can also lead to applications in classical information theory.

Last but not least, in the context of building scalable quantum computers, it is interesting to study if AC can become a practical tool for advancing the non-conventional model of quantum computing called the one pure qubit (or one clean qubit) model as suggested in [1, 8] and to study if AC can be useful for designing fault-tolerant quantum computers as suggested in [20].

Experimental Results

Various ideas of AC had already led to several experiments using 3–4 qubit quantum computing devices in NMR (AC used in other systems was mentioned earlier in section "Applications"):

(1) An experiment [6] that implemented a single RPC step.
(2) Two experiments [5, 10] in which entropy-conservation bounds (which apply in any closed system) were bypassed. The second one [10] was done on bio-molecules – amino acids.
(3) A full AC experiment [3] that includes the initialization of three carbon nuclei to the bias of a hydrogen spin, followed by a single compression step on these three carbons. This work was later on extended also to multi-cycle AC [23].

Cross-References

Quantum computing entries such as ▸ Quantum Algorithm for Factoring, ▸ Quantum Algorithm for the Parity Problem and ▸ Quantum Key Distribution. Data compression entries such as ▸ Dictionary-Based Data Compression.

Recommended Reading

1. Ambainis A, Schulman LJ, Vazirani U (2006) Computing with highly mixed states. J ACM 53: 507–531
2. Bakr WS, Preiss PM, Tai ME, Ma R, Simon J, Greiner M (2011) Orbital excitation blockade and algorithmic cooling in quantum gases. Nature 480:500–503
3. Baugh J, Moussa O, Ryan CA, Nayak A, Laflamme R (2005) Experimental implementation of heat-bath algorithmic cooling using solid-state nuclear magnetic resonance. Nature 438:470–473
4. Boykin PO, Mor T, Roychowdhury V, Vatan F, Vrijen R (2002) Algorithmic cooling and scalable NMR quantum computers. Proc Natl Acad Sci USA 99:3388–3393
5. Brassard G, Elias Y, Fernandez JM, Gilboa H, Jones JA, Mor T, Weinstein Y, Xiao L (2005) Experimental heat-bath cooling of spins. Submitted to Euro Phys Lett PLUS. See also arXiv:0511156 [quant-ph]. See a much improved version in arXiv:1404.6885 [quant-ph], 2014
6. Chang DE, Vandersypen LMK, Steffen M (2001) NMR implementation of a building block for scalable quantum computation. Chem Phys Lett 338: 337–344
7. Cory DG, Fahmy AF, Havel TF (1997) Ensemble quantum computing by NMR spectroscopy. Proc Natl Acad Sci USA 94:1634–1639
8. Datta A, Flammia ST, Caves CM (2005) Entanglement and the power of one qubit. Phys Rev A 72:042316
9. Elias Y, Fernandez JM, Mor T, Weinstein Y (2007) Optimal algorithmic cooling of spins. Isr J Chem 46:371–391
10. Elias Y, Gilboa H, Mor T, Weinstein Y (2011) Heat-bath cooling of spins in two amino acids. Chem Phys Lett 517:126–131
11. Elias Y, Mor T, Weinstein Y (2011) Semioptimal practicable algorithmic cooling. Phys Rev A 83:042340
12. Fernandez JM (2004) De computatione quantica. Ph.D. Dissertation, University of Montreal, Montreal
13. Fernandez JM, Lloyd S, Mor T, Roychowdhury V (2004) Practicable algorithmic cooling of spins. Int J Quantum Inf 2:461–477
14. Gershenfeld NA, Chuang IL (1997) Bulk spin-resonance quantum computation. Science 275:350–356
15. Kaye P (2007) Cooling algorithms based on the 3-bit majority. Quantum Inf Proc 6:295–322
16. Ladd TD, Goldman JR, Yamaguchi F, Yamamoto Y, Abe E, Itoh KM (2002) All-Silicon quantum computer. Phys Rev Lett 89:017901
17. Linden N, Popescu P, Skrzypczyk P (2010) How small can thermal machines be? The smallest possible refrigerator. Phys Rev Lett 105:130401

18. Lloyd S (2014) Quantum optics: cool computation, hot bits. Nat Photon 8:90–91
19. Mor T, Roychowdhury V, Lloyd S, Fernandez JM, Weinstein Y (2005) Algorithmic cooling. US patent No. 6,873,154
20. Paz-Silva GA, Brennen GK, Twamley J (2010) Fault tolerance with noisy and slow measurements and preparation. Phys Rev Lett 105:100501
21. Rempp F, Michel M, Mahler G (2007) Cyclic cooling algorithm. Phys Rev A 76:032325
22. Renner R (2012) Thermodynamics: the fridge gate. Nature 482:164–165
23. Ryan CA, Moussa O, Baugh J, Laflamme R (2008) Spin based heat engine: demonstration of multiple rounds of algorithmic cooling. Phys Rev Lett 100:140501
24. Schulman LJ (2005) Quantum computing: a bit chilly. Nature 438:431–432
25. Schulman LJ, Mor T, Weinstein Y (2005) Physical limits of heat-bath algorithmic cooling. Phys Rev Lett 94:120501
26. Schulman LJ, Mor T, Weinstein Y (2007) Physical limits of heat-bath algorithmic cooling. SIAM J Comput (SICOMP) 36:1729–1747
27. Schulman LJ, Vazirani U (1999) Molecular scale heat engines and scalable quantum computation. In: Proceedings of the 31st ACM STOC (Symposium on Theory of Computing), Atlanta, pp 322–329
28. Simmons S, Brown RM, Riemann H, Abrosimov NV, Becker P, Pohl HJ, Thewalt MLW, Itoh KM, Morton JJL (2011) Entanglement in a solid-state spin ensemble. Nature 470:69–72
29. Sørensen OW (1989) Polarization transfer experiments in high-resolution NMR spectroscopy. Prog Nucl Magn Reson Spectrosc 21:503–569
30. Twamley J (2003) Quantum-cellular-automata quantum computing with endohedral fullerenes. Phys Rev A 67:052318
31. Xu JS, Yung MH, Xu XY, Boixo S, Zhou ZW, Li CF, Aspuru-Guzik A, Guo GC (2014) Demon-like algorithmic quantum cooling and its realization with quantum optics. Nat Photonics 8:113–118

Algorithmic Mechanism Design

Ron Lavi
Faculty of Industrial Engineering and Management, Technion, Haifa, Israel

Years and Authors of Summarized Original Work

1999; Nisan, Ronen

Problem Definition

Mechanism design is a sub-field of economics and game theory that studies the construction of social mechanisms in the presence of selfish agents. The nature of the agents dictates a basic contrast between the social planner, that aims to reach a socially desirable outcome, and the agents, that care only about their own private utility. The underlying question is how to incentivize the agents to cooperate, in order to reach the desirable social outcomes.

In the Internet era, where computers act and interact on behalf of selfish entities, the connection of the above to algorithmic design suggests itself: suppose that the input to an algorithm is kept by selfish agents, who aim to maximize their own utility. How can one design the algorithm so that the agents will find it in their best interest to cooperate, and a close-to-optimal outcome will be outputted? This is different than classic distributed computing models, where agents are either "good" (meaning obedient) or "bad" (meaning faulty, or malicious, depending on the context). Here, no such partition is possible. It is simply assumed that all agents are utility maximizers. To illustrate this, let us describe a motivating example:

A Motivating Example: Shortest Paths

Given a weighted graph, the goal is to find a shortest path (with respect to the edge weights) between a given source and target nodes. Each edge is controlled by a selfish entity, and the weight of the edge, w_e is private information of that edge. If an edge is chosen by the algorithm to be included in the shortest path, it will incur a cost which is minus its weight (the cost of communication). Payments to the edges are allowed, and the total utility of an edge that participates in the shortest path and gets a payment p_e is assumed to be $u_e = p_e - w_e$. Notice that the shortest path is *with respect to the true weights of the agents, although these are not known to the designer.*

Assuming that each edge will act in order to maximize its utility, how can one choose the

path and the payments? One option is to ignore the strategic issue all together, ask the edges to simply report their weights, and compute the shortest path. In this case, however, an edge dislikes being selected, and will therefore prefer to report a very high weight (much higher than its true weight) in order to decrease the chances of being selected. Another option is to pay each selected edge its reported weight, or its reported weight plus a small fixed "bonus". However in such a case all edges will report lower weights, as being selected will imply a positive gain.

Although this example is written in an algorithmic language, it is actually a mechanism design problem, and the solution, which is now a classic, was suggested in the 1970's. The chapter continues as follows: First, the abstract formulation for such problems is given, the classic solution from economics is described, and its advantages and disadvantages for algorithmic purposes are discussed. The next section then describes the new results that algorithmic mechanism design offers.

Abstract Formulation

The framework consists of a set A of alternatives, or outcomes, and n players, or agents. Each player i has a valuation function $v_i: A \to \Re$ that assigns a value to each possible alternative. This valuation function belongs to a domain V_i of all possible valuation functions. Let $V = V_1 \times \cdots \times V_n$, and $V_{-i} = \prod_{j \neq i} V_j$. Observe that this generalizes the shortest path example of above: A is all the possible $s-t$ paths in the given graph, $v_e(a)$ for some path $a \in A$ is either $-w_e$ (if $e \in a$) or zero.

A *social choice function* $f: V \to A$ assigns a socially desirable alternative to any given profile of players' valuations. This parallels the notion of an algorithm. A *mechanism* is a tuple $M = (f, p_1, \ldots, p_n)$, where f is a social choice function, and $p_i: V \to \Re$ (for $i = 1, \ldots, n$) is the price charged from player i. The interpretation is that the social planner asks the players to reveal their true valuations, chooses the alternative according to f as if the players have indeed acted truthfully, and in addition rewards/punishes the players with the prices. These prices should induce "truthfulness" in the following strong sense: no matter what the other players declare, it is always in the best interest of player i to reveal her true valuation, as this will maximize her utility. Formally, this translates to:

Definition 1 (Truthfulness) M is "truthful" (in dominant strategies) if, for any player i, any profile of valuations of the other players $v_{-i} \in V_{-i}$, and any two valuations of player i $v_i, v_i' \in V_i$,

$$v_i(a) - p_i(v_i, v_{-i}) \geq v_i(b) - p_i(v_i', v_{-i})$$

where $f(v_i, v_{-i}) = a$ and $f(v_i', v_{-i}) = b$.

Truthfulness is quite strong: a player need not know anything about the other players, even not that they are rational, and still determine the best strategy for her. Quite remarkably, there exists a truthful mechanism, even under the current level of abstraction. This mechanism suits all problem domains, where the social goal is to maximize the "social welfare":

Definition 2 (Social welfare maximization) A social choice function $f: V \to A$ maximizes the social welfare if $f(v) \in \text{argmax}_{a \in A} \sum_i v_i(a)$, for any $v \in V$.

Notice that the social goal in the shortest path domain is indeed welfare maximization, and, in general, this is a natural and important economic goal. Quite remarkably, there exists a general technique to construct truthful mechanisms that implement this goal:

Theorem 1 (Vickrey–Clarke–Groves (VCG)) *Fix any alternatives set A and any domain V, and suppose that $f: V \to A$ maximizes the social welfare. Then there exist prices p such that the mechanism (f, p) is truthful.*

This gives "for free" a solution to the shortest path problem, and to many other algorithmic problems. The great advantage of the VCG scheme is its generality: it suits *all* problem domains. The disadvantage, however, is that the method is tailored to social welfare maximization. This turns out to be restrictive,

especially for algorithmic and computational settings, due to several reasons: (i) different algorithmic goals: the algorithmic literature considers a variety of goals, including many that cannot be translated to welfare maximization. VCG does not help us in such cases. (ii) computational complexity: even if the goal is welfare maximization, in many settings achieving exactly the optimum is computationally hard. The CS discipline usually overcomes this by using approximation algorithms, but VCG will not work with such algorithm – reaching exact optimality is a necessary requirement of VCG. (iii) different algorithmic models: common CS models change "the basic setup", hence cause unexpected difficulties when one tries to use VCG (for example, an online model, where the input is revealed over time; this is common in CS, but changes the implicit setting that VCG requires). This is true even if welfare maximization is still the goal.

Answering any one of these difficulties requires the design of a non-VCG mechanism. What analysis tools should be used for this purpose? In economics and classic mechanism design, average-case analysis, that relies on the knowledge of the underlying distribution, is the standard. Computer science, on the other hand, usually prefers to avoid strong distributional assumptions, and to use worst-case analysis. This difference is another cause to the uniqueness of the answers provided by algorithmic mechanism design. Some of the new results that have emerged as a consequence of this integration between Computer Science and Economics is next described. Many other research topics that use the tools of algorithmic mechanism design are described in the entries on Adword Pricing, Competitive Auctions, False Name Proof Auctions, Generalized Vickrey Auction, Incentive Compatible Ranking, Mechanism for One Parameter Agents Single Buyer/Seller, Multiple Item Auctions, Position Auctions, and Truthful Multicast.

There are two different but closely related research topics that should be mentioned in the context of this entry. The first is the line of works that studies the "price of anarchy" of a given system. These works analyze *existing* systems, trying to quantify the loss of social efficiency due to the selfish nature of the participants, while the approach of algorithmic mechanism design is to understand how new systems should be designed. For more details on this topic the reader is referred to the entry on Price of Anarchy. The second topic regards the algorithmic study of various equilibria computation. These works bring computational aspects into economics and game theory, as they ask what equilibria notions are reasonable to assume, if one requires computational efficiency, while the works described here bring game theory and economics into computer science and algorithmic theory, as they ask what algorithms are reasonable to design, if one requires the resilience to selfish behavior. For more details on this topic the reader is referred (for example) to the entry on Algorithms for Nash Equilibrium and to the entry on General Equilibrium.

Key Results

Problem Domain 1: Job Scheduling

Job scheduling is a classic algorithmic setting: n jobs are to be assigned to m machines, where job j requires processing time p_{ij} on machine i. In the game-theoretic setting, it is assumed that each machine i is a selfish entity, that incurs a cost p_{ij} from processing job j. Note that the payments in this setting (and in general) may be negative, offsetting such costs. A popular algorithmic goal is to assign jobs to machines in order to minimize the "makespan": $\max_i \sum_{j \text{ is assigned to } i} p_{ij}$. This is different than welfare maximization, which translates in this setting to the minimization of $\sum_i \sum_{j \text{ is assigned to } i} p_{ij}$, further illustrating the problem of different algorithmic goals. Thus the VCG scheme cannot be used, and new methods must be developed.

Results for this problem domain depend on the specific assumptions about the structure of the processing time vectors. In the *related machines* case, $p_{ij} = p_j/s_i$ for any $i\,j$, where the p_j's are public knowledge, and the only secret parameter of player i is its *speed*, s_i.

Theorem 2 ([3, 22]) *For job scheduling on related machines, there exists a truthful exponential-time mechanism that obtains the optimal makespan, and a truthful polynomial-time mechanism that obtains a 3-approximation to the optimal makespan.*

More details on this result are given in the entry on Mechanism for One Parameter Agents Single Buyer. The bottom line conclusion is that, although the social goal is different than welfare maximization, there still exists a truthful mechanism for this goal. A non-trivial approximation guarantee is achieved, even under the additional requirement of computational efficiency. However, this guarantee does not match the best possible without the truthfulness requirement, since in this case a PTAS is known.

Open Question 1 *Is there a truthful PTAS for makespan minimization in related machines?*

If the number of machines is fixed then [2] give such a truthful PTAS.

The above picture completely changes in the move to the more general case of *unrelated machines*, where the p_{ij}'s are allowed to be arbitrary:

Theorem 3 ([13, 30]) *Any truthful scheduling mechanism for unrelated machines cannot approximate the optimal makespan by a factor better than* $1+\sqrt{2}$ *(for deterministic mechanisms) and* $2-1/m$ *(for randomized mechanisms).*

Note that this holds regardless of computational considerations. In this case, switching from welfare maximization to makespan minimization results in a strong impossibility. On the possibilities side, virtually nothing (!) is known. The VCG mechanism (which minimizes the total social cost) is an m-approximation of the optimal makespan [32], and, in fact, nothing better is currently known:

Open Question 2 *What is the best possible approximation for truthful makespan minimization in unrelated machines?*

What caused the switch from "mostly possibilities" to "mostly impossibilities"? Related machines is a single-dimensional domain (players hold only one secret number), for which truthfulness is characterized by a simple monotonicity condition, that leaves ample flexibility for algorithmic design. Unrelated machines, on the other hand, are a multi-dimensional domain, and the algorithmic conditions implied by truthfulness in such a case are harder to work with. It is still unclear whether these conditions imply real mathematical impossibilities, or perhaps just pose harder obstacles that can be in principle solved. One multi-dimensional scheduling domain for which possibility results are known is the case where $p_{ij} \in \{L_j, H_j\}$, where the "low" 's and "high" 's are fixed and known. This case generalizes the classic multi-dimensional model of restricted machines ($p_{ij} \in \{p_j, \infty\}$), and admits a truthful 3-approximation [27].

Problem Domain 2: Digital Goods and Revenue Maximization

In the E-commerce era, a new kind of "digital goods" have evolved: goods with no marginal production cost, or, in other words, goods with unlimited supply. One example is songs being sold on the Internet. There is a sunk cost of producing the song, but after that, additional electronic copies incur no additional cost. How should such items be sold? One possibility is to conduct an *auction*. An auction is a one-sided market, where a monopolistic entity (the auctioneer) wishes to sell one or more items to a set of buyers.

In this setting, each buyer has a privately known value for obtaining one copy of the good. Welfare maximization simply implies the allocation of one good to every buyer, but a more interesting question is the question of revenue maximization. How should the auctioneer design the auction in order to maximize his profit? Standard tools from the study of revenue-maximizing auctions (This model was not explicitly studied in classic auction theory, but standard results from there can be easily adjusted to this setting.) suggest to simply declare a price-per-buyer, determined by the probability distribution of the buyer's value, and make a take-it-or-leave-it offer.

However, such a mechanism needs to know the underlying distribution. Algorithmic mechanism design suggests an alternative, worst-case result, in the spirit of CS-type models and analysis.

Suppose that the auctioneer is required to sell all items in the same price, as is the case for many "real-life" monopolists, and denote by $F(\vec{v})$ the maximal revenue from a fixed-price sale to bidders with values $\vec{v} = v_1, \ldots v_n$, *assuming that all values are known*. Reordering indexes so that $v_1 \geq v_2 \geq \cdots \geq v_n$, let $F(\vec{v}) = \max_i i \cdot v_i$. The problem is, of-course, that in fact *nothing* about the values is known. Therefore, a truthful auction that extracts the players' values is in place. Can such an auction obtain a profit that is a constant fraction of $F(\vec{v})$, for any $\vec{v}$ (i.e., in the worst case)? Unfortunately, the answer is provably no [17]. The proof makes use of situations where the entire profit comes from the highest bidder. Since there is no potential for competition among bidders, a truthful auction cannot force this single bidder to reveal her value.

Luckily, a small relaxation in the optimality criteria significantly helps. Specifically, denote by $F^{(2)}(\vec{v}) = \max_{i \geq 2} i \cdot v_i$ (i.e., the benchmark is the auction that sells to at least two buyers).

Theorem 4 ([17, 20]) *There exists a truthful randomized auction that obtains an expected revenue of at least $F^{(2)}/3.25$, even in the worst-case. On the other hand, no truthful auction can approximate $F^{(2)}$ within a factor better than 2.42.*

Several interesting formats of distribution-free revenue-maximizing auctions have been considered in the literature. The common building block in all of them is the random partitioning of the set of buyers to random subsets, analyzing each set separately, and using the results on the other sets. Each auction utilizes a different analysis on the two subsets, which yields slightly different approximation guarantees. Aggarwal et al. [1] describe an elegant method to derandomize these type of auctions, while losing another factor of 4 in the approximation. More details on this problem domain can be found in the entry on Competitive Auctions.

Problem Domain 3: Combinatorial Auctions

Combinatorial auctions (CAs) are a central model with theoretical importance and practical relevance. It generalizes many theoretical algorithmic settings, like job scheduling and network routing, and is evident in many real-life situations. This new model has various pure computational aspects, and, additionally, exhibits interesting game theoretic challenges. While each aspect is important on its own, obviously only the integration of the two provides an acceptable solution.

A combinatorial auction is a multi-item auction in which players are interested in *bundles* of items. Such a valuation structure can represent substitutabilities among items, complementarities among items, or a combination of both. More formally, m items (Ω) are to be allocated to n players. Players value subsets of items, and $v_i(S)$ denotes i's value of a bundle $S \subseteq \Omega$. Valuations additionally satisfy: (i) monotonicity, i.e., $v_i(S) \leq v_i(T)$ for $S \subseteq T$, and (ii) normalization, i.e., $v_i(\emptyset) = 0$. The literature has mostly considered the goal of maximizing the social welfare: find an allocation $(S_1, \ldots, S_n)$ that maximizes $\sum_i v_i(S_i)$.

Since a general valuation has size exponential in n and m, the representation issue must be taken into account. Two models are usually considered (see [11] for more details). In the *bidding languages* model, the bid of a player represents his valuation is a concise way. For this model it is NP-hard to approximate the social welfare within a ratio of $\Omega(m^{1/2-\epsilon})$, for any $\epsilon > 0$ (if "single-minded" bids are allowed; the exact definition is given below). In the *query access* model, the mechanism iteratively queries the players in the course of computation. For this model, any algorithm with polynomial communication cannot obtain an approximation ratio of $\Omega(m^{1/2-\epsilon})$ for any $\epsilon > 0$. These bounds are tight, as there exist a deterministic $\sqrt{m}$-approximation with polynomial computation and communication. Thus, for the general valuation structure, the computational status by itself is well-understood.

The basic incentives issue is again well-understood: VCG obtains truthfulness. Since

VCG requires the exact optimum, which is NP-hard to compute, the two considerations therefore clash, when attempting to use classic techniques. Algorithmic mechanism design aims to develop new techniques, to integrate these two desirable aspects.

The first positive result for this integration challenge was given by [29], for the special case of "single-minded bidders": each bidder, i, is interested in a specific bundle S_i, for a value v_i (any bundle that contains S_i is worth v_i, and other bundles have zero value). Both v_i, S_i are private to the player i.

Theorem 5 ([29]) *There exists a truthful and polynomial-time deterministic combinatorial auction for single-minded bidders, which obtains a $\sqrt{m}$-approximation to the optimal social welfare.*

A possible generalization of the basic model is to assume that each item has B copies, and each player still desires at most one copy from each item. This is termed "multi-unit CA". As B grows, the integrality constraint of the problem reduces, and so one could hope for better solutions. Indeed, the next result exploits this idea:

Theorem 6 ([7]) *There exists a truthful and polynomial-time deterministic multi-unit CA, for $B \geq 3$ copies of each item, that obtains $O(B \cdot m^{1/(B-2)})$-approximation to the optimal social welfare.*

This auction copes with the representation issue (since general valuations are assumed) by accessing the valuations through a "demand oracle": given per-item prices $\{p_x\}_{x \in \Omega}$, specify a bundle S that maximizes $v_i(S) - \sum_{x \in S} p_x$.

Two main drawbacks of this auction motivate further research on the issue. First, as B gets larger it is reasonable to expect the approximation to approach 1 (indeed polynomial-time algorithms with such an approximation guarantee do exist). However here the approximation ratio does not decrease below $O(\log m)$ (this ratio is achieved for $B = O(\log m)$). Second, this auction does not provide a solution to the original setting, where $B = 1$, and, in general for small B's the approximation factor is rather high. One way to cope with these problems is to introduce randomness:

Theorem 7 ([26]) *There exists a truthful-in-expectation and polynomial-time randomized multi-unit CA, for any $B \geq 1$ copies of each item, that obtains $O(m^{1/(B+1)})$-approximation to the optimal social welfare.*

Thus, by allowing randomness, the gap from the standard computational status is being completely closed. The definition of truthfulness-in-expectation is the natural extension of truthfulness to a randomized environment: the *expected* utility of a player is maximized by being truthful.

However, this notion is strictly weaker than the deterministic notion, as this implicitly implies that players care only about the expectation of their utility (and not, for example, about the variance). This is termed "the risk-neutrality" assumption in the economics literature. An intermediate notion for randomized mechanisms is that of "universal truthfulness": the mechanism is truthful given any fixed result of the coin toss. Here, risk-neutrality is no longer needed. Dobzinski et al. [15] give a universally truthful CA for $B = 1$ that obtains an $O(\sqrt{m})$-approximation. Universally truthful mechanisms are still weaker than deterministic truthful mechanisms, due to two reasons: (i) It is not clear how to actually create the correct and exact probability distribution with a deterministic computer. The situation here is different than in "regular" algorithmic settings, where various derandomization techniques can be employed, since these in general does not carry through the truthfulness property. (ii) Even if a natural randomness source exists, one cannot improve the quality of the actual output by repeating the computation several times (using the the law of large numbers). Such a repetition will again destroy truthfulness. Thus, exactly because the game-theoretic issues are being considered in parallel to the computational ones, the importance of determinism increases.

Open Question 3 *What is the best-possible approximation ratio that deterministic and*

truthful combinatorial auctions can obtain, in polynomial-time?

There are many valuation classes, that restrict the possible valuations to some reasonable format (see [28] for more details). For example, sub-additive valuations are such that, for any two bundles $S, T, \subseteq \Omega$, $v(S \cup T) \leq v(S) + v(T)$. Such classes exhibit much better approximation guarantees, e.g., for sub-additive valuation a polynomial-time 2-approximation is known [16]. However, no polynomial-time truthful mechanism (be it randomized, or deterministic) with a constant approximation ratio, is known for any of these classes.

Open Question 4 *Does there exist polynomial-time truthful constant-factor approximations for special cases of CAs that are NP-hard?*

Revenue maximization in CAs is of-course another important goal. This topic is still mostly unexplored, with few exceptions. The mechanism [7] obtains the same guarantees with respect to the optimal revenue. Improved approximations exist for multi-unit auctions (where all items are identical) with budget constrained players [12], and for unlimited-supply CAs with single-minded bidders [6].

The topic of Combinatorial Auctions is discussed also in the entry on Multiple Item Auctions.

Problem Domain 4: Online Auctions

In the classic CS setting of "online computation", the input to an algorithm is not revealed all at once, before the computation begins, but gradually, over time (for a detailed discussion see the many entries on online problems in this book). This structure suits the auction world, especially in the new electronic environments. What happens when players arrive over time, and the auctioneer must make decisions facing only a subset of the players at any given time?

The integration of online settings, worst-case analysis, and auction theory, was suggested by [24]. They considered the case where players arrive one at a time, and the auctioneer must provide an answer to each player *as it arrives*, without knowing the future bids. There are k identical items, and each bidder may have a distinct value for every possible quantity of the item. These values are assumed to be marginally decreasing, where each marginal value lies in the interval $[\underline{v}, \bar{v}]$. The private information of a bidder includes both her valuation function, and her arrival time, and so a truthful auction need to incentivize the players to arrive on time (and not later on), and to reveal their true values. The most interesting result in this setting is for a large k, so that in fact there is a continuum of items:

Theorem 8 ([24]) *There exists a truthful online auction that simultaneously approximates, within a factor of* $O(\log(\bar{v}/\underline{v}))$*, the optimal offline welfare, and the offline revenue of VCG. Furthermore, no truthful online auction can obtain a better approximation ratio to either one of these criteria (separately).*

This auction has the interesting property of being a "posted price" auction. Each bidder is not required to reveal his valuation function, but, rather, he is given a price for each possible quantity, and then simply reports the desired quantity under these prices.

Ideas from this construction were later used by [10] to construct *two*-sided online auction markets, where multiple sellers and buyers arrive online.

This approximation ratio can be dramatically improved, to be a constant, 4, if one assumes that (i) there is only one item, and (ii) player values are i.i.d from some fixed distribution. No a–priori knowledge of this distribution is needed, as neither the mechanism nor the players are required to make any use of it. This work, [19], analyzes this by making an interesting connection to the class of "secretary problems".

A general method to convert online algorithms to online mechanisms is given by [4]. This is done for one item auctions, and, more generally, for one parameter domains. This method is competitive both with respect to the welfare and the revenue.

The revenue that the online auction of Theorem 8 manages to raise is competitive only with respect to VCG's revenue, which may be far from optimal. A parallel line of works is concerned with revenue maximizing auctions. To achieve good results, two assumptions need to be made: (i) there exists an unlimited supply of items (and recall from section "Problem Domain 2: Digital Goods and Revenue Maximization" that $F(v)$ is the offline optimal monopolistic fixed-price revenue), and (ii) players cannot lie about their arrival time, only about their value. This last assumption is very strong, but apparently needed. Such auctions are termed here "value-truthful", indicating that "time-truthfulness" is missing.

Theorem 9 ([9]) *For any $\epsilon > 0$, there exists a value-truthful online auction, for the unlimited supply case, with expected revenue of at least $(F(v))/(1+\epsilon) - O(h/\epsilon^2)$.*

The construction exploits principles from learning theory in an elegant way. Posted price auctions for this case are also possible, in which case the additive loss increases to $O(h \log\log h)$. Hajiaghayi et al. [19] consider fully-truthful online auctions for revenue maximization, but manage to obtain only very high (although fixed) competitive ratios. Constructing fully-truthful online auctions with a close-to-optimal revenue remains an open question. Another interesting open question involves multi-dimensional valuations. The work [24] remains the only work for players that may demand multiple items. However their competitive guarantees are quite high, and achieving better approximation guarantees (especially with respect to the revenue) is a challenging task.

Advanced Issues

Monotonicity

What is the general way for designing a truthful mechanism? The straight-forward way is to check, for a given social choice function f, whether truthful prices exist. If not, try to "fix" f. It turns out, however, that there exists a more structured way, an *algorithmic* condition that will imply the *existence* of truthful prices. Such a condition shifts the designer back to the familiar territory of algorithmic design. Luckily, such a condition do exist, and is best described in the abstract social choice setting of section "Problem Definition":

Definition 3 ([8, 23]) A social choice function $f : V \to A$ is "weakly monotone" (W-MON) if for any i, $v_{-i} \in V_{-i}$, and any $v_i, v_i' \in V_i$, the following holds. Suppose that $f(v_i, v_{-i}) = a$, and $f(v_i', v_{-i}) = b$. Then $v_i'(b) - v_i(b) \geq v_i'(a) - v_i(a)$.

In words, this condition states the following. Suppose that player i changes her declaration from v_i to v_i', and this causes the social choice to change from a to b. Then it must be the case that i's value for b has increased in the transition from v_i to v_i' no-less than i's value for a.

Theorem 10 ([35]) *Fix a social choice function $f : V \to A$, where V is convex, and A is finite. Then there exist prices p such that $M = (f, p)$ is truthful if and only if f is weakly monotone.*

Furthermore, given a weakly monotone f, there exists an explicit way to determine the appropriate prices p (see [18] for details).

Thus, the designer should aim for weakly monotone algorithms, and need not worry about actual prices. But how difficult is this? For single-dimensional domains, it turns out that W-MON leaves ample flexibility for the algorithm designer. Consider for example the case where every alternative has a value of either 0 (the player "loses") or some $v_i \in \Re$ (the player "wins" and obtains a value v_i). In such a case, it is not hard to show that W-MON reduces to the following monotonicity condition: if a player wins with v_i, and increases her value to $v_i' > v_i$ (while v_{-i} remains fixed), then she must win with v_i' as well. Furthermore, in such a case, the price of a winning player must be set to the infimum over all winning values.

Impossibilities of truthful design

It is fairly simple to construct algorithms that satisfy W-MON for single-dimensional domains, and a variety of positive results were obtained for such domains, in classic mechanism design, as well as in algorithmic mechanism design. But how hard is it to satisfy W-MON for multi-dimensional domains? This question is yet unclear, and seems to be one of the challenges of algorithmic mechanism design. The contrast between single-dimensionality and multi-dimensionality appears in all problem domains that were surveyed here, and seems to reflect some inherent difficulty that is not exactly understood yet. Given a social choice function f, call f *implementable* (in dominant strategies) if there exist prices p such that $M = (f, p)$ is truthful. The basic question is then *what forms of social choice functions are implementable.*

As detailed in the beginning, the welfare maximizing social choice function is implementable. This specific function can be slightly generalized to allow weights, in the following way: fix some non-negative real constants $\{w_i\}_{i=1}^n$ (not all are zero) and $\{\gamma_a\}_{a\in A}$, and choose an alternative that maximizes the *weighted* social welfare, i.e., $f(v) \in \text{argmax}_{a\in A} \sum_i w_i v_i(a) + \gamma_a$. This class of functions is sometimes termed "affine maximizers". It turns out that these functions are also implementable, with prices similar in spirit to VCG. In the context of the above characterization question, one sharp result stands out:

Theorem 11 ([34]) *Fix a social choice function $f : V \to A$, such that (i) A is finite, $|A| \geq 3$, and f is onto A, and (ii) $V_i = \Re^A$ for all i. Then f is implementable (in dominant strategies) if and only if it is an affine maximizer.*

The domain V that satisfies $V_i = \Re^A$ for all i is term an "unrestricted domain". The theorem states that, if the domain is unrestricted, at least three alternatives are chosen, and the set A of alternatives is finite, then nothing besides affine maximizers can be implemented!

However, the assumption that the domain is unrestricted is very restrictive. All the above example domains exhibit some basic combinatorial structure, and are therefore restricted in some way. And as discussed above, for many restricted domains the theorem is simply not true. So what *is* the possibilities – impossibilities border? As mentioned above, this is an unsolved challenge. Lavi, Mu'alem, and Nisan [23] explore this question for Combinatorial Auctions and similar restricted domains, and reach partial answers. For example:

Theorem 12 ([23]) *Any truthful combinatorial auction or multi-unit auction among two players, that must always allocate all items, and that approximates the welfare by a factor better than 2, must be an affine maximizer.*

Of-course, this is far from being a complete answer. What happens if there are more than two players? And what happens if it is possible to "throw away" part of the items? These questions, and the more general and abstract characterization question, are all still open.

Alternative solution concepts

In light of the conclusions of the previous section, a natural thought would be to re-examine the *solution concept* that is being used. Truthfulness relies on the strong concept of dominant strategies: for each player there is a unique strategy that maximizes her utility, no matter what the other players are doing. This is very strong, but it fits very well the worst-case way of thinking in CS. What other solution concepts can be used? As described above, randomization, and truthfulness-in-expectation, can help. A related concept, again for randomized mechanisms, is truthfulness with high probability. Another direction is to consider mechanisms where players cannot improve their utility *too much* by deviating from the truth-telling strategy [21].

Algorithm designers do not care so much about actually reaching an equilibrium point, or finding out what will the players play – the major concern is to guarantee the optimality of the solution, taking into account the strategic behavior of the players. Indeed, one way of doing this is to guarantee a good equilibrium point. But there is

no reason to rule out mechanisms where several acceptable strategic choices for the players exist, provided that the approximation will be achieved *in each of these choices*.

As a first attempt, one is tempted to simply let the players try and improve the basic result by allowing them to lie. However, this can cause unexpected dynamics, as each player chooses her lies under some assumptions about the lies of the others, etc. etc. To avoid such an unpredictable situation, it is important to insist on using rigorous game theoretic reasoning to explain exactly why the outcome will be satisfactory.

The work [31] suggests the notion of "feasibly dominant" strategies, where players reveal the possible lies they consider, and the mechanism takes this into account. By assuming that the players are computationally bounded, one can show that, instead of actually "lying", the players will prefer to reveal their true types plus all the lies they might consider. In such a case, since the mechanism has obtained the true types of the players, a close-to-optimal outcome will be guaranteed.

Another definition tries to capture the initial intuition by using the classic game-theoretic notion of undominated strategies:

Definition 4 ([5]) A mechanism M is an "algorithmic implementation of a c-approximation (in undominated strategies)" if there exists a set of strategies, D, such that (i) M obtains a c-approximation for any combination of strategies from D, in polynomial time, and (ii) For any strategy not in D, there exists a strategy in D that weakly dominates it, and this transition is polynomial-time computable.

By the second condition, it is reasonable to assume that a player will indeed play *some* strategy in D, and, by the first condition, it does not matter what tuple of strategies in D will actually be chosen, as any of these will provide the approximation. This transfers some of the burden from the game-theoretic design to the algorithmic design, since now a guarantee on the approximation should bu provided for a larger range of strategies. Babaioff et al. [5] exploit this notion to design a deterministic CA for multi-dimensional players that achieves a close-to-optimal approximation guarantee. A similar-in-spirit notion, although a weaker one, is the notion of "Set-Nash" [25].

Applications

One of the popular examples to a "real-life" combinatorial auction is the spectrum auction that the US government conducts, in order to sell spectrum licenses. Typical bids reflect values for different spectrum ranges, to accommodate different geographical and physical needs, where different spectrum ranges may complement or substitute one another. The US government invests research efforts in order to determine the best format for such an auction, and auction theory is heavily exploited. Interestingly, the US law guides the authorities to allocate these spectrum ranges in a way that will maximize *the social welfare*, thus providing a good example for the usefulness of this goal.

Adword auctions are another new and fast-growing application of auction theory in general, and of the new algorithmic auctions in particular. These are auctions that determine the advertisements that web-search engines place close to the search results they show, after the user submits her search keywords. The interested companies compete, for every given keyword, on the right to place their ad on the results' page, and this turns out to be the main source of income for companies like Google. Several entries in this book touch on this topic in more details, including the entries on Adwords Pricing and on Position Auctions.

A third example to a possible application, in the meanwhile implemented only in the academic research labs, is the application of algorithmic mechanism design to pricing and congestion control in communication networks. The existing fixed pricing scheme has many disadvantages, both with respect to the needs of efficiently allocating the available resources, and with respect to the new opportunities of the Internet companies to raise more revenue due to specific types of

traffic. Theory suggests solutions to both of these problems.

Cross-References

- Adwords Pricing
- Competitive Auction
- False-Name-Proof Auction
- Generalized Vickrey Auction
- Incentive Compatible Selection
- Position Auction
- Truthful Multicast

Recommended Reading

The topics presented here are detailed in the textbook [33]. Section "Problem Definition" is based on the paper [32], that also coined the term "algorithmic mechanism design". The book [14] covers the various aspects of combinatorial auctions.

1. Aggarwal G, Fiat A, Goldberg A, Immorlica N, Sudan M (2005) Derandomization of auctions. In: Proceedings of the 37th ACM symposium on theory of computing (STOC'05)
2. Andelman N, Azar Y, Sorani M (2005) Truthful approximation mechanisms for scheduling selfish related machines. In: Proceedings of the 22nd international symposium on theoretical aspects of computer science (STACS), pp 69–82
3. Archer A, Tardos É (2001) Truthful mechanisms for one-parameter agents. In: Proceedings of the 42nd annual symposium on foundations of computer science (FOCS), pp 482–491
4. Awerbuch B, Azar Y, Meyerson A (2003) Reducing truth-telling online mechanisms to online optimization. In: Proceedings of the 35th ACM symposium on theory of computing (STOC'03)
5. Babaioff M, Lavi R, Pavlov E (2006) Single-value combinatorial auctions and implementation in undominated strategies. In: Proceedings of the 17th symposium on discrete algorithms (SODA)
6. Balcan M, Blum A, Hartline J, Mansour Y (2005) Mechanism design via machine learning. In: Proceedings of the 46th annual symposium on foundations of computer science (FOCS'05)
7. Bartal Y, Gonen R, Nisan N (2003) Incentive compatible multiunit combinatorial auctions. In: Proceedings of the 9th conference on theoretical aspects of rationality and knowledge (TARK'03)
8. Bikhchandani S, Chatterjee S, Lavi R, Mu'alem A, Nisan N, Sen A (2006) Weak monotonicity characterizes deterministic dominant-strategy implementation. Econometrica 74:1109–1132
9. Blum A, Hartline J (2005) Near-optimal online auctions. In: Proceedings of the 16th symposium on discrete algorithms (SODA)
10. Blum A, Sandholm T, Zinkevich M (2006) Online algorithms for market clearing. J ACM 53(5):845–879
11. Blumrosen L, Nisan N (2005) On the computational power of iterative auctions. In: Proceedings of the 7th ACM conference on electronic commerce (EC'05)
12. Borgs C, Chayes J, Immorlica N, Mahdian M, Saberi A (2005) Multi-unit auctions with budget-constrained bidders. In: Proceedings of the 7th ACM conference on electronic commerce (EC'05)
13. Christodoulou G, Koutsoupias E, Vidali A (2007) A lower bound for scheduling mechanisms. In: Proceedings of the 18th symposium on discrete algorithms (SODA)
14. Cramton P, Shoham Y, Steinberg R (2005) Combinatorial auctions. MIT
15. Dobzinski S, Nisan N, Schapira M (2006) Truthful randomized mechanisms for combinatorial auctions. In: Proceedings of the 38th ACM symposium on theory of computing (STOC'06)
16. Feige U (2006) On maximizing welfare when utility functions are subadditive. In: Proceedings of the 38th ACM symposium on theory of computing (STOC'06)
17. Goldberg A, Hartline J, Karlin A, Saks M, Wright A (2006) Competitive auctions. Games Econ Behav 55(2):242–269
18. Gui H, Muller R, Vohra RV (2004) Characterizing dominant strategy mechanisms with multidimensional types. Working paper
19. Hajiaghayi M, Kleinberg R, Parkes D (2004) Adaptive limited-supply online auctions. In: Proceedings of the 6th ACM conference on electronic commerce (EC'04)
20. Hartline J, McGrew R (2005) From optimal limited to unlimited supply auctions. In: Proceedings of the 7th ACM conference on electronic commerce (EC'05)
21. Kothari A, Parkes D, Suri S (2005) Approximately-strategy proof and tractable multi-unit auctions. Decis Support Syst 39:105–121
22. Kovács A (2005) Fast monotone 3-approximation algorithm for scheduling related machines. In: Proceedings of the 13th annual European symposium on algorithms (ESA), pp 616–627
23. Lavi R, Mu'alem A, Nisan N (2003) Towards a characterization of truthful combinatorial auctions. In: Proceedings of the 44rd annual symposium on foundations of computer science (FOCS'03)
24. Lavi R, Nisan N (2004) Competitive analysis of incentive compatible on-line auctions. Theor Comput Sci 310:159–180
25. Lavi R, Nisan N (2005) Online ascending auctions for gradually expiring items. In: Proceedings of the 16th symposium on discrete algorithms (SODA)

26. Lavi R, Swamy C (2005) Truthful and near-optimal mechanism design via linear programming. In: Proceedings of the 46th annual symposium on foundations of computer science (FOCS), pp 595–604
27. Lavi R, Swamy C (2007) Truthful mechanism design for multi-dimensional scheduling via cycle monotonicity. Working paper
28. Lehmann B, Lehmann D, Nisan N (2006) Combinatorial auctions with decreasing marginal utilities. Games Econ Behav 55(2):270–296
29. Lehmann D, O'Callaghan L, Shoham Y (2002) Truth revelation in approximately efficient combinatorial auctions. J ACM 49(5):577–602
30. Mu'alem A, Schapira M (2007) Setting lower bounds on truthfulness. In: Proceedings of the 18th symposium on discrete algorithms (SODA)
31. Nisan N, Ronen A (2000) Computationally feasible VCG mechanisms. In: Proceedings of the 2nd ACM conference on electronic commerce (EC'00)
32. Nisan N, Ronen A (2001) Algorithmic mechanism design. Games Econ Behav 35:166–196
33. Nisan N, Roughgarden T, Tardos E, Vazirani V (2007) Algorithmic game theory. Cambridge University Press (expected to appear)
34. Roberts K (1979) The characterization of implementable choice rules. In: Laffont JJ (ed) Aggregation and revelation of preferences. North-Holland, pp 321–349
35. Saks M, Yu L (2005) Weak monotonicity suffices for truthfulness on convex domains. In: Proceedings of the 6th ACM conference on electronic commerce (ACM-EC), pp 286–293

Algorithms for Combining Rooted Triplets into a Galled Phylogenetic Network

Jesper Jansson[1] and Wing-Kin Sung[2]
[1]Laboratory of Mathematical Bioinformatics, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto, Japan
[2]Department of Computer Science, National University of Singapore, Singapore, Singapore

Keywords

Dense set; Galled phylogenetic network; Phylogenetic tree; Polynomial-time approximation algorithm; Rooted triplet

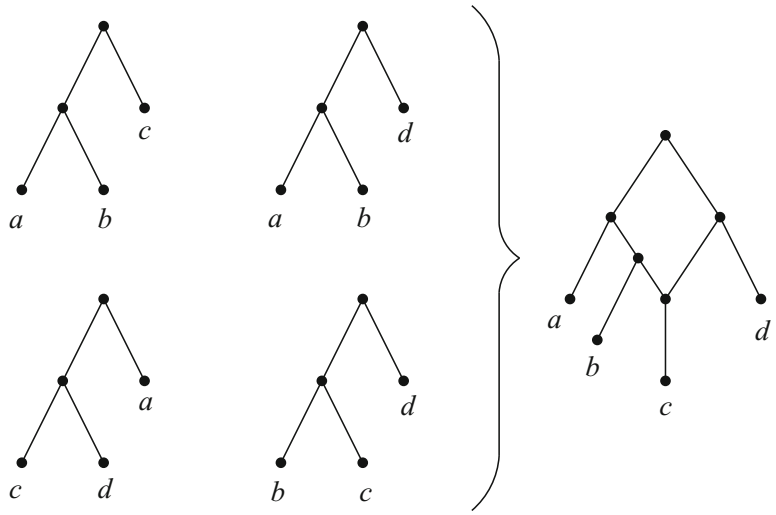
Years and Authors of Summarized Original Work

2006; Jansson, Sung
2006; Jansson, Nguyen, Sung
2006; He, Huynh, Jansson, Sung
2010; Byrka, Gawrychowski, Huber, Kelk
2011; van Iersel, Kelk

Problem Definition

A *phylogenetic tree* is a binary, rooted, unordered tree whose leaves are distinctly labeled. A *phylogenetic network* is a generalization of a phylogenetic tree formally defined as a rooted, connected, directed acyclic graph in which (1) each node has outdegree at most 2; (2) each node has indegree 1 or 2, except the root node which has indegree 0; (3) no node has both indegree 1 and outdegree 1; and (4) all nodes with outdegree 0 are labeled by elements from a finite set L in such a way that no two nodes are assigned the same label. Nodes of outdegree 0 are referred to as *leaves* and are identified with their corresponding elements in L. Nodes with indegree 2 are called *reticulation nodes*. For any phylogenetic network N, let $\mathcal{U}(N)$ be the undirected graph obtained from N by replacing each directed edge by an undirected edge. N is said to be a *galled phylogenetic network* (*galled network*, for short) if all cycles in $\mathcal{U}(N)$ are node-disjoint. Galled networks are also known in the literature as *topologies with independent recombination events* [15], *galled-trees* [6], and *level*-1 *phylogenetic networks* [2, 5, 7, 9, 10, 14].

A phylogenetic tree with exactly three leaves is called a *rooted triplet*. The unique rooted triplet on a leaf set $\{x, y, z\}$ in which the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of x and z (or equivalently, where the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of y and z) is denoted by $xy|z$. For any phylogenetic network N, the rooted triplet $xy|z$ is said to be *consistent* with N if N contains three leaves labeled by x, y, and z



Algorithms for Combining Rooted Triplets into a Galled Phylogenetic Network, Fig. 1 A dense set $\mathcal{T} = \{ab|c, ab|d, cd|a, bc|d\}$ of rooted triplets with leaf set $\{a, b, c, d\}$ and a galled phylogenetic network that is consistent with $\mathcal{T}$. Note that this solution is not unique

as well as two internal vertices w and z such that there are four directed paths of nonzero length from w to a, from w to b, from z to w, and from z to c that are vertex-disjoint except for in the vertices w and z. A set $\mathcal{T}$ of rooted triplets is *consistent* with N if every rooted triplet in $\mathcal{T}$ is consistent with N. See Fig. 1 for an example.

Denote the set of leaves in any phylogenetic network N by $\Lambda(N)$, and for any set $\mathcal{T}$ of rooted triplets, define $\Lambda(\mathcal{T}) = \bigcup_{t_i \in \mathcal{T}} \Lambda(t_i)$. A set $\mathcal{T}$ of rooted triplets is *dense* if for each $\{x, y, z\} \subseteq \Lambda(\mathcal{T})$, at least one of the three possible rooted triplets $xy|z$, $xz|y$, and $yz|x$ belongs to $\mathcal{T}$. Observe that if $\mathcal{T}$ is dense, then $|\mathcal{T}| = \Theta(|\Lambda(\mathcal{T})|^3)$. Jansson and Sung introduced the following problem in [10].

Problem 1 Given a set $\mathcal{T}$ of rooted triplets, output a galled network N with $\Lambda(N) = \Lambda(\mathcal{T})$ such that N and $\mathcal{T}$ are consistent, if such a network exists; otherwise, output *null*.

A natural optimization version of Problem 1 is:

Problem 2 Given a set $\mathcal{T}$ of rooted triplets, output a galled network N with $\Lambda(N) = \Lambda(\mathcal{T})$ that is consistent with the maximum possible number of rooted triplets belonging to $\mathcal{T}$.

A generalization of Problem 1 studied by He et al. in [8] involves *forbidden* rooted triplets and is defined as follows.

Problem 3 Given two sets $\mathcal{T}$ and $\mathcal{F}$ of rooted triplets, output a galled network N with $\Lambda(N) = \Lambda(\mathcal{T}) \cup \Lambda(\mathcal{F})$ such that (1) N and $\mathcal{T}$ are consistent and (2) N is not consistent with any rooted triplet belonging to $\mathcal{F}$; if no such network exists, output *null*.

Below, we write $L = \Lambda(\mathcal{T})$ and $n = |L|$.

Key Results

As shown in [11], Problem 1 can be solved in (optimal) $O(|\mathcal{T}|) = O(n^3)$ time for dense inputs:

Theorem 1 ([11]) *Given any dense set $\mathcal{T}$ of rooted triplets with leaf set L, a galled network consistent with $\mathcal{T}$ (if one exists) can be constructed in $O(n^3)$ time, where $n = |L|$.*

The algorithm referred to in Theorem 1 was extended by van Iersel and Kelk [14] as follows.

Theorem 2 ([14]) *Given any dense set $\mathcal{T}$ of rooted triplets with leaf set L, a galled network consistent with $\mathcal{T}$ (if one exists) that contains as few reticulation nodes as possible can be constructed in $O(n^5)$ time, where $n = |L|$.*

For the more general case of nondense inputs, Problem 1 becomes harder:

Theorem 3 ([11]) *The problem of determining if there exists a galled network that is consistent*

with an input nondense set $\mathcal{T}$ of rooted triplets is NP-hard.

Since not all sets of rooted triplets are consistent with a galled network, it is of interest to consider Problem 2. It follows from Theorem 3 that Problem 2 is also NP-hard for nondense inputs, and this motivates polynomial-time approximation algorithms. Say that an algorithm for Problem 2 is an *f-approximation algorithm* if it always returns a galled network N such that $\frac{N(\mathcal{T})}{|\mathcal{T}|} \geq f$, where $N(\mathcal{T})$ is the number of rooted triplets in $\mathcal{T}$ that are consistent with N. Define the nonlinear recurrence relation $S(n) = \max_{1 \leq a \leq n} \left\{ \binom{a}{3} + 2 \cdot \binom{a}{2} \cdot (n-a) + a \cdot \binom{n-a}{2} + S(n-a) \right\}$ for $n > 0$ and $S(0) = 0$. It was shown in [4] that $\lim_{n \to \infty} \frac{S(n)}{3\binom{n}{3}} = \frac{2(\sqrt{3}-1)}{3} \approx 0.488033\ldots$ and that $\frac{S(n)}{3\binom{n}{3}} > \frac{2(\sqrt{3}-1)}{3} \approx 0.488033\ldots$ for all $n > 2$. The following theorem was proved by Byrka et al. in [2].

Theorem 4 ([2]) *There exists an $\frac{S(n)}{3\binom{n}{3}}$-approximation algorithm for Problem 2 that runs in $O(n^3 + n|\mathcal{T}|)$ time.*

A matching negative bound is:

Theorem 5 ([11]) *For any $f > \lim_{n \to \infty} \frac{S(n)}{3\binom{n}{3}}$, there exists a set $\mathcal{T}$ of rooted triplets such that no galled network can be consistent with at least a factor of f of the rooted triplets in $\mathcal{T}$. (Thus, no f-approximation algorithm for Problem 2 is possible.)*

For Problem 3, Theorem 3 immediately implies NP-hardness by taking $\mathcal{F} = \emptyset$. The following positive result is known for the optimization version of Problem 3.

Theorem 6 ([8]) *There exists an $O(|L|^2|\mathcal{T}|(|\mathcal{T}| + |\mathcal{F}|))$-time algorithm for inferring a galled network N that guarantees $|N(\mathcal{T})| - |N(\mathcal{F})| \geq \frac{5}{12} \cdot (|\mathcal{T}| - |\mathcal{F}|)$, where $L = \Lambda(\mathcal{T}) \cup \Lambda(\mathcal{F})$.*

Finally, we remark that the analogous version of Problem 1 of inferring a phylogenetic *tree* consistent with all the rooted triplets in an input set (when such a tree exists) can be solved in polynomial time with a classical algorithm by Aho et al. [1] from 1981. Similarly, for Problem 2, to infer a phylogenetic tree consistent with as many rooted triplets from an input set of rooted triplets as possible is NP-hard and admits a polynomial-time 1/3-approximation algorithm, which is optimal in the sense that there exist certain inputs for which no tree can achieve a factor larger than 1/3. See, e.g., [3] for a survey of known results about maximizing rooted triplet consistency for trees. On the other hand, more complex network structures such as the *level-k phylogenetic networks* [5] permit a higher percentage of the input rooted triplets to be embedded; in the extreme case, if there are no restrictions on the reticulation nodes at all, then a sorting network-based construction yields a phylogenetic network that is trivially consistent with every rooted triplet over L [10]. A number of efficient algorithms for combining rooted triplets into higher level networks have been developed; see, e.g., [2,7,14] for further details and references.

Applications

Phylogenetic networks are used by scientists to describe evolutionary relationships that do not fit the traditional models in which evolution is assumed to be treelike. Evolutionary events such as horizontal gene transfer or hybrid speciation (often referred to as *recombination events*) which suggest convergence between objects cannot be represented in a single tree but can be modeled in a phylogenetic network as internal nodes having more than one parent (i.e., reticulation nodes). The phylogenetic network is a relatively new tool, and various fast and reliable methods for constructing and comparing phylogenetic networks are currently being developed.

Galled networks form an important class of phylogenetic networks. They have attracted special attention in the literature [5, 6, 15] due to their biological significance (see [6]) and their simple, almost treelike, structure. When the number of recombination events is limited and most of the recombination events have occurred recently, a galled network may suffice to accurately describe the evolutionary process under study [6]. The motivation behind the

rooted triplet approach taken here is that a highly accurate tree for each cardinality-three subset of the leaf set can be obtained through maximum likelihood-based methods or Sibley-Ahlquist-style DNA-DNA hybridization experiments (see [13]). The algorithms mentioned above can be used as the merging step in a divide-and-conquer approach for constructing phylogenetic networks analogous to the quartet method paradigm for inferring unrooted phylogenetic trees [12] and other supertree methods. We consider dense input sets in particular because this case can be solved in polynomial time.

Open Problems

The approximation factor given in Theorem 4 is expressed in terms of the number of rooted triplets in the input $\mathcal{T}$, and Theorem 5 shows that it cannot be improved. However, if one measures the quality of the approximation in terms of a galled network N_{OPT} that is consistent with the maximum possible number of rooted triplets from $\mathcal{T}$, Theorem 4 can be far from optimal. An open problem is to determine the polynomial-time approximability and inapproximability of Problem 2 when the approximation ratio is defined as $\frac{N(\mathcal{T})}{N_{OPT}(\mathcal{T})}$ instead of $\frac{N(\mathcal{T})}{|\mathcal{T}|}$.

Another research direction is to develop fixed-parameter polynomial-time algorithms for Problem 1. The level of the constructed network, the number of allowed reticulation nodes, or some measure of the density of the input set of rooted triplet might be suitable parameters.

URLs to Code and Data Sets

A Java implementation of the algorithm for Problem 1 referred to in Theorem 2 (coded by its authors [14]) is available at http://skelk.sdf-eu.org/marlon.html. See also http://skelk.sdf-eu.org/lev1athan/ for a Java implementation of a polynomial-time heuristic described in [9] for Problem 2.

Cross-References

- Directed Perfect Phylogeny (Binary Characters)
- Distance-Based Phylogeny Reconstruction (Fast-Converging)
- Distance-Based Phylogeny Reconstruction: Safety and Edge Radius
- Perfect Phylogeny (Bounded Number of States)
- Phylogenetic Tree Construction from a Distance Matrix

Acknowledgments JJ was funded by the Hakubi Project at Kyoto University and KAKENHI grant number 26330014.

Recommended Reading

1. Aho AV, Sagiv Y, Szymanski TG, Ullman JD (1981) Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. SIAM J Comput 10(3):405–421
2. Byrka J, Gawrychowski P, Huber KT, Kelk S (2010) Worst-case optimal approximation algorithms for maximizing triplet consistency within phylogenetic networks. J Discret Algorithms 8(1):65–75
3. Byrka J, Guillemot S, Jansson J (2010) New results on optimizing rooted triplets consistency. Discret Appl Math 158(11):1136–1147
4. Chao K-M, Chu A-C, Jansson J, Lemence RS, Mancheron A (2012) Asymptotic limits of a new type of maximization recurrence with an application to bioinformatics. In: Proceedings of the 9th annual conference on theory and applications of models of computation (TAMC 2012), Beijing, pp 177–188
5. Choy C, Jansson J, Sadakane K, Sung W-K (2005) Computing the maximum agreement of phylogenetic networks. Theor Comput Sci 335(1):93–107
6. Gusfield D, Eddhu S, Langley C (2003) Efficient reconstruction of phylogenetic networks with constrained recombination. In: Proceedings of computational systems bioinformatics (CSB2003), Stanford, pp 363–374
7. Habib M, To T-H (2012) Constructing a minimum phylogenetic network from a dense triplet set. J Bioinform Comput Biol 10(5):1250013
8. He Y-J, Huynh TND, Jansson J, Sung W-K (2006) Inferring phylogenetic relationships avoiding forbidden rooted triplets. J Bioinform Comput Biol 4(1):59–74
9. Huber KT, van Iersel L, Kelk S, Suchecki R (2011) A practical algorithm for reconstructing level-1 phylogenetic networks. IEEE/ACM Trans Comput Biol Bioinform 8(3):635–649

10. Jansson J, Sung W-K (2006) Inferring a level-1 phylogenetic network from a dense set of rooted triplets. Theor Comput Sci 363(1):60–68
11. Jansson J, Nguyen NB, Sung W-K (2006) Algorithms for combining rooted triplets into a galled phylogenetic network. SIAM J Comput 35(5):1098–1121
12. Jiang T, Kearney P, Li M (2001) A polynomial time approximation scheme for inferring evolutionary trees from quartet topologies and its application. SIAM J Comput 30(6):1942–1961
13. Kannan S, Lawler E, Warnow T (1996) Determining the evolutionary tree using experiments. J Algorithms 21(1):26–50
14. van Iersel L, Kelk S (2011) Constructing the simplest possible phylogenetic network from triplets. Algorithmica 60(2):207–235
15. Wang L, Zhang K, Zhang L (2001) Perfect phylogenetic networks with recombination. J Comput Biol 8(1):69–78

All Pairs Shortest Paths in Sparse Graphs

Seth Pettie
Electrical Engineering and Computer Science (EECS) Department, University of Michigan, Ann Arbor, MI, USA

Keywords

Quickest route; Shortest route

Years and Authors of Summarized Original Work

2004; Pettie

Problem Definition

Given a communications network or road network, one of the most natural algorithmic questions is how to determine the shortest path from one point to another. The *all pairs* shortest path problem (APSP) is, given a directed graph $G = (V, E, l)$, to determine the distance and shortest path between every pair of vertices, where $|V| = n$, $|E| = m$, and $l : E \to \mathbb{R}$ is the edge length (or weight) function. The output is in the form of two $n \times n$ matrices: $D(u, v)$ is the distance from u to v and $S(u, v) = w$ if (u, w) is the first edge on a shortest path from u to v. The APSP problem is often contrasted with the *point-to-point* and *single source* (SSSP) shortest path problems. They ask for, respectively, the shortest path from a given source vertex to a given target vertex and all shortest paths from a given source vertex.

Definition of Distance

If ℓ assigns only non-negative edge lengths then the definition of distance is clear: $D(u, v)$ is the length of the minimum length path from u to v, where the length of a path is the total length of its constituent edges. However, if ℓ can assign negative lengths then there are several sensible notations of distance that depend on how negative length cycles are handled. Suppose that a cycle C has negative length and that $u, v \in V$ are such that C is reachable from u and v reachable from C. Because C can be traversed an arbitrary number of times when traveling from u to v, there is no shortest path from u to v using a finite number of edges. It is sometimes assumed a priori that G has no negative length cycles; however it is cleaner to define $D(u, v) = -\infty$ if there is no finite shortest path. If $D(u, v)$ is defined to be the length of the shortest *simple* path (no repetition of vertices) then the problem becomes NP-hard. (If all edges have length -1 then $D(u, v) = -(n-1)$ if and only if G contains a Hamiltonian path [7] from u to v.) One could also define distance to be the length of the shortest path without repetition of edges.

Classic Algorithms

The Bellman-Ford algorithm solves SSSP in $O(mn)$ time and under the assumption that edge lengths are non-negative, Dijkstra's algorithm solves it in $O(m + n \log n)$ time. There is a well known $O(mn)$-time shortest path preserving transformation that replaces any length function with a non-negative length function. Using this transformation and n runs of Dijkstra's algorithm gives an APSP algorithm running in $O(mn + n^2 \log n) = O(n^3)$ time. The Floyd-Warshall algorithm computes APSP in a more

direct manner, in $O(n^3)$ time. Refer to [4] for a description of these algorithms. It is known that APSP on complete graphs is asymptotically equivalent to (min, +) matrix multiplication [1], which can be computed by a non-uniform algorithm that performs $O(n^{2.5})$ numerical operations [6].

Integer-Weighted Graphs

Much recent work on shortest paths assume that edge lengths are integers in the range $\{-C, \ldots, C\}$ or $\{0, \ldots, C\}$. One line of research reduces APSP to a series of standard matrix multiplications. These algorithms are limited in their applicability because their running times scale linearly with C. There are faster SSSP algorithms for both non-negative edge lengths and arbitrary edge lengths. The former exploit the power of RAMs to sort in $o(n \log n)$ time and the latter are based on the *scaling* technique. See Zwick [20] for a survey of shortest path algorithms up to 2001.

Key Results

Pettie's APSP algorithm [12] adapts the *hierarchy* approach of Thorup [16] (designed for undirected, integer-weighted graphs) to general real-weighted directed graphs. Theorem 1 is the first improvement over the $O(mn+n^2 \log n)$ time bound of Dijkstra's algorithm on arbitrary real-weighted graphs.

Theorem 1 *Given a real-weighted directed graph, all pairs shortest paths can be solved in* $O(mn + n^2 \log\log n)$ *time.*

This algorithm achieves a logarithmic speedup through a trio of new techniques. The first is to exploit the necessary similarity between the SSSP trees emanating from nearby vertices. The second is a method for computing discrete approximate distances in real-weighted graphs. The third is a new hierarchy-type SSSP algorithm that runs in $O(m + n \log\log n)$ time when given suitably accurate approximate distances.

Theorem 1 should be contrasted with the time bounds of other hierarchy-type APSP algorithms [11, 14, 16].

Theorem 2 ([14], 2005) *Given a real-weighted undirected graph, APSP can be solved in* $O(mn \log\alpha(m, n))$ *time.*

Theorem 3 ([16], 1999) *Given an undirected graph* $G(V, E, l)$*, where* ℓ *assigns integer edge lengths in the range* $\{-2^{w-1}, \ldots, 2^{w-1} - 1\}$*, APSP can be solved in* $O(mn)$ *time on a RAM with w-bit word length.*

Theorem 4 ([13], 2002) *Given a real-weighted directed graph, APSP can be solved in polynomial time by an algorithm that performs* $O(mn \log\alpha(m, n))$ *numerical operations, where* α *is the inverse-Ackermann function.*

A secondary result of [12, 14] is that no *hierarchy-type* shortest path algorithm can improve on the $O(m + n \log n)$ running time of Dijkstra's algorithm.

Theorem 5 *Let* G *be an input graph such that the ratio of the maximum to minimum edge length is* r*. Any hierarchy-type SSSP algorithm performs* $\Omega(m + \min\{n \log n, n \log r\})$ *numerical operations if* G *is directed and* $\Omega(m + \min\{n \log n, n \log\log r\})$ *if* G *is undirected.*

Applications

Shortest paths appear as a subproblem in other graph optimization problems; the minimum weight perfect matching, minimum cost flow, and minimum mean-cycle problems are some examples. A well known commercial application of shortest path algorithms is finding efficient routes on road networks; see, for example, Google Maps, MapQuest, or Yahoo Maps.

Open Problems

The longest standing open shortest path problems are to improve the SSSP algorithms of Dijkstra's and Bellman-Ford on *real*-weighted graphs.

Problem 1 Is there an $o(mn)$ time SSSP or point-to-point shortest path algorithm for arbitrarily weighted graphs?

Problem 2 Is there an $O(m) + o(n \log n)$ time SSSP algorithm for directed, non-negatively weighted graphs? For undirected graphs?

A partial answer to Problem 2 appears in [14], which considers undirected graphs. Perhaps the most surprising open problem is whether there is any (asymptotic) difference between the complexities of the all pairs, single source, and point-to-point shortest path problems on arbitrarily weighted graphs.

Problem 3 Is point-to-point shortest paths easier than all pairs shortest paths on arbitrarily weighted graphs?

Problem 4 Is there a truly subcubic APSP algorithm, i.e., one running in time $O(n^{3-\epsilon})$? In a recent breakthrough on this problem, Williams [19] gave a new APSP algorithm running in $n^3/2^{\Theta(\sqrt{\log n/\log\log n})}$ time. Vassilevska Williams and Williams [17] proved that a truly subcubic algorithm for APSP would imply truly subcubic algorithms for other graph problems.

Experimental Results

See [5, 8, 15] for recent experiments on SSSP algorithms. On sparse graphs the best APSP algorithms use repeated application of an SSSP algorithm, possibly with some precomputation [15]. On dense graphs cache-efficiency becomes a major issue. See [18] for a cache conscious implementation of the Floyd-Warshall algorithm.

The trend in recent years is to construct a linear space data structure that can quickly answer exact or approximate point-to-point shortest path queries; see [2, 6, 9, 10].

Data Sets

See [5] for a number of U.S. and European road networks.

URL to Code

See [5].

Cross-References

- All Pairs Shortest Paths via Matrix Multiplication
- Single-Source Shortest Paths

Recommended Reading

1. Aho AV, Hopcroft JE, Ullman JD (1975) The design and analysis of computer algorithms. Addison-Wesley, Reading
2. Bast H, Funke S, Matijevic D, Sanders P, Schultes D (2007) In transit to constant shortest-path queries in road networks. In: Proceedings of the 9th workshop on algorithm engineering and experiments (ALENEX), San Francisco
3. Chan T (2007) More algorithms for all-pairs shortest paths in weighted graphs. In: Proceedings of the 39th ACM symposium on theory of computing (STOC), San Diego, pp 590–598
4. Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms. MIT, Cambridge
5. Demetrescu C, Goldberg AV, Johnson D (2006) 9th DIMACS implementation challenge – shortest paths. http://www.dis.uniroma1.it/~challenge9/
6. Fredman ML (1976) New bounds on the complexity of the shortest path problem. SIAM J Comput 5(1):83–89
7. Garey MR, Johnson DS (1979) Computers and intractability: a guide to NP-completeness. Freeman, San Francisco
8. Goldberg AV (2001) Shortest path algorithms: engineering aspects. In: Proceedings of the 12th international symposium on algorithms and computation (ISAAC), Christchurch. LNCS, vol 2223. Springer, Berlin, pp 502–513
9. Goldberg AV, Kaplan H, Werneck R (2006) Reach for A*: efficient point-to-point shortest path algorithms. In: Proceedings of the 8th workshop on algorithm engineering and experiments (ALENEX), Miami
10. Knopp S, Sanders P, Schultes D, Schulz F, Wagner D (2007) Computing many-to-many shortest paths using highway hierarchies. In: Proceedings of the 9th workshop on algorithm engineering and experiments (ALENEX), New Orleans
11. Pettie S (2002) On the comparison-addition complexity of all-pairs shortest paths. In: Proceedings of the 13th international symposium on algorithms and computation (ISAAC), Vancouver, pp 32–43

12. Pettie S (2004) A new approach to all-pairs shortest paths on real-weighted graphs. Theor Comput Sci 312(1):47–74
13. Pettie S, Ramachandran V (2002) Minimizing randomness in minimum spanning tree, parallel connectivity and set maxima algorithms. In: Proceedings of the 13th ACM-SIAM symposium on discrete algorithms (SODA), San Francisco, pp 713–722
14. Pettie S, Ramachandran V (2005) A shortest path algorithm for real-weighted undirected graphs. SIAM J Comput 34(6):1398–1431
15. Pettie S, Ramachandran V, Sridhar S (2002) Experimental evaluation of a new shortest path algorithm. In: Proceedings of the 4th workshop on algorithm engineering and experiments (ALENEX), San Francisco, pp 126–142
16. Thorup M (1999) Undirected single-source shortest paths with positive integer weights in linear time. J ACM 46(3):362–394
17. Vassilevska Williams V, Williams R (2010) Subcubic equivalences between path, matrix, and triangle problems. In Proceedings of the 51st IEEE symposium on foundations of computer science (FOCS), Los Alamitos, pp 645–654
18. Venkataraman G, Sahni S, Mukhopadhyaya S (2003) A blocked all-pairs shortest paths algorithm. J Exp Algorithms 8
19. Williams R (2014) Faster all-pairs shortest paths via circuit complexity. In Proceedings of the 46th ACM symposium on theory of computing (STOC), New York, pp 664–673
20. Zwick U (2001) Exact and approximate distances in graphs – a survey. In: Proceedings of the 9th European symposium on algorithms (ESA), Aarhus, pp 33–48. See updated version at http://www.cs.tau.ac.il/~zwick/

All Pairs Shortest Paths via Matrix Multiplication

Tadao Takaoka
Department of Computer Science and Software Engineering, University of Canterbury, Christchurch, New Zealand

Keywords

Algorithm analysis; All pairs shortest path problem; Bridging set; Matrix multiplication; Shortest path problem; Two-phase algorithm; Witness

Years and Authors of Summarized Original Work

2002; Zwick

Problem Definition

The all pairs shortest path (APSP) problem is to compute shortest paths between all pairs of vertices of a directed graph with nonnegative real numbers as edge costs. Focus is given on shortest distances between vertices, as shortest paths can be obtained with a slight increase of cost. Classically, the APSP problem can be solved in cubic time of $O(n^3)$. The problem here is to achieve a sub-cubic time for a graph with small integer costs.

A directed graph is given by $G = (V, E)$, where $V = \{1, \ldots, n\}$, the set of vertices, and E is the set of edges. The cost of edge $(i, j) \in E$ is denoted by d_{ij}. The (n, n)-matrix D is one whose (i, j) element is d_{ij}. It is assumed for simplicity that $d_{ij} > 0$ and $d_{ii} = 0$ for all $i \neq j$. If there is no edge from i to j, let $d_{ij} = \infty$. The cost, or distance, of a path is the sum of costs of the edges in the path. The length of a path is the number of edges in the path. The shortest distance from vertex i to vertex j is the minimum cost over all paths from i to j, denoted by d^*_{ij}. Let $D^* = \{d^*_{ij}\}$. The value of n is called the size of the matrices.

Let A and B are (n, n)-matrices. The three products are defined using the elements of A and B as follows: (1) Ordinary matrix product over a ring $C = AB$ (2) Boolean matrix product $C = A \cdot B$ (3) Distance matrix product $C = A \times B$, where

$$(1)\ c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \quad (2)\ c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj},$$

$$(3)\ c_{ij} = \min_{1 \le k \le n} \{a_{ik} + b_{kj}\}.$$

The matrix C is called a product in each case; the computational process is called multiplication, such as distance matrix multiplication. In those

three cases, k changes through the entire set $\{1,\ldots,n\}$. A partial matrix product of A and B is defined by taking k in a subset I of V. In other words, a partial product is obtained by multiplying a vertically rectangular matrix, $A(*,I)$, whose columns are extracted from A corresponding to the set I, and similarly a horizontally rectangular matrix, $B(I,*)$, extracted from B with rows corresponding to I. Intuitively, I is the set of check points k, when going from i to j.

The best algorithm [11] computes (1) in $O(n^\omega)$ time, where $\omega = 2.373$. This was recently achieved as improvement from $\omega = 2.376$ in [4] after more than two decades of interval. We use $\omega = 2.376$ to describe Zwick's result in this article. Three decimal points are carried throughout this article. To compute (2), Boolean values 0 and 1 in A and B can be regarded as integers and use the algorithm for (1), and convert nonzero elements in the resulting matrix to 1. Therefore, this complexity is $O(n^\omega)$. The witnesses of (2) are given in the witness matrix $W = \{w_{ij}\}$ where $w_{ij} = k$ for some k such that $a_{ik} \wedge b_{kj} = 1$. If there is no such k, $w_{ij} = 0$. The witness matrix $W = \{w_{ij}\}$ for (3) is defined by $w_{ij} = k$ that gives the minimum to c_{ij}. If there is an algorithm for (3) with $T(n)$ time, ignoring a polylog factor of n, the APSP problem can be solved in $\tilde{O}(T(n))$ time by the repeated squaring method, described as the repeated use of $D \leftarrow D \times D$ $O(\log n)$ times.

The definition here of computing shortest paths is to give a witness matrix of size n by which a shortest path from i to j can be given in $O(\ell)$ time where ℓ is the length of the path. More specifically, if $w_{ij} = k$ in the witness matrix $W = \{w_{ij}\}$, it means that the path from i to j goes through k. Therefore, a recursive function path(i,j) is defined by (path(i,k), k, path(k,j)) if path$(i,j) = k > 0$ and nil if path$(i,j) = 0$, where a path is defined by a list of vertices excluding endpoints. In the following sections, k is recorded in w_{ij} whenever k is found such that a path from i to j is modified or newly set up by paths from i to k and from k to j. Preceding results are introduced as a framework for the key results.

Alon-Galil-Margalit Algorithm

The algorithm by Alon, Galil, and Margalit [1] is reviewed. Let the costs of edges of the given graph be ones. Let $D^{(\ell)}$ be the ℓ-th approximate matrix for D^* defined by $d_{ij}^{(\ell)} = d_{ij}^*$ if $d_{ij}^* \le \ell$, and $d_{ij}^{(\ell)} = \infty$ otherwise. Let A be the adjacency matrix of G, that is, $a_{ij} = 1$ if there is an edge (i,j), and $a_{ij} = 0$, otherwise. Let $a_{ii} = 1$ for all i. The algorithm consists of two phases. In the first phase, $D^{(\ell)}$ is computed for $\ell = 1,\ldots,r$, by checking the (i,j) element of $A^\ell = \{a_{ij}^\ell\}$. Note that if $a_{ij}^\ell = 1$, there is a path from i to j of length ℓ or less. Since Boolean matrix multiplication can be computed in $O(n^\omega)$ time, the computing time of this part is $O(rn^\omega)$.

In the second phase, the algorithm computes $D^{(\ell)}$ for $\ell = r, \lceil \frac{3}{2}r \rceil, \left\lceil \frac{3}{2}\lceil \frac{3}{2}r \rceil \right\rceil, \ldots, n'$ by repeated squaring, where n' is the smallest integer in this sequence of ℓ such that $\ell \ge n$. Let $T_{i\alpha} = \{j | d_{ij}^{(\ell)} = \alpha\}$ and $I_i = T_{i\alpha}$ such that $|T_{i\alpha}|$ is minimum for $\lceil \ell/2 \rceil \le \alpha \le \ell$. The key observation in the second phase is that it is only needed to check k in I_i whose size is not larger than $2n/\ell$, since the correct distances between $\ell+1$ and $\lceil 3\ell/2 \rceil$ can be obtained as the sum $d_{ik}^{(\ell)} + d_{kj}^{(\ell)}$ for some k satisfying $\lceil \ell/2 \rceil \le d_{ik}^{(\ell)} \le \ell$. The meaning of I_i is similar to I for partial products except that I varies for each i. Hence, the computing time of one squaring is $O(n^3/\ell)$. Thus, the time of the second phase is given with $N = \lceil \log_{3/2} n/r \rceil$ by $O\left(\sum_{s=1}^{N} n^3/((3/2)^s r)\right) = O(n^3/r)$. Balancing the two phases with $rn^\omega = n^3/r$ yields $O(n^{(\omega+3)/2}) = O(n^{2.688})$ time for the algorithm with $r = O(n^{(3-\omega)/2})$.

Witnesses can be kept in the first phase in time polylog of n by the method in [2]. The maintenance of witnesses in the second phase is straightforward.

When a directed graph G whose edge costs are integers between 1 and M is given, where M is a positive integer, the graph G can be expanded to G' by creating up to $M-1$ new vertices for each vertex and replacing each edge by up to M edges with unit cost. Obviously, the problem for G can be solved by applying the above algorithm to G',

which takes $O\left((Mn)^{(\omega+3)/2}\right)$ time. This time is sub-cubic when $M < n^{0.116}$. The maintenance of witnesses has an extra polylog factor in each case.

For undirected graphs with unit edge costs, $\tilde{O}(n^{\omega})$ time is known in Seidel [9].

Takaoka Algorithm

When the edge costs are bounded by a positive integer M, a better algorithm can be designed than in the above as shown in Takaoka [10]. Romani's algorithm [7] for distance matrix multiplication is reviewed briefly.

Let A and B be (n, m) and (m, n) distance matrices whose elements are bounded by M or infinite. Let the diagonal elements be 0. A and B are converted into A' and B' where $a'_{ij} = (m+1)^{M-a_{ij}}$, if $a_{ij} \neq \infty, 0$, otherwise, and $b'_{ij} = (m+1)^{M-b_{ij}}$, if $b_{ij} \neq \infty, 0$, otherwise.

Let $C' = A'B'$ be the product by ordinary matrix multiplication and $C = A \times B$ be that by distance matrix multiplication. Then, it holds that

$$c'_{ij} = \sum_{k=1}^{m} (m+1)^{2M-(a_{ik}+b_{kj})},$$

$$c_{ij} = 2M - \lfloor \log_{m+1} c'_{ij} \rfloor.$$

This distance matrix multiplication is called (n, m)-Romani. In this section, the above multiplication is used with square matrices, that is, (n, n)-Romani is used. In the next section, the case where $m < n$ is dealt with.

C can be computed with $O(n^{\omega})$ arithmetic operations on integers up to $(n+1)^M$. Since these values can be expressed by $O(M \log n)$ bits and Schönhage and Strassen's algorithm [8] for multiplying k-bit numbers takes $O(k \log k \log\log k)$ bit operations, C can be computed in $O(n^{\omega} M \log n \log(M \log n) \log\log (M \log n))$ time, or $\tilde{O}(Mn^{\omega})$ time.

The first phase is replaced by the one based on (n, n)-Romani and the second phase is modified based on path lengths, not distances.

Note that the bound M is replaced by ℓM in the distance matrix multiplication in the first phase. Ignoring polylog factors, the time for the first phase is given by $\tilde{O}(n^{\omega} r^2 M)$. It is assumed that M is $O(n^k)$ for some constant k. Balancing this complexity with that of second phase, $O(n^3/r)$, yields the total computing time of $\tilde{O}(n^{(6+\omega)/3} M^{1/3})$ with the choice of $r = O(n^{(3-\omega)/3} M^{-1/3})$. The value of M can be almost $O(n^{0.624})$ to keep the complexity within sub-cubic.

Key Results

Zwick improved the Alon-Galil-Margalit algorithm in several ways. The most notable is an improvement of the time for the APSP problem with unit edge costs from $O(n^{2.688})$ to $O(n^{2.575})$. The main accelerating engine in Alon-Galil-Margalit [1] was the fast Boolean matrix multiplication and that in Takaoka [10] was the fast distance matrix multiplication by Romani, both powered by the fast matrix multiplication of square matrices.

In this section, the engine is the fast distance matrix multiplication by Romani powered by the fast matrix multiplication of rectangular matrices given by Coppersmith [3] and Huang and Pan [5]. Suppose the product of (n, m) matrix and (m, n) matrix can be computed with $O(n^{\omega(1,\mu,1)})$ arithmetic operations, where $m = n^{\mu}$ with $0 \leq \mu \leq 1$. Several facts such as $O(n^{\omega(1,1,1)}) = O(n^{2.376})$ and $O(n^{\omega(1,0.294,1)}) = \tilde{O}(n^2)$ are known. To compute the product of (n, n) square matrices, $n^{1-\mu}$ matrix multiplications are needed, resulting in $O(n^{\omega(1,\mu,1)+1-\mu})$ time, which is reformulated as $O(n^{2+\mu})$, where μ satisfies the equation $\omega(1, \mu, 1) = 2\mu + 1$. Also, the upper bound of $\omega(1, \mu, 1)$ is given by

$$\omega(1, \mu, 1) = 2, \text{ if } 0 \leq \mu \leq \alpha$$

$$\omega(1, \mu, 1) = 2 + (\omega - 2)(\mu - \alpha)/(1 - \alpha), \text{ if } \alpha \leq \mu \leq 1$$

The best known value for μ, when [12] was published, was $\mu = 0.575$, derived from the above formulae, $\alpha > 0.294$ and $\omega < 2.376$. So, the time becomes $O(n^{2.575})$, which is not as

good as $O(n^{2.376})$. Thus, we use the algorithm for rectangular matrices in the following.

The above algorithm for rectangular matrix multiplication is incorporated into (n,m)-Romani with $m = n^{\mu}$ and $M = n^t$, and the computing time of $\tilde{O}(Mn^{\omega(1,\mu,1)})$. The next step is how to incorporate (n,m)-Romani into the APSP algorithm. The first algorithm is a monophase algorithm based on repeated squaring, similar to the second phase of the algorithm in [1]. To take advantage of rectangular matrices in (n,m)-Romani, the following definition of the bridging set is needed, which plays the role of the set I in the partial distance matrix product in section "Problem Definition."

Let $\delta(i,j)$ be the shortest distance from i to j, and $\eta(i,j)$ be the minimum length of all shortest paths from i to j. A subset I of V is an ℓ-bridging set if it satisfies the condition that if $\eta(i,j) \geq \ell$, there exists $k \in I$ such that $\delta(i,j) = \delta(i,k) + \delta(k,j)$. I is a strong ℓ-bridging set if it satisfies the condition that if $\eta(i,j) \geq \ell$, there exists $k \in I$ such that $\delta(i,j) = \delta(i,k) + \delta(k,j)$ and $\eta(i,j) = \eta(i,k) + \eta(k,j)$. Note that those two sets are the same for a graph with unit edge costs.

Note that if $(2/3)\ell \leq \mu(i,j) \leq \ell$ and I is a strong $\ell/3$-bridging set, there is a $k \in I$ such that $\delta(i,j) = \delta(i,k) + \delta(k,j)$ and $\mu(i,j) = \mu(i,k) + \mu(k,j)$. With this property of strong bridging sets, (n,m)-Romani can be used for the APSP problem in the following way. By repeated squaring in a similar way to Alon-Galil-Margalit, the algorithm computes $D^{(\ell)}$ for $\ell = 1, \lceil \frac{3}{2} \rceil, \left\lceil \frac{3}{2} \left\lceil \frac{3}{2} \right\rceil \right\rceil, \ldots, n'$, where n' is the first value of ℓ that exceeds n, using various types of set I described below. To compute the bridging set, the algorithm maintains the witness matrix with extra polylog factor in the complexity. In [12], there are three ways for selecting the set I. Let $|I| = n^r$ for some r such that $0 \leq r \leq 1$.

1. Select $9n \ln n/\ell$ vertices for In from V at random. In this case, it can be shown that the algorithm solves the APSP problem with high probability, i.e., with $1 - 1/n^c$ for some constant $c > 0$, which can be shown to be 3. In other words, I is a strong $\ell/3$-bridging set with high probability. The time T is dominated by (n,m)-Romani. It holds that $T = \tilde{O}(\ell M n^{\omega(1,r,1)})$, since the magnitude of matrix elements can be up to ℓM. Since $m = O(n \ln n/\ell) = n^r$, it holds that $\ell = \tilde{O}(n^{1-r})$, and thus $T = O(Mn^{1-r}n^{\omega(1,r,1)})$. When $M = 1$, this bound on r is $\mu = 0.575$, and thus $T = O(n^{2.575})$. When $M = n^t \geq 1$, the time becomes $O(n^{2+\mu(t)})$, where $t \leq 3 - \omega = 0.624$ and $\mu = \mu(t)$ satisfies $\omega(1,\mu,1) = 1 + 2\mu - t$. It is determined from the best known $\omega(1,\mu,1)$ and the value of t. As the result is correct with high probability, this is a randomized algorithm.
2. Consider the case of unit edge costs here. In (1), the computation of witnesses is an extra thing, i.e., not necessary if only shortest distances are needed. To achieve the same complexity in the sense of an exact algorithm, not a randomized one, the computation of witnesses is essential. As mentioned earlier, maintenance of witnesses, that is, matrix W, can be done with an extra polylog factor, meaning the analysis can be focused on Romani within the $\tilde{O}$-notation. Specifically, I is selected as an $\ell/3$-bridging set, which is strong with unit edge costs. To compute I as an $O(\ell)$-bridging set, obtain the vertices on the shortest path from i to j for each i and j using the witness matrix W in $O(\ell)$ time. After obtaining those n^2 sets spending $O(\ell n^2)$ time, it is shown in [12] how to obtain a $O(\ell)$-bridging set of $O(n \ln n/\ell)$ size within the same time complexity. The process of obtaining the bridging set must stop at $\ell = n^{1/2}$ as the process is too expensive beyond this point, and thus, the same bridging set is used beyond this point. The time before this point is the same as that in (1) and that after this point is $\tilde{O}(n^{2.5})$. Thus, this is a two-phase algorithm.
3. When edge costs are positive and bounded by $M = n^t > 0$, a similar procedure can be used to compute an $O(\ell)$-bridging set of $O(n \ln n/\ell)$ size in $\tilde{O}(\ell n^2)$ time. Using the bridging set, the APSP problem can be solved in $\tilde{O}(n^{2+\mu(t)})$ time in a similar way to (1).

The result can be generalized into the case where edge costs are between $-M$ and M within the same time complexity by modifying the procedure for computing an ℓ-bridging set, provided there is no negative cycle. The details are shown in [12].

Applications

The eccentricity of a vertex v of a graph is the greatest distance from v to any other vertices. The diameter of a graph is the greatest eccentricity of any vertices. In other words, the diameter is the greatest distance between any pair of vertices. If the corresponding APSP problem is solved, the maximum element of the resulting matrix is the diameter.

Open Problems

Recently, LeGall [6] discovered an algorithm for multiplying rectangular matrices with $\omega(1, 0.530, 1) < 2.060$, which gives the upper bound $\mu < 0.530$. This improves the complexity of APSP with unit edge costs from $O(n^{2.575})$ by Zwick to $O(n^{2.530})$ in the same framework as that of Zwick in this article. Two major challenges are stated here among others. The first is to improve the complexity of $\tilde{O}(n^{2.530})$ for the APSP with unit edge costs.

The other is to improve the bound of $M < O(n^{0.624})$ for the complexity of the APSP with integer costs up to M to be sub-cubic.

Cross-References

▸ All Pairs Shortest Paths in Sparse Graphs

Recommended Reading

1. Alon N, Galil Z, Margalit O (1991) On the exponent of the all pairs shortest path problem. In: Proceedings of the 32th IEEE FOCS, San Juan, pp 569–575. Also JCSS 54 (1997), pp 255–262
2. Alon N, Galil Z, Margalit O, Naor M (1992) Witnesses for Boolean matrix multiplication and for shortest paths. In: Proceedings of the 33th IEEE FOCS, Pittsburgh, pp 417–426
3. Coppersmith D (1997) Rectangular matrix multiplication revisited. J Complex 13:42–49
4. Coppersmith D, Winograd S (1990) Matrix multiplication via arithmetic progressions. J Symb Comput 9:251–280
5. Huang X, Pan VY (1998) Fast rectangular matrix multiplications and applications. J Complex 14:257–299
6. Le Gall F (2012) Faster algorithms for rectangular matrix multiplication. In: Proceedings of the 53rd FOCS, New Brunswick, pp 514–523
7. Romani F (1980) Shortest-path problem is not harder than matrix multiplications. Inf Proc Lett 11:134–136
8. Schönhage A, Strassen V (1971) Schnelle Multiplikation Großer Zahlen. Computing 7:281–292
9. Seidel R (1992) On the all-pairs-shortest-path problem. In: Proceedings of the 24th ACM STOC, Victoria, pp 745–749. Also JCSS 51 (1995), pp 400–403
10. Takaoka T (1998) Sub-cubic time algorithms for the all pairs shortest path problem. Algorithmica 20:309–318
11. Williams VV (2012) Multiplying matrices faster than Coppersmith-Winograd. In: Proceedings of the 44th symposium on theory of computing, ACM STOC, New York, pp 887–898
12. Zwick U (2002) All pairs shortest paths using bridging sets and rectangular matrix multiplication. J ACM 49(3):289–317

All-Distances Sketches

Edith Cohen
Tel Aviv University, Tel Aviv, Israel
Stanford University, Stanford, CA, USA

Keywords

Distance distribution; Distinct counting; Graph analysis; Influence; Nodes similarity; Summary structures

Years and Authors of Summarized Original Work

1997, 2014; Cohen
2002; Palmer, Gibbons, Faloutsos
2004, 2007; Cohen, Kaplan

Problem Definition

All-distances sketches (The term *least element lists* was used in [3]; the terms MV/D lists and Neighborhood summaries were used in [6].) are randomized summary structures of the distance relations of nodes in a graph. The graph can be directed or undirected, and edges can have uniform or general nonnegative weights.

Preprocessing Cost

A set of sketches, ADS(v) for each node v, can be computed efficiently, using a near-linear number of edge traversals. The sketch sizes are well concentrated, with logarithmic dependence on the total number of nodes.

Supported Queries

The sketches support approximate distance-based queries, which include:

- ***Distance distribution***: The query specifies a node v and value $d \geq 0$ and returns the cardinality $|N_d(v)|$ of the d-neighborhood of v $N_d(v) = \{u \mid d_{vu} \leq d\}$, where d_{uv} is the shortest path distance from u to v. We are interested in estimating $|N_d(v)|$ from ADS(v).

  A related property is the *effective diameter* of the graph, which is a quantile of the distance distribution of all node pairs; we are interested in computing an estimate efficiently.
- ***Closeness centrality*** (distance-decaying) is defined for a node v, a monotone nonincreasing function $\alpha(x) \geq 0$ (where $\alpha(+\infty) \equiv 0$), and a nonnegative function $\beta(u) \geq 0$:

$$C_{\alpha,\beta}(v) = \sum_u \alpha(d_{vu})\beta(u) \,. \tag{1}$$

  The function α specifies the decay of relevance with distance and the function β weighs nodes based on metadata to focus on a topic or property of relevance. Neighborhood cardinality is a special case obtained using $\beta \equiv 1$ and $\alpha(x) = 1$ if $x \leq d$ and $\alpha(x) = 0$ otherwise. The number of reachable nodes from v is obtained using $\alpha(x) \equiv 1$. Also studied were exponential decay $\alpha(x) = 2^{-x}$ with distance [10], the (inverse) harmonic mean of distances $\alpha(x) = 1/x$ [1, 14], and general decay functions [4,6]. We would like to estimate $C_{\alpha,\beta}(v)$ from ADS(v).
- ***Closeness similarity*** [8] relates a pair of nodes based on the similarity of their distance relations to all other nodes.

$$\underset{\alpha,\beta}{\mathrm{SIM}}(v,u) = \frac{\sum_j \alpha(\max\{d_{u,j}, d_{v,j}\})\beta(j)}{\sum_j \alpha(\min\{d_{u,j}, d_{v,j}\})\beta(j)} \,. \tag{2}$$

  We would like to estimate $\mathrm{SIM}_{\alpha,\beta}(v,u) \in [0,1]$ from ADS(v) and ADS(u).
- ***Timed influence*** of a seed set S of nodes depends on the set of distances from S to other nodes. Intuitively, when edge lengths model transition times, the distance is the "elapsed time" needed to reach the node from S. Influence is higher when distances are shorter:

$$\underset{\alpha,\beta}{\mathrm{INF}}(S) = \sum_j \alpha(\min_{i \in S} d_{ij})\beta(j) \,. \tag{3}$$

  We would like to estimate $\mathrm{INF}_{\alpha,\beta}(S)$ from the sketches $\{\mathrm{ADS}(v) \mid v \in S\}$.
- ***Approximate distance oracles:*** For two nodes v, u, use ADS(u) and ADS(v) to estimate $d_{u,v}$.

Key Results

We provide a precise definition of ADSs, overview algorithms for scalable computation, and finally discuss estimators.

Definition

The ADS of a node v, ADS(v), is a set of node ID and distance pairs (u, d_{vu}). The included nodes are a sample of the nodes reachable from v. The sampling is such that the inclusion probability of a node is inversely proportional to its Dijkstra rank (nearest neighbor rank). That is, the probability that the ith closest node is sampled is proportional to $1/i$.

The ADSs are defined with respect to random mappings/permutations r of the set of all nodes

and come in three flavors: bottom-k, k-mins, and k-partition. The integer parameter k determines a tradeoff between the sketch size and computation time on one hand and the information and estimate quality on the other. For simplicity, we assume here that distances d_{vu} are unique for different u (using tie breaking). We use the notation $\Phi_{<u}(v)$ for the set of nodes that are closer to node v than node u and $\pi_{vu} = 1 + |\Phi_{<u}(v)|$ for the *Dijkstra rank* of u with respect to v (u is the π_{vu} closest node from v). For a set N and a numeric function $r : N$, the function $\mathrm{k}_r\mathrm{th}(N)$ returns the kth smallest value in the range of r on N. If $|N| < k$, then we define $\mathrm{k}_r\mathrm{th}(N)$ to be the supremum of the range of r.

A *bottom-k* ADS [3,7] is defined with respect to a single random permutation r. ADS(v) includes a node u if and only if the rank $r(u)$ is one of the k smallest ranks among nodes that are at least as close to v:

$$u \in \mathrm{ADS}(v) \iff r(u) < \mathrm{k}_r\mathrm{th}(\Phi_{<u}(v)). \quad (4)$$

A *k-partition* ADS (implicit in) [2] is defined with respect to a random partition $\mathrm{BUCKET} : V \to [k]$ of the nodes to k subsets and a random permutation r. ADS(v) includes u if and only if u has the smallest rank among nodes in its bucket that are at least as close to v.

$$u \in \mathrm{ADS}(v) \iff r(u) < \min\left\{ r(h) \mid \begin{array}{l} \mathrm{BUCKET}(h) = \mathrm{BUCKET}(u) \\ \wedge\, h \in \Phi_{<u}(v) \end{array} \right\}.$$

A *k-mins* ADS [3,15] is k bottom-1 ADSs, defined with respect to k independent permutations.

It is often convenient to specify the ranks $r(j)$ and (for k-partition ADSs) the bucket $\mathrm{BUCKET}(j)$ using random hash functions, so they are readily available from the node ID. The same randomization is used for all nodes, which results in the sketches being coordinated. This means that a node sampled in one sketch is more likely to be included in other sketches. Coordination is an artifact of scalable computation of the sketches but also facilitates more accurate similarity and influence queries.

Relation to MinHash Sketches

All-distances sketches are related to MinHash sketches: ADS(v) is the union of the MinHash sketches of the neighborhoods $N_d(v)$, for all possible values of d. We explain how a MinHash sketch of a neighborhood $N_d(v)$ can be obtained from ADS(v). From a k-mins ADS, we obtain a k-mins MinHash sketch of $N_d(v)$, which includes for each of the k permutations r the value $x \leftarrow \min_{u \in N_d(v)} r(u)$. Note that x is the minimum rank of a node of distance at most d in the respective bottom-1 ADS defined for r. The k minimum rank values $x^{(t)}$ $t \in [k]$ we obtain from the different permutations are the k-mins MinHash sketch of $N_d(v)$. We now consider obtaining a bottom-k MinHash sketch of $N_d(v)$ from a bottom-k ADS(v). The MinHash sketch of $N_d(v)$ includes the k nodes of minimum rank in $N_d(v)$, which are also the k nodes of minimum rank in ADS(v) within distance at most d. Finally, a k-partition MinHash sketch of $N_d(v)$ is similarly obtained from a k-partition ADS by taking, for each bucket $i \in [k]$, the smallest rank value of a node in bucket i that is in $N_d(v)$. This is also the smallest value in ADS(v) over nodes in bucket i that have distance at most d from v.

Direction

For directed graphs, influence, centrality, and closeness similarity queries can be defined with respect to either forward or reversed distances. Accordingly, we can separately consider for each node v the *forward* ADS and the *backward* ADS of each node, which are defined respectively using forward or reverse paths from v.

Node Weights

All the ADS flavors can be extended to be with respect to specified node weights $\beta(v) \geq 0$ [3,4]. This makes queries formulated with respect to the weights more efficient.

Algorithms

There are two meta-approaches for scalable computation of an ADS set. The first approach,

PRUNEDDIJKSTRA (Algorithm 1), performs pruned applications of Dijkstra's single-source shortest paths algorithm (BFS when unweighted) [3, 7]. The second approach, DP (implicit in [2, 15]), applies to unweighted graphs and is based on dynamic programming or Bellman-Ford shortest paths computation. LOCALUPDATES (Algorithm 2) [4] extends DP to weighted graphs. LOCALUPDATES is node-centric and is appropriate for MapReduce or similar platforms, but can incur more overhead than PRUNEDDIJKSTRA. The algorithms are presented for bottom-k sketches, but both approaches can be easily adopted to work with all three ADS flavors.

Algorithm 1: ADS set for G via PRUNEDDIJKSTRA

```
1 for u by increasing r(u) do
2     Perform Dijkstra from u on G^T (the transpose graph)
3     foreach scanned node v do
4         if |{(x, y) ∈ ADS(u) | y < d_vu}| = k then
5             prune Dijkstra at v
6         else
7             ADS(v) ← ADS(v) ∪ {(r(u), d_vu)}
```

Both PRUNEDDIJKSTRA and DP can be performed in $O(km \log n)$ time (on unweighted graphs) on a single processor in main memory, where n and m are the number of nodes and edges in the graph. These algorithms maintain a partial ADS for each node, as entries of node ID and distance pairs. ADS(v) is initialized with the pair $(v, 0)$. The basic operation we use is *edge relaxation*: when relaxing (v, u), ADS(v) is updated using ADS(u). For bottom-k, the relaxation modifies ADS(v) when ADS(u) contains a node i such that $r(i)$ is smaller than the kth smallest rank among nodes in ADS(v) with distance at most $d_{ui} + w_{vu}$ from v. Both PRUNEDDIJKSTRA and DP perform relaxations in an order which guarantees that inserted entries are part of the final ADS, that is, there are no other nodes that are both closer and have lower rank: PRUNEDDIJKSTRA iterates over all nodes in increasing rank, runs Dijkstra's algorithm from the node on the transpose graph, and prunes at nodes when the ADS is not updated. DP performs iterations, where in each iteration, all edges (v, u), such that ADS(v) was updated in the previous step, are relaxed. Therefore, entries are inserted by increasing distance.

Algorithm 2: ADS set for G via LOCALUPDATES

```
   // Initialization
1  for u do
2      ADS(u) ← {(r(u), 0)}
   // Propagate updates (r, d) at node u
3  if (r, d) is added to ADS(u) then
4      foreach y | (u, y) ∈ G do
5          send (r, d + w(u, y)) to y
   // Process update (r, d) received at u
6  if node u receives (r, d) then
7      if r < k^th_x{(x, y) ∈ ADS(u) | y < d} then
8          ADS(u) ← ADS(u) ∪ {(r(v), d)}
       // Clean-up ADS(u)
9      for entries (x, y) ∈ ADS(u) | y > d by increasing y do
10         if x > k^th_h{(h, z) ∈ ADS(u) | z < y} then
11             ADS(u) ← ADS(u) \ (x, y)
```

Estimation

Distance Distribution

Neighborhood cardinality queries for a node v, and $d \geq 0$ can be estimated with a small relative error from ADS(v). The generic estimator extracts a MINHASH sketch of the neighborhood $N_d(v)$ from ADS(v) and applies a MINHASH cardinality estimator to this sketch. This approach was used in [3, 7, 15]. A nearly optimal estimator, the Historic Inverse Probability (HIP) estimator [4], has a factor 2 improvement in variance by using all information in the ADS instead of just the MINHASH sketch. HIP works by considering for each entry (u, d) in the sketch, the *HIP threshold probability*, which is the prob-

ability, under randomly drawn rank for the node u, but fixing ranks of all other nodes, that the entry is included in the sketch. The entry then obtains an adjusted weight that is the inverse of the HIP threshold probability. Neighborhood cardinality can be estimated by the sum of the adjusted weights of ADS entries that fall in the neighborhood.

Closeness Centrality (Distance-Decaying)

$C_{\alpha,\beta}(v)$ can be estimated from ADS(v) with a small relative error when the set of ADSs is computed with respect to β. Estimators using MINHASH sketches were given in [6]. The tighter HIP estimator in [4] simply sums, over entries $(u, d) \in$ ADS(v), the product of the adjusted weight of the entry and $\alpha(d)\beta(u)$.

Closeness Similarity and Influence

When the sketches are computed with respect to β, the closeness similarity of two nodes u and v can be estimated from ADS(u) and ADS(v) within a small additive ϵ [8]. The influence of a set of nodes S can be estimated from $\{\text{ADS}(v) \mid v \in S\}$ to within a small relative error [9]. These estimators are instances of the L* estimator [5] applied with the HIP inclusion probabilities [4].

Approximate Distance Oracles

An upper bound on the distance of two nodes u, v can be computed from ADS(u) and ADS(v) [8]. This is done by looking at the minimum, over nodes h that are in the intersection ADS(u) $\cap$ ADS(v), of $d_{uh} + d_{hv}$. When the graph is undirected, the oracle has worst-case quality guarantees that match the distance oracle of [16] (oracle time can be improved by looking only at nodes in the few ADS entries that correspond to $k = 1$). We note that observed quality in practice (using the full oracle) tends to have a small relative error [8].

Applications

Massive graphs, with billions of edges, are prevalent and model web graphs and social networks. Centralities, similarities, influence, and distances are basic data analysis tasks on these graphs. ADSs are a powerful tool for scalable analysis of very large graphs.

Extensions

An ADS can be viewed as a MINHASH sketch constructed from a stream, where all updates are recorded. This means that the HIP estimator [4] can be applied for distinct counting on streams, obtaining improved performance over estimators applied to the MINHASH sketch alone [11, 12].

In a graph context, ADS(v) is a recording of all updates to a MINHASH sketch obtained by sweeping through nodes in increasing distance from v. More generally, we can construct ADS for other settings and apply the same estimation machinery. One example is Euclidean distances [6, 13]. Another example is constructing a *combined ADS* of multiple graphs for the application of timed influence oracles [9].

Cross-References

▸ All-Distances Sketches can be viewed as an extension of ▸ Min-Hash Sketches. They are also ▸ Coordinated Sampling.

Recommended Reading

1. Boldi P, Vigna S (2014) Axioms for centrality. Internet Math 10:222–262
2. Boldi P, Rosa M, Vigna S (2011) HyperANF: approximating the neighbourhood function of very large graphs on a budget. In: WWW, Hyderabad
3. Cohen E (1997) Size-estimation framework with applications to transitive closure and reachability. J Comput Syst Sci 55:441–453
4. Cohen E (2014) All-distances sketches, revisited: HIP estimators for massive graphs analysis. In: PODS. ACM. http://arxiv.org/abs/1306.3284
5. Cohen E (2014) Estimation for monotone sampling: competitiveness and customization. In: PODC. ACM. http://arxiv.org/abs/1212.0243, full version http://arxiv.org/abs/1212.0243
6. Cohen E, Kaplan H (2007) Spatially-decaying aggregation over a network: model and algorithms. J Comput Syst Sci 73:265–288. Full version of a SIGMOD 2004 paper
7. Cohen E, Kaplan H (2007) Summarizing data using bottom-k sketches. In: PODC, Portland. ACM

8. Cohen E, Delling D, Fuchs F, Goldberg A, Goldszmidt M, Werneck R (2013) Scalable similarity estimation in social networks: closeness, node labels, and random edge lengths. In: COSN, Boston. ACM
9. Cohen E, Delling D, Pajor T, Werneck RF (2014) Timed influence: computation and maximization. Manuscript. ArXiv 1410.6976
10. Dangalchev C (2006) Residual closeness in networks. Phisica A 365:556–564
11. Flajolet P, Martin GN (1985) Probabilistic counting algorithms for data base applications. J Comput Syst Sci 31:182–209
12. Flajolet P, Fusy E, Gandouet O, Meunier F (2007) Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In: Analysis of algorithms (AOFA), Juan des Pins
13. Guibas LJ, Knuth DE, Sharir M (1992) Randomized incremental construction of Delaunay and Voronoi diagrams. Algorithmica 7:381–413
14. Opsahl T, Agneessens F, Skvoretz J (2010) Node centrality in weighted networks: generalizing degree and shortest paths. Soc Netw 32. http://toreopsahl.com/2010/03/20/
15. Palmer CR, Gibbons PB, Faloutsos C (2002) ANF: a fast and scalable tool for data mining in massive graphs. In: KDD, Edmonton
16. Thorup M, Zwick U (2001) Approximate distance oracles. In: Proceedings of the 33th annual ACM symposium on theory of computing, Crete, pp 183–192

Alternate Parameterizations

Neeldhara Misra
Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India

Keywords

Above guarantee; Complexity ecology of parameters; Dual parameters; Structural parameterization

Years and Authors of Summarized Original Work

2013; Fellows, Jansen, Rosamond
2014; Lokshtanov, Narayanaswamy, Raman, Ramanujan, Saurabh
2014; Marx, Pilipczuk

Problem Definition

A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$. Such a problem is said to be *fixed-parameter tractable* if there is an algorithm that decides if $(x,k) \in L$ in time $f(k)|X|^{O(1)}$. For attacking an intractable problem within the multivariate algorithmic framework, a necessary first step is to identify some reasonable parameters. The relevance of an FPT algorithm will depend on the quality of the choice of parameters. The first objective is of a practical concern: the choice of parameter should not "cheat," that is, it should be a choice that leads to tractability in the context of instances that are relevant to real-world applications. On the other hand, the parameter should also lend a perspective that is useful to the algorithm designer, usually by providing additional structural insights, thereby making an otherwise unwieldy problem manageable. Finally, the parameter itself should be accessible, in the sense that it should either typically accompany the input or be easy to compute from the input.

For a combinatorial optimization problem, the size of the desired solution is a natural parameter. For a minimization problem, it is usually reasonable to assume that this parameter is also small in practice. For a maximization problem, the *dual* parameter, which is the difference from the best possible upper bound on the optimum, is also a natural choice. For example, consider the problem of satisfying at least k clauses of a CNF formula. Here, the standard parameter would be k, while the dual parameter would be $(m-k)$: in other words, can we satisfy *all but* k clauses in the formula? In the rest of this section, we broadly describe the other possibilities for parameters.

Structural Parameterizations

Structural parameters are a considered attempt at acknowledging that various aspects of an instance influence its complexity. A classic example is ML-type checking [11], which is an NP-complete problem but can be resolved in time $O(2^k)n^{O(1)}$, where k is the maximum nesting depth of the

input program. Fortunately, nesting depths of most programs are no more than four or five; the algorithm proposed is entirely adequate for real-world instances.

Since every problem context is inherently suggestive of several possible parameters, we are only able to describe a few illustrative examples. In the context of graph problems, *width parameters* such as treewidth, cliquewidth, and rankwidth have enjoyed immense success. The notion of *treewidth* is particularly popular because of a number of real-world instances that are known to exhibit small treewidth, and on the other hand, the theoretical foundations of algorithms on graphs of bounded treewidth are extremely well established and actively developed (see, e.g., [3, Chapter 7]). An analogous notion for treewidth for directed graphs remains elusive, although several proposals with varying merits exist in the literature [7].

Special graph classes, such as interval graphs, chordal graphs, planar graphs, and so on, have been extensively studied, and most hard problems turn out to be tractable on these classes. For an arbitrary graph, one might hope that the tractability carries over if the graph is "close enough" to being, say, a chordal graph. An increasingly popular program involves considering distance-to-$\mathcal{C}$ parameterizations, where $\mathcal{C}$ is a class of graphs on which the problem of interest is easily solvable. For instance, we might let k be the size of a smallest subset of vertices whose removal makes the input graph a member of $\mathcal{C}$. Other measures of closeness, using operations like addition, removal, or contraction of edges, are also frequently considered.

In the context of satisfiability and constraint satisfaction also, the notion of distance from tractable subclasses has garnered much attention in recent times. This is formalized by the notion of *backdoors*, which are subsets of variables whose "removal" makes the formula tractable (for instance, one of the Schaefer classes). Much work has been done with backdoors as parameters for determining satisfiability, and we refer the reader to [9].

Above or Below Guarantee Parameterizations

Consider the standard parameterization of VERTEX COVER: given a graph $G = (V, E)$ on n vertices, decide whether G has a vertex cover of size at most k. The best-known algorithm for vertex cover is due to Chen et al. [1] and runs in time $O(1.2852^k + kn)$. Observe that if G has a matching of size μ, then any vertex cover also has size at least μ. In particular, if G has a perfect matching, then all vertex covers of G have $\Omega(n)$ edges, and even the FPT algorithms for the standard parameter will be obliged to spend time that is exponential in n.

Mahajan and Raman [13] consider the following alternative parameterization: does G have a vertex cover of size at most $\mu + k$? Note that the parameter k here is the size of the vertex cover *above* the matching size. Since the matching size is a guaranteed lower bound on the vertex cover size, this problem is referred to as the ABOVE GUARANTEE VERTEX COVER problem. Just as one can parameterize above guaranteed values, one can consider parameterizations below guaranteed values. A classic example is the following variant of VERTEX COVER: given a planar graph $G = (V, E)$ on n vertices and an integer parameter k, does G admit a vertex cover of size $\lfloor 3n/4 \rfloor - k$?

Key Results

One of the earliest attempts at parameterizing by the size of the vertex cover was made in [4]. Various graph layout problems were considered, where it turned out that a small vertex cover led to a very convenient structure for formulating a linear program. For many of these problems, it is not known if they are FPT parameterized by treewidth, and some are hard even on graphs of bounded treewidth (indeed, BANDWIDTH is NP-hard even on trees). This justifies the need for a stronger structural parameter, and in these examples, vertex cover turned out to be a very fruitful parameterization.

These examples led to a broader theme, namely, the "complexity ecology of parameters"

program, which was proposed in [5]. The theoretical foundations of this program were further established and surveyed in [6]. An immediate concern is that of how one formalizes the structural parameterization in question. Along the lines of [6], we distinguish the following possible objectives from the formalization:

1. The complexity of verifying and then exploiting a bounded parameter value
2. The complexity of exploiting structure that is guaranteed, but not given explicitly (a "promise" problem)
3. The complexity of exploiting structure that is explicitly provided along with the input, as a "witness"

The first setting is the most general but also the most computationally restrictive, as it puts the burden of discovering the structure also on the algorithm. This definition makes the study of parameters like bandwidth and cliquewidth prohibitive, as these are hard to determine even in the parameterized framework. On the other hand, while the other two notions are increasingly relaxed, the premise of a promise or the availability of witnesses in real-world witnesses remains a concern. We point the reader to [6] for a detailed discussion of the precise formalisms and their respective merits and trade-offs.

One of the major theoretical themes with alternate parameterizations is the exercise of identifying *meta-theorems* that explain the influence of the parameter over a large class of problems, usually specified in an appropriate logic. A cornerstone result of this kind is Courcelle's theorem, establishing that a problem expressible in Monadic Second Order Logic is FPT when parameterized by treewidth and the size of the formula [2]. Several generalizations of Courcelle's theorem have since been proposed, and many of them are surveyed in [10]. More recent work also establishes a similar result in the context of kernelization and parameters like vertex cover [8].

There is a rich literature that evidences the growing consideration of alternate parameterizations for optimization problems in varied contexts. As a concluding example, we turn our attention to [14], which serves to illustrate the scale at which it is possible to execute an exercise in understanding a question from several perspectives. Given two graphs H and G, the SUBGRAPH ISOMORPHISM problem asks if H is isomorphic to a subgraph of G. In [14], a framework is developed involving ten relevant parameters for each of H and G (such as treewidth, maximum degree, number of components, and so on). The generic question addressed in this work is if the problem admits an algorithm with running time:

$$f_1(p_1, p_2, \ldots, p_\ell) \cdot n^{f_2(p_{\ell+1}, \ldots, p_k)},$$

where each of $p_1, \ldots, p_k$ is one of the ten parameters depending only on H or G. We refer the reader to Figure 1 in [14] for a concise tabulation of the results. Notably, *all* combinations of questions (the number of which runs into the billions) are answered by a set of 28 of positive and negative results.

There are many examples of problems that are parameterized away from guaranteed bounds. We note that parameterizing vertex cover above the LP optimum has attracted considerable interest because a number of fundamental problems including Above Guarantee Vertex Cover, Odd Cycle Transversal, Split Vertex Deletion, and Almost 2-SAT reduce to this problem. Indeed, for many of these problems, the fastest algorithms at the time of this writing are obtained by reducing these problems to vertex cover parameterized above the LP optimum [12].

Open Problems

We direct the reader to the excellent survey [6] for several open problems concerning specific combinations of parameters for particular problems. In an applied context, an interesting possibility is to investigate if parameters can be learned from large samples of data.

Cross-References

- Kernelization, Constraint Satisfaction Problems Parameterized above Average
- Kernelization, Max-Cut Above Tight Bounds
- Kernelization, MaxLin Above Average
- Kernelization, Permutation CSPs Parameterized above Average
- LP Based Parameterized Algorithms
- Parameterization in Computational Social Choice
- Parameterized SAT
- Treewidth of Graphs

Recommended Reading

1. Chen J, Kanj IA, Jia W (2001) Vertex cover: further observations and further improvements. J Algorithms 41(2):280–301
2. Courcelle B (1990) The monadic second-order logic of graphs I: recognizable sets of finite graphs. Inf Comput 85:12–75
3. Cygan M, Fomin FV, Kowalik L, Lokshtanov D, Marx D, Pilipczuk M, Pilipczuk M, Saurabh S (2015) Parameterized algorithms. Springer, Cham. http://www.springer.com/us/book/9783319212746
4. Fellows MR, Lokshtanov D, Misra N, Rosamond FA, Saurabh S (2008) Graph layout problems parameterized by vertex cover. In: 19th international symposium on algorithms and computation (ISAAC). Lecture notes in computer science, vol 5369. Springer, Berlin, pp 294–305
5. Fellows M, Lokshtanov D, Misra N, Mnich M, Rosamond F, Saurabh S (2009) The complexity ecology of parameters: an illustration using bounded max leaf number. ACM Trans Comput Syst 45:822–848
6. Fellows MR, Jansen BMP, Rosamond FA (2013) Towards fully multivariate algorithmics: parameter ecology and the deconstruction of computational complexity. Eur J Comb 34(3):541–566
7. Ganian R, Hlinený P, Kneis J, Meister D, Obdrzálek J, Rossmanith P, Sikdar S (2010) Are there any good digraph width measures? In: IPEC, vol 6478. Springer, Berlin, pp 135–146
8. Ganian R, Slivovsky F, Szeider S (2013) Meta-kernelization with structural parameters. In: 38th international symposium on mathematical foundations of computer science, MFCS 2013. Lecture notes in computer science, vol 8087. Springer, Heidelberg, pp 457–468
9. Gaspers S, Szeider S (2012) Backdoors to satisfaction. In: Bodlaender HL, Downey R, Fomin FV, Marx D (eds) The multivariate algorithmic revolution and beyond. Lecture notes in computer science, vol 7370. Springer, Berlin/Heidelberg, pp 287–317
10. Grohe M, Kreutzer S (2011) Methods for algorithmic meta theorems. In: Grohe M, Makowsky J (eds) Model theoretic methods in finite combinatorics. Contemporary mathematics, vol 558. American Mathematical Society, Providence, pp 181–206
11. Henglein F, Mairson HG (1991) The complexity of type inference for higher-order typed lambda calculi. J Funct Program 4:119–130
12. Lokshtanov D, Narayanaswamy NS, Raman V, Ramanujan MS, Saurabh S (2014) Faster parameterized algorithms using linear programming. ACM Trans Algorithms 11(2):15:1–15:31
13. Mahajan M, Raman V (1999) Parameterizing above guaranteed values: maxsat and maxcut. J Algorithms 31(2):335–354
14. Marx D, Pilipczuk M (2014) Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In: 31st international symposium on theoretical aspects of computer science (STACS), Lyon, pp 542–553

Alternative Performance Measures in Online Algorithms

Alejandro López-Ortiz
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Keywords

Bijective analysis; Diffuse adversary; Loose competitiveness; Relative interval analysis; Relative worst-order ratio; Smoothed analysis

Years and Authors of Summarized Original Work

2000; Koutsoupias, Papadimitriou
2005; Dorrigiv, López-Ortiz

Problem Definition

While the competitive ratio [19] is the most common metric in online algorithm analysis and it has led to a vast amount of knowledge in the field, there are numerous known applications in

which the competitive ratio produces unsatisfactory results. Far too often, it leads to unrealistically pessimistic measures including the failure to distinguish between algorithms that have vastly differing performance under any practical characterization in practice. Because of this there, has been extensive research in alternatives to the competitive ratio, with a renewed effort in the period from 2005 to the present date.

The competitive ratio metric can be derived from the observation that an online algorithm, in essence, computes a partial solution to a problem using incomplete information. Then, it is only natural to quantify the performance drop due to this absence of information. That is, we compare the quality of the solution obtained by the online algorithm with the one computed in the presence of full information, namely, that of the offline optimal OPT, in the *worst case*. More formally,

Definition 1 An online algorithm $\mathcal{A}$ is said to have (asymptotic) competitive ratio c if $\mathcal{A}(\sigma) \leq c \cdot \text{OPT}(\sigma) + b$ for all input sequences σ and fixed constants b and c.

The early literature considered only algorithms with constant competitive ratio, and all others are termed as algorithms with *unbounded* competitive ratio. However, it is easy to extend this definition to a $C(n)$-competitive algorithm as follows:

Definition 2 An online algorithm $\mathcal{A}$ is said to have (asymptotic) competitive ratio $C(n)$ if $\mathcal{A}(\sigma) \leq C(n) \cdot \text{OPT}(\sigma) + b$ for all σ and a fixed constant b. When $b = 0$, $C(n)$ is termed the *absolute* competitive ratio.

A natural expectation would be that the performance of OPT reflects both knowledge of the future and the inherent structure of the specific instance being solved, and hence, an online algorithm with optimal competitive ratio must handle most if not all instances in an efficient manner. Unfortunately, for most problems, the worst-case nature of the competitive ratio leads to algorithms of varying degrees of sophistication having the same equally bad competitive ratio. As a consequence the competitive ratio leads to "equivalence" for online algorithms with vastly differing performance in practice.

In the next sections we discuss the main alternatives to and refinements of the competitive ratio and highlight their relative benefits and drawbacks.

Key Results

Relative Worst-Order Ratio

The relative worst-order ratio [8, 10, 11] combines some desirable properties of two earlier measures, namely, the max/max ratio [6] and the random order ratio [15]. Using this measure we can directly compare two online algorithms. Informally, for a given sequence, it considers the worst-case ordering of that sequence for each algorithm and compares their behavior as a ratio on these orderings. Then it finds among all sequences (not just reorderings) the one that maximizes the ratio above in the worst-case performance.

Let $\mathcal{A}$ and $\mathcal{B}$ be online algorithms for an online minimization problem and let $\mathcal{A}(I)$ be the cost of $\mathcal{A}$ on an input sequence $I = (i_1, i_2, \ldots, i_n)$. Denote by I_σ the sequence obtained by applying a permutation σ to I, i.e., $I_\sigma = (i_{\sigma_1}, \ldots, i_{\sigma_n})$. Define $\mathcal{A}_W(I) = \min_\sigma \mathcal{A}(I_\sigma)$.

Definition 3 ([11]) Let $S_1(c)$ and $S_2(c)$ be the statements about algorithms $\mathcal{A}$ and $\mathcal{B}$ defined in the following way:

$S_1(c)$: There exists a constant b such that $\mathcal{A}_W(I) \leq c \cdot \mathcal{B}_W(I) + b$ for all I.

$S_2(c)$: There exists a constant b such that $\mathcal{A}_W(I) \geq c \cdot \mathcal{B}_W(I) - b$ for all I.

The relative worst-order ratio $WR_{\mathcal{A},\mathcal{B}}$ of an online algorithm $\mathcal{A}$ to algorithm $\mathcal{B}$ is defined if $S_1(1)$ or $S_2(1)$ holds. In this case $\mathcal{A}$ and $\mathcal{B}$ are said to be comparable. If $S_1(1)$ holds, then $WR_{\mathcal{A},\mathcal{B}} = \sup\{r | S_2(r)\}$, and if $S_2(r)$ holds, then $WR_{\mathcal{A},\mathcal{B}} = \inf\{r | S_1(r)\}$.

$WR_{\mathcal{A},\mathcal{B}}$ can be used to compare the qualities of $\mathcal{A}$ and $\mathcal{B}$. If $WR_{\mathcal{A},\mathcal{B}} = 1$, then these two

algorithms have the same quality with respect to this measure. The magnitude of difference between $WR_{\mathcal{A},\mathcal{B}}$ and 1 reflects the difference between the behavior of the two algorithms. For a minimization problem, $\mathcal{A}$ is better than $\mathcal{B}$ with respect to this measure if $WR_{\mathcal{A},\mathcal{B}} < 1$ and vice versa. Boyar and Favrholdt showed that the relative worst-order ratio is transitive [8].

Note that we can also compare the online algorithm $\mathcal{A}$ to an optimal offline algorithm OPT. The *worst-order ratio* of $\mathcal{A}$ is defined as $WR_{\mathcal{A}} = WR_{\mathcal{A},\mathrm{OPT}}$. For some problems, OPT is the same for all order of requests on a given input sequence, and hence, the worst-order ratio is the same as the competitive ratio. However, for other problems such as paging the order does matter for OPT.

In [10], three online algorithms (FIRST-FIT, BEST-FIT, and WORST-FIT) for two variants of the seat reservation problem [9] are compared using the relative worst-order ratio. The relative worst-order ratio when applied to paging algorithms can be used to differentiate LRU which is strictly better than FWF with respect to the worst-order ratio, while they have the same competitive ratio [11]. Similarly, [11] proposes a new paging algorithm, retrospective LRU (RLRU), and shows that it is better than LRU under this measure while not under the competitive ratio.

Loose Competitiveness

Loose competitiveness was first proposed in [22] and later modified in [25]. It attempts to obtain a more realistic measure by observing that first, in many real online problems, we can ignore those input sequences on which the online algorithm incurs a cost less than a certain threshold and, second, many online problems have a second resource parameter (e.g., size of cache, number of servers) and the input sequences are independent of these parameters. In contrast, in competitive analysis, the adversary can select sequences tailored against those parameters. For example, for caching the worst-case input with competitive ratio k can only be constructed by the adversary if it is aware of the size k of the cache. However, in practice the competitive ratios of many online paging algorithms have been observed to be constant [25], i.e., independent of k.

In loose competitiveness we consider an adversary that is oblivious to the parameter by requiring it to give a sequence that is bad for most values of the parameter rather than just a specific bad value of the parameter. Let $\mathcal{A}_k(I)$ denote the cost of an algorithm $\mathcal{A}$ on an input sequence I, when the parameter of the problem is k.

Definition 4 ([25]) An algorithm $\mathcal{A}$ is (ϵ,δ)-loosely c-competitive if, for any input sequence I and for any n, at least $(1-\delta)n$ of the values $k \in \{1,2,\ldots,n\}$ satisfy $\mathcal{A}_k(I) \le \max\{c \cdot \mathrm{OPT}_k(I), \epsilon\,|I|\}$.

Therefore, we ignore the input sequences I which cost less than $\epsilon\,|I|$. Also we require the algorithm to be good for at least $(1-\delta)$ fraction of the possible parameters. For each online problem, we can select the appropriate constants ϵ and δ. The following result shows that by this modification of the competitive analysis, we can obtain paging algorithms with constant performance ratios.

Theorem 1 ([25]) *Every k-competitive paging algorithm is (ϵ,δ)-loosely c-competitive for any $0 < \epsilon, \delta < 1$, and $c = (e/\delta)\ln(e/\epsilon)$, where e is the base of the natural logarithm.*

Diffuse Adversary Model

The diffuse adversary model [16] tries to refine the competitive ratio by restricting the set of legal input sequences. In the diffuse adversary model, the input is generated according to a distribution belonging to a member of a class Δ of distributions.

Definition 5 Let $\mathcal{A}$ be an online algorithm for a minimization problem and let Δ be a class of distributions for the input sequences. Then $\mathcal{A}$ is c-competitive against Δ, if there exists a constant b, such that $\mathcal{E}_{I\in D}\mathcal{A}(I) \le c\cdot\mathcal{E}_{I\in D}\mathrm{OPT}(I)+b$, for every distribution $D \in \Delta$, where $\mathcal{A}(I)$ denotes the cost of $\mathcal{A}$ on the input sequence I and the expectations are taken over sequences that are picked according to D.

In other words, for a given algorithm A, the adversary selects the distribution D in Δ that

leads to its worst-case performance in that family. If Δ is highly restrictive, then A knows more about the distribution of input sequences and the power of adversary is more constrained. When Δ contains all possible distributions, then the competitive analysis against Δ is the same as the standard competitive ratio.

Computing the actual competitive ratio of both deterministic and randomized paging algorithms against Δ_ϵ is studied in [23, 24]. An estimation of the optimal competitive ratio for several algorithms (such as LRU and FIFO) within a factor of 2 is given. Also it is observed that around the threshold $\epsilon \approx 1/k$, the best competitive ratios against Δ_ϵ are $\Theta(\ln k)$. The competitive ratios rapidly become constant for values of ϵ less than the threshold. For $\epsilon = \omega(1/k)$, i.e., values greater than the threshold, the competitive ratio rapidly tends to $\Theta(k)$ for deterministic algorithms while it remains unchanged for randomized algorithms.

Note that we can also model locality of reference using the diffuse adversary model by considering only those distributions that are consistent with distributions obeying a locality of reference principle. In particular Dorrigiv et al. showed that for the list update problem MTF is optimal in expected cost under any probability distribution that has locality of reference monotonicity, i.e., a recently accessed item has equal or larger probability of being accessed than a less recently accessed item [14].

Bijective Analysis

Bijective analysis and average analysis [3] build upon the framework of locality of reference by [1]. These models directly compare two online algorithms without appealing to the concept of the offline "optimal" cost. In addition, these measures do not evaluate the performance of the algorithm on a single "worst-case" request, but instead use the cost that the algorithm incurs on each and all request sequences. Informally, bijective analysis aims to pair input sequences for two algorithms $\mathcal{A}$ and $\mathcal{B}$ using a bijection in such a way that the cost of $\mathcal{A}$ on input σ is no more than the cost of $\mathcal{B}$ on the image of σ, for all request sequences σ of the same length. In this case, intuitively, $\mathcal{A}$ is no worse than $\mathcal{B}$. On the other hand, average analysis compares the average cost of the two algorithms over all request sequences of the same length. For an online algorithm $\mathcal{A}$ and an input sequence σ, let $\mathcal{A}(\sigma)$ be the cost incurred by $\mathcal{A}$ on σ. Denote by I_n the set of all input sequences of length n.

We say that an online algorithm $\mathcal{A}$ is *no worse* than an online algorithm $\mathcal{B}$ according to bijective analysis if there exists an integer $n_0 \geq 1$ so that for each $n \geq n_0$, there is a bijection $b : \mathcal{I}_n \leftrightarrow \mathcal{I}_n$ satisfying $\mathcal{A}(\sigma) \leq \mathcal{B}(b(\sigma))$ for each $\sigma \in \mathcal{I}_n$. We denote this by $\mathcal{A} \preceq_b \mathcal{B}$.

We say that an online algorithm $\mathcal{A}$ is *no worse* than an online algorithm $\mathcal{B}$ according to average analysis if there exists an integer $n_0 \geq 1$ so that for each $n \geq n_0$, $\sum_{I \in \mathcal{I}_n} \mathcal{A}(I) \leq \sum_{I \in \mathcal{I}_n} \mathcal{B}(I)$. We denote this by $\mathcal{A} \preceq_a \mathcal{B}$.

Under both bijective analysis and average analysis alone, all *lazy* algorithms (including LRU and FIFO, but not FWF) are in fact strongly equivalent. This is evidence of an inherent difficulty to separate these algorithms in any general unrestricted setting. Their superiority is seemingly derived from the well-known observation that input sequences for paging and several other problems show locality of reference [12, 13]. This means that when a page is requested, it is more likely to be requested in the near future. Therefore, several models for paging with locality of reference have been proposed.

Hence, the need to combine bijective analysis with an assumption of locality of reference model such as concave analysis. In this model a request sequence has high locality of reference if the number of distinct pages in a window of size n is small.

Using this measure Angelopoulos et al. [3] show that LRU is never outperformed in any possible subpartition on the request sequence space induced by concave analysis, while it always outperforms *any other paging algorithm* in at least one subpartition of the sequence space. This result proves separation between LRU and all other algorithms and provides theoretical backing to the observation that LRU is preferable in practice. This is the first deterministic theoretical

model to provide full separation between LRU and all other algorithms. Recently this result was strengthened by Angelopolous and Schweitzer [2] where they showed that the separation also holds under the stricter bijective analysis (as opposed to average analysis) using the concave analysis framework.

Smoothed Competitiveness

Some algorithms that have very bad *worst-case* performance behave very well in practice. One of the most famous examples is the simplex method. This algorithm has a very good performance in practice but it has exponential worst-case running time. *Average case* analysis of algorithms can somehow explain this behavior, but sometimes there is no basis to the assumption that the inputs to an algorithm are random.

Smoothed analysis of algorithms [21] tries to explain this intriguing behavior without assuming anything about the distribution of the input instances. In this model, we randomly perturb (smoothen) the input instances according to a probability distribution f and then analyze the behavior of the algorithm on these perturbed (smoothed) instances. For each input instance $\check{I}$, we compute the neighborhood $N(\check{I})$ of $\check{I}$ which contains the set of all perturbed instances that can be obtained from $\check{I}$. Then we compute the expected running time of the algorithm over all perturbed instances in this neighborhood. The smoothed complexity of the algorithm is the maximum of this expected running time over all the input instances. Intuitively, an algorithm with a bad worst-case performance can have a good smoothed performance if its worst-case instances are isolated. Spielman and Teng show [21] that the simplex algorithm has polynomial smoothed complexity. Several other results are known about the smoothed complexity of the algorithms [4, 7, 18, 20].

Becchetti et al. [5] introduced *smoothed competitive analysis* which mirrors competitive analysis except that we consider the cost of the algorithm on randomly perturbed adversarial sequences. As in the analysis of the randomized online algorithms, we can have either an *oblivious adversary* or an *adaptive adversary*. The smoothed competitive ratio of an online algorithm $\mathcal{A}$ for a minimization problem can be formally defined as follows.

Definition 6 ([5]) The *smoothed competitive ratio* of an algorithm $\mathcal{A}$ is defined as

$$c = \sup_{\check{I}} \mathcal{E}_{I \leftarrow N(\check{I})} \left[\frac{\mathcal{A}(I)}{OPT(I)} \right],$$

where the supremum is taken over all input instances $\check{I}$ and the expectation is taken over all instances I that are obtainable by smoothening the input instance $\check{I}$ according to f in the neighborhood $N(\check{I})$.

In [5], they use the smoothed competitive ratio to analyze the MULTI-LEVEL FEEDBACK(MLF) algorithm for processor scheduling in a time-sharing multitasking operating system. This algorithm has very good practical performance, but its competitive ratio is very bad and obtains strictly better ratios using the smooth competitive analysis than with the competitive ratio.

Search Ratio

The search ratio belongs to the family of measures in which the offline OPT is weakened. It is defined only for the specific case of geometric searches in an unknown terrain for a target of unknown position. Recall that the competitive ratio compares against an all-knowing OPT; indeed, for geometric searches in the competitive ratio framework, the OPT is simply a shortest path algorithm, while the online search algorithm has intricate methods for searching. The search ratio instead considers the case where OPT knows the terrain but not the position of the target. That is, the search ratio compares two search algorithms, albeit one more powerful than the other. By comparing two instances of like objects, the search ratio can be argued to be a more meaningful measure of the quality of an online search algorithm. Koutsopias et al. show that searching in trees results the same large competitive ratio regardless of the search strategy, yet under the search ratio framework, certain algorithms are far superior to others [17].

Cross-References

▸ Online Paging and Caching

Recommended Reading

1. Albers S, Favrholdt LM, Giel O (2005) On paging with locality of reference. JCSS 70(2): 145–175
2. Angelopoulos S, Schweitzer P (2009) Paging and list update under bijective analysis. In: Proceedings of the twentieth annual ACM-SIAM symposium on discrete algorithms (SODA 2009), New York, 4–6 Jan 2009, pp 1136–1145
3. Angelopoulos S, Dorrigiv R, López-Ortiz A (2007) On the separation and equivalence of paging strategies. In: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms (SODA 2007), New Orleans, 7–9 Jan 2007, pp 229–237
4. Banderier C, Mehlhorn K, Beier R (2003) Smoothed analysis of three combinatorial problems. In: Proceedings of the 28th international symposium on mathematical foundations of computer science 2003 (MFCS 2003), Bratislava, 25–29 Aug 2003, vol 2747, pp 198–207
5. Becchetti L, Leonardi S, Marchetti-Spaccamela A, Schafer G, Vredeveld T (2003) Average case and smoothed competitive analysis of the multi-level feedback algorithm. In: IEEE (ed) Proceedings of the 44th symposium on foundations of computer science (FOCS 2003), Cambridge, 11–14 Oct 2003, pp 462–471
6. Ben-David S, Borodin A (1994) A new measure for the study of on-line algorithms. Algorithmica 11:73–91
7. Blum A, Dunagan J (2002) Smoothed analysis of the perceptron algorithm for linear programming. In: Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms, San Francisco, 6–8 Jan 2002, pp 905–914
8. Boyar J, Favrholdt LM (2003) The relative worst order ratio for on-line algorithms. In: Proceedings of the 5th Italian conference on algorithms and complexity (CIAC 2003), Rome, 28–30 May 2003,
9. Boyar J, Larsen KS (1999) The seat reservation problem. Algorithmica 25(4):403–417
10. Boyar J, Medvedev P (2004) The relative worst order ratio applied to seat reservation. In: Proceedings of the 9th Scandinavian workshop on algorithm theory (SWAT 2004), Humlebaek, 8–10 July 2004
11. Boyar J, Favrholdt LM, Larsen KS (2005) The relative worst order ratio applied to paging. In: Proceedings of the sixteenth annual ACM-SIAM symposium on discrete algorithms (SODA 2005), Vancouver, 23–25 Jan 2005. ACM, pp 718–727
12. Denning PJ (1968) The working set model for program behaviour. CACM 11(5):323–333
13. Denning PJ (1980) Working sets past and present. IEEE Trans Softw Eng SE-6(1):64–84
14. Dorrigiv R, López-Ortiz A (2012) List update with probabilistic locality of reference. Inf Process Lett 112(13):540–543
15. Kenyon C (1996) Best-fit bin-packing with random order. In: Proceedings of the seventh annual ACM-SIAM symposium on discrete algorithms, Atlanta, 28–30 Jan 1996, pp 359–364
16. Koutsoupias E, Papadimitriou C (2000) Beyond competitive analysis. SIAM J Comput 30: 300–317
17. Koutsoupias E, Papadimitriou C, Yannakakis M (1996) Searching a fixed graph. In: Proceedings of the 23rd international colloquium on automata, languages and programming (ICALP96), Paderborn, 8–12 July 1996, vol 1099, pp 280–289
18. Manthey B, Reischuk R (2005) Smoothed analysis of the height of binary search trees. Schriftenreihe der Institute für Informatik und Mathematik A-05-17, Universität zu Lübeck, Lübeck. http://www.tcs.uni-luebeck.de/pages/manthey/publications/SearchTrees-SIIM-A-05-17.pdf
19. Sleator DD, Tarjan RE (1985) Amortized efficiency of list update and paging rules. Commun ACM 28:202–208
20. Spielman DA, Teng SH (2003) Smoothed analysis of termination of linear programming algorithms. Math Program 97(1–2):375–404
21. Spielman DA, Teng SH (2004) Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. J ACM 51(3):385–463
22. Young NE (1994) The k-server dual and loose competitiveness for paging. Algorithmica 11(6):525–541
23. Young NE (1998) Bounding the diffuse adversary. In: Proceedings of the ninth annual ACM-SIAM symposium on discrete algorithms, San Francisco, 25–27 Jan 1998, pp 420–425
24. Young NE (2000) On-line paging against adversarially biased random inputs. J Algorithms 37(1):218–235
25. Young NE (2002) On-line file caching. Algorithmica 33(3):371–383

Amortized Analysis on Enumeration Algorithms

Takeaki Uno
National Institute of Informatics, Chiyoda, Tokyo, Japan

Keywords

Amortized analysis; Delay; Enumeration tree; Recursion

Years and Authors of Summarized Original Work

1998; Uno

Problem Definition

Let $\mathcal{A}$ be an enumeration algorithm. Suppose that $\mathcal{A}$ is a recursive type algorithm, i.e., composed of a subroutine that recursively calls itself several times (or none). Thus, the recursion structure of the algorithm forms a tree. We call the subroutine or the execution of the subroutine an *iteration*. We here assume that an iteration does not include the computation done in the recursive calls generated by itself. We regard a series of subroutines of different types as an iteration if they form a nested recursion. We simply write the set of all iterations of an execution of $\mathcal{A}$ by $\mathcal{X}$.

When an iteration X recursively calls an iteration Y, X is called the *parent* of Y, and Y is called a *child* of X. The *root iteration* is that with no parent. For non-root iteration X, its parent is unique and is denoted by $P(X)$. The set of the children of X is denoted by $C(X)$. The parent-child relation between iterations forms a tree structure called a *recursion tree*, or an *enumeration tree*. An iteration is called a *leaf iteration* if it has no child and an *inner iteration* otherwise.

For iteration X, an upper bound of the execution time (the number of operations) of X is denoted by $T(X)$. Here we exclude the computation for the output process from the computation time. We remind that $T(X)$ is the time for local execution time and thus does not include the computation time in the recursive calls generated by X. For example, when $T(X) = O(n^2)$, $T(X)$ is written as cn^2 for some constant c. T^* is the maximum $T(X)$ among all leaf iterations X. Here, T^* can be either constant or a polynomial of the input size. If X is an inner iteration, let $\overline{T}(X) = \sum_{Y \in C(X)} T(Y)$.

Key Results

We explain methods to amortize the computation time of iterations that only requires a local condition and give simple algorithms which achieves nontrivial time complexity. On enumeration algorithms, it is very hard to grasp the global structures of the computation and the recursion tree that is coming from the hardness of estimating the number of iterations in a branch. Instead of that, we approach from local amortization from parent and children. When we go deep in a recursion tree, the number of iterations tends to increase exponentially, and the size of the input of each iteration often decreases on the other hand. Motivated by this observation, we amortize the computation time by moving the computation time of each iteration to its children from the top to bottom, so that the long computation time on upper levels is diffused.

Amortization by Children

Suppose that each iteration X takes $O((|C(X)|+1)T)$ time. Note that this implies that a leaf iteration takes $O(T)$ time. Then, the total computation time of the algorithm is $O(T \sum_{X \in \mathcal{X}} |C(X)| + 1) = O(T(|\mathcal{X}| + \sum_{X \in \mathcal{X}} |C(X)|) = O(T|\mathcal{X}|)$, since any iteration is a child of at most one iteration. Hence, an iteration takes $O(T)$ time on average. Let us see an example on the following algorithm for enumerating all subsets of $\{1, \ldots, n\}$.

We can confirm that the algorithm correctly enumerates all subsets without duplications, and an iteration X takes $O(|C(X)|)$ time, except for the output process. Without amortization, the time complexity is $O(n)$ for each iteration, but the above amortization reduces it to $O(1)$. Note that the output process is shortened by outputting each subset by the difference from the previously output subset, and by this the accumulated computation time for output process is also bounded by $O(1)$ for each subset. This amortization technique is common in many algorithms. Further, in the enumeration of spanning trees, the time

Algorithm EnumSubset (S, x)**:**

```
1 output S
2 for i := x + 1 to n; call EnumSubset
  (S ∪ {i}, i + 1)
```

complexity is amortized by not only the children but also the grandchildren [3]. More sophisticated amortization is used in [1, 2] for path connecting given two vertices and subtrees of size k.

Push-Out Amortization

When the computation time of an iteration X is not proportional $|C(X)|$, the above amortization does not work. In such cases, *push-out* amortization [4–6] can work. We amortize the computation time by charging the computation time of iterations near by the root of the recursion to those in bottom levels, by recursively moving the computation time from an iteration to its children from top to down. The move is done in the following push-out rule.

Push-out rule (PO rule): Suppose that iteration X receives a computation time of $S(X)$ from its parent; thus X has computation time of $S(X) + T(X)$ in total. Then, we fix $\frac{\beta}{\alpha-1}(|C(X)|+1)T^*$ of the computation time to X and charge (push-out) the remaining computation time of quantity $S(X) + T(X) - \frac{\beta}{\alpha-1}(|C(X)| + 1)T^*$ to its children. Each child Z of X receives computation time proportional to $T(Z)$, that is, $S(Z) = (S(X) + T(X) - \frac{\beta}{\alpha-1}(|C(X)| + 1)T^*)\frac{T(Z)}{\overline{T}(X)}$.

After the moves in this rule from the top to bottom of the recursion tree, each inner iteration has $O((|C(X)| + 1)T^*)$ computation time, thus $O(T^*)$ time per iteration. Moreover, when the following push-out condition holds for any non-leaf iteration X, each leaf iteration receives computation time of $O(T^*)$ from its parent; thus the computation time per iteration is bounded by $O(T^*)$. Suppose that $\alpha > 1$ and $\beta \geq 0$ are two constants.

Push-Out Condition (PO Condition)
$\overline{T}(X) \geq \alpha T(X) - \beta(|C(X)| + 1)T^*$

Intuitively, this means that $\overline{T}(X) \geq \alpha T(X)$ holds after the assignment of the computation time of $\alpha\beta(|C(X)| + 1)T^*$ to children and the remaining to itself, the inequation. Thus, the computation time of one level of recursion intuitively increases as the depth, unless there are not so many leaf iterations. These suggest that the total computation time spent by middle-level iterations is relatively short compared to that by leaf iterations.

Theorem 1 *If any inner iteration of an enumeration algorithm satisfies PO condition, the amortized computation time of an iteration is* $O(T^*)$.

Proof We state by induction that when we charge computation time with PO rule, from the root iteration to the leaf iterations, each iteration X satisfies $S(X) \leq T(X)/(\alpha - 1)$. The root iteration satisfies this condition. Suppose that an iteration X satisfies it. Then, for any child Z of X, we have

$$\begin{aligned} S(Z) &= (S(X) + T(X) - \frac{\beta}{\alpha-1}(|C(X)| + 1)T^*)\frac{T(Z)}{\overline{T}(X)} \\ &\leq (T(X)/(\alpha-1) + T(X) - \frac{\beta}{\alpha-1}(|C(X)| + 1)T^*)\frac{T(Z)}{\overline{T}(X)} \\ &= \frac{\alpha T(X) - \beta(|C(X)| + 1)T^*}{\overline{T}(X)} \times \frac{T(Z)}{\alpha-1}. \end{aligned}$$

Therefore, any leaf iteration receives $O(T^*)$ time from its parent, and the statement holds.

Since PO condition is satisfied, $\overline{T}(X) \geq \alpha T(X) - \beta(|C(X)| + 1)T^*$. Thus,

$$\frac{\alpha T(X) - \beta(|C(X)| + 1)T^*}{\overline{T}(X)}\frac{T(Z)}{\alpha-1} \leq \frac{T(Z)}{\alpha-1}.\square$$

Matching Enumeration

Let us see an example of designing algorithms so that push-out amortization does work. The problem is the enumeration of matchings in an undirected graph $G = (V, E)$. A *matching* is an edge set $M \subseteq E$ such that any vertex is incident to at most one edge in M. A straightforward

way to enumerate all matchings is to choose an edge e and enumerate matchings including e and enumerate matchings not including e, recursively. This algorithm yields the time complexity of $O(|V|)$ for each matching.

We here consider another way for the enumeration. We choose a vertex v of the maximum degree and partition the problem into enumeration of matchings including e_1, matchings including e_2, ..., matchings including e_k, and matchings including none of $e_1, \ldots, e_k$. Here $e_1, \ldots, e_k$ are the edges incident to v. Since any matching has at most one edge incident to v, this algorithm is complete and makes no duplication. The algorithm is described as follows. Note that $G \setminus \{v\}$ denotes the graph obtained by removing vertex v and edges incident to v from G.

Algorithm EnumMatching $(G = (V, E), M)$**:**

1 **if** $E = \emptyset$ **then output** M; **return**
2 choose a vertex v having the maximum degree in G
3 **call** EnumMatching $(G \setminus \{v\}, M)$
4 **for** each edge $e = (v, u)$, **call** EnumMatching $(G \setminus \{u, v\}, M \cup \{e\})$

$G \setminus \{u, v\}$ is obtained from $G \setminus \{u', v\}$ in $O(d(u) + d(u'))$ time, where $d(u)$ and $d(u')$ are the degrees of u and u', respectively. From this, the computation time in step 4 is bounded by the sum of degrees of all vertices adjacent to v. Here $T(X) = c|E|$ for some c, except for the output process. Note that $|E|$ is the number of edges in the graph given to X.

The input graph of the child generated in step 3 has $|E| - d(v)$ edges and that in step 4 has $|E| - d(v) - d(u) + 1$ edges. Thus, when $d(v) < |E|/4$, we have $\overline{T}(X) = c((|E| - d(v)) + (|E| - d(v) - d(u) + 1)) \geq 1.25c|E|$. When $d(v) \geq |E|/4$, $|C(X)| \geq |E|/4$. Thus, PO condition holds by setting $\alpha = 1.25$ and choosing a certain β. The output process can be shorten as the subset enumeration.

Theorem 2 *Matchings of a graph can be enumerated in $O(1)$ time for each matching.*

Elimination Ordering

An *elimination ordering* is a sequence of elements obtained by iteratively removing an element from an object G with keeping a property satisfied, until the object will be empty. Examples are perfect elimination ordering and perfect sequence. The former is the removal sequence of simplicial vertices from a chordal graph, and the latter is the removal sequence of cliques from a connected chordal graph. Elimination orderings can be enumerated by a simple algorithm as follows.

Algorithm EnumElim (G, S)**:**

1 **if** $G = \emptyset$ **then output** S; **return**
2 **for** each element e of G that can be removed, **call** EnumElim $(G \setminus \{e\}, S \cup \{e\})$

Here we assume that $T(X) = \text{poly}(|G|)$ except for output process. The decision problem of removing an element from G is naturally considered to be solved in $O(\text{poly}(|G|))$ time; thus this assumption is natural.

Theorem 3 *If any G of size larger than some constant c has at least two removable elements, elimination orderings are enumerated in $O(1)$ time for each.*

Proof The statement means that each iteration has at least two children, if its computation time is not constant. For sufficiently large constant δ, we always have $\text{poly}(|G|) \leq 2\text{poly}(|G| - 1)$ for any $|G| > \delta$. This implies that PO condition always holds for these iterations. □

Recommended Reading

1. Birmele E, Ferreira RA, Grossi R, Marino A, Pisanti N, Rizzi R, Sacomoto G (2013) Optimal listing of cycles and st-paths in undirected graphs. SODA 2013:1884–1896
2. Ferreira RA, Grossi R, Rizzi R (2011) Output-sensitive listing of bounded-size trees in undirected graphs. ESA 2011:275–286
3. Shioura A, Tamura A, Uno T (1997) An optimal algorithm for scanning all spanning trees of undirected graphs. SIAM J Comput 26:678–692

4. Uno T (1998) New approach for speeding up enumeration algorithms. LNCS 1533:287–296
5. Uno T (1999) A new approach for speeding up enumeration algorithms and its application for matroid bases. LNCS 1627:349–359
6. Uno T (2014) A new approach to efficient enumeration by push-out amortization. arXiv:1407.3857

AMS Sketch

Graham Cormode
Department of Computer Science, University of Warwick, Coventry, UK

Keywords

Euclidean norm; Second-moment estimation; Sketch; Streaming algorithms

Years and Authors of Summarized Original Work

1996; Alon, Matias, Szegedy

Problem Definition

Streaming algorithms aim to summarize a large volume of data into a compact summary, by maintaining a data structure that can be incrementally modified as updates are observed. They allow the approximation of particular quantities. The AMS sketch is focused on approximating the sum of squared entries of a vector defined by a stream of updates. This quantity is naturally related to the Euclidean norm of the vector and so has many applications in high-dimensional geometry and in data mining and machine learning settings that use vector representations of data.

The data structure maintains a linear projection of the stream (modeled as a vector) with a number of randomly chosen vectors. These random vectors are defined implicitly by simple hash functions, and so do not have to be stored explicitly. Varying the size of the sketch changes the accuracy guarantees on the resulting estimation. The fact that the summary is a linear projection means that it can be updated flexibly, and sketches can be combined by addition or subtraction, yielding sketches corresponding to the addition and subtraction of the underlying vectors.

Key Results

The AMS sketch was first proposed by Alon, Matias, and Szegedy in 1996 [1]. Several refinements or variants have subsequently appeared in the literature, for example, in the work of Thorup and Zhang [4]. The version presented here works by using hashing to map each update to one of t counters rather than taking the average of t repetitions of an "atomic" sketch, as was originally proposed. This hash-based variation is often referred to as the "fast AMS" summary.

Data Structure Description

The AMS summary maintains an array of counts which are updated with each arriving item. It gives an estimate of the ℓ_2-norm of the vector v that is induced by the sequence of updates. The estimate is formed by computing the norm of each row and taking the median of all rows. Given parameters ε and δ, the summary uses space $O(1/\varepsilon^2 \log 1/\delta)$ and guarantees with probability of at least $1 - \delta$ that its estimate is within relative ε-error of the true ℓ_2-norm, $\|v\|_2$.

Initially, v is taken to be the zero vector. A stream of updates modifies v by specifying an index i to which an update w is applied, setting $v_i \leftarrow v_i + w$. The update weights w can be positive or negative.

The AMS summary is represented as a compact array C of $d \times t$ counters, arranged as d rows of length t. In each row j, a hash function h_j maps the input domain U uniformly to $\{1, 2, \ldots t\}$. A second hash function g_j maps elements from U uniformly onto $\{-1, +1\}$. For the analysis to hold, we require that g_j is *four-wise* independent. That is, over the random choice of g_j from the set of all possible hash functions, the probability that any four distinct items from

the domain that get mapped to $\{-1, +1\}^4$ is uniform: each of the 16 possible outcomes is equally likely. This can be achieved by using polynomial hash functions of the form $g_j(x) = 2((ax^3 + bx^2 + cx + d \mod p) \mod 2) - 1$, with parameters a, b, c, d chosen uniformly from the prime field p.

The sketch is initialized by picking the hash functions to be used and initializing the array of counters to all zeros. For each update operation to index i with weight w (which can be either positive or negative), the item is mapped to an entry in each row based on the hash functions h and the update applied to the corresponding counter, multiplied by the corresponding value of g. That is, for each $1 \leq j \leq d$, $h_j(i)$ is computed, and the quantity $wg_j(i)$ is added to entry $C[j, h_j(i)]$ in the sketch array. Processing each update therefore takes time $O(d)$, since each hash function evaluation takes constant time.

The sketch allows an estimate of $\|v\|_2^2$, the squared Euclidean norm of v, to be obtained. This is found by taking the sums of the squares of the rows of the sketch and in turn finding the median of these sums. That is, for row j, it computes $\sum_{k=1}^{t} C[j,k]^2$ as an estimate and takes the median of these d estimates. The query time is linear in the size of the sketch, $O(td)$, as is the time to initialize a new sketch. Meanwhile, update operations take time $O(d)$.

The analysis of the algorithm follows by considering the produced estimate as a random variable. The random variable can be shown to be correct in expectation: its expectation is the desired quantity, $\|v\|_2^2$. This can be seen by expanding the expression of the estimator. The resulting expression has terms $\sum_i v_i^2$ but also terms of the form $v_i v_j$ for $i \neq j$. However, these "unwanted terms" are multiplied by either $+1$ or -1 with equal probability, depending on the choice of the hash function g. Therefore, their expectation is zero, leaving only $\|v\|_2$. To show that it is likely to fall close to its expectation, we also analyze the variance of the estimator and use Chebyshev's inequality to argue that with constant probability, each estimate is close to the desired value. Then, taking the median of sufficient repetitions amplifies this constant probability to be close to certainty.

This analysis shows that the estimate is between $(1 - \varepsilon)\|v\|_2^2$ and $(1 + \varepsilon)\|v\|_2^2$. Taking the square root of the estimate gives a result that is between $(1 - \varepsilon)^{1/2}\|v\|_2$ and $(1 + \varepsilon)^{1/2}\|v\|_2$, which means it is between $(1 - \varepsilon/2)\|v\|_2$ and $(1 + \varepsilon/2)\|v\|_2$.

Note that since the updates to the AMS sketch can be positive or negative, it can be used to measure the Euclidean distance between two vectors v and u: we can build an AMS sketch of v and one of $-u$ and merge them together by adding the sketches. Note also that a sketch of $-u$ can be obtained from a sketch of u by negating all the counter values.

Applications

The sketch can also be applied to estimate the inner product between a pair of vectors. A similar analysis shows that the inner product of corresponding rows of two sketches (formed with the same parameters and using the same hash functions) is an unbiased estimator for the inner product of the vectors. This use of the summary to estimate the inner product of vectors was described in a follow-up work by Alon, Matias, Gibbons, and Szegedy [2], and the analysis was similarly generalized to the fast version by Cormode and Garofalakis [3]. The ability to capture norms and inner products in Euclidean space means that these sketches have found many applications in settings where there are high-dimensional vectors, such as machine learning and data mining.

URLs to Code and Data Sets

Sample implementations are widely available in a variety of languages.

C code is given by the MassDAL code bank: http://www.cs.rutgers.edu/~muthu/massdalcode-index.html.

C++ code given by Marios Hadjieleftheriou is available at http://hadjieleftheriou.com/sketches/index.html.

Cross-References

▸ Count-Min Sketch

Recommended Reading

1. Alon N, Matias Y, Szegedy M (1996) The space complexity of approximating the frequency moments. In: ACM symposium on theory of computing, Philadelphia, pp 20–29
2. Alon N, Gibbons P, Matias Y, Szegedy M (1999) Tracking join and self-join sizes in limited storage. In: ACM principles of database systems, New York, pp 10–20
3. Cormode G, Garofalakis M (2005) Sketching streams through the net: distributed approximate query tracking. In: International conference on very large data bases, Trondheim
4. Thorup M, Zhang Y (2004) Tabulation based 4-universal hashing with applications to second moment estimation. In: ACM-SIAM symposium on discrete algorithms, New Orleans

Analyzing Cache Behaviour in Multicore Architectures

Alejandro López-Ortiz[1] and Alejandro Salinger[2]
[1]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada
[2]Department of Computer Science, Saarland University, Saarbücken, Germany

Keywords

Cache; Chip multiprocessor; Multicore; Online algorithms; Paging

Years and Authors of Summarized Original Work

2010; Hassidim
2012; López-Ortiz, Salinger

Problem Definition

Multicore processors are commonly equipped with one or more levels of cache memory, some of which are shared among two or more cores. Multiple cores compete for the use of shared caches for fast access to their program's data, with the cache usage patterns of a program running on one core, possibly affecting the cache performance of programs running on other cores.

Paging

The management of data across the various levels of the memory hierarchy of modern computers is abstracted by the paging problem. Paging models a two-level memory system with a small and fast memory – known as cache – and a large and slow memory. Data is transferred between the two levels of memory in units known as pages. The input to the problem is a sequence of page requests that must be made available in cache as they are requested. If the currently requested page is already present in the cache, then this is known as a *hit*. Otherwise a *fault* occurs, and the requested page must be brought from slow memory to cache, possibly requiring the eviction of a page currently residing in the cache. An algorithm for this problem must decide, upon each request that results in a fault with a full cache, which page to evict in order to minimize the number of faults. Since the decision of which page to evict must be taken without information of future requests, paging is an online problem.

The most popular framework to analyze the performance of online algorithms is competitive analysis [10]: an algorithm A for a minimization problem is said to be c-competitive if its cost is at most c times that of an optimal algorithm that knows the input in advance. Formally, let $A(r)$ and $\mathrm{OPT}(r)$ denote the costs of A and the optimal algorithm OPT on an input r. Then A is c-competitive if for all inputs r, $A(r) \leq c \cdot \mathrm{OPT}(r) + \beta$, where β is a constant that does not depend on r. The infimum of all such values c is known as A's *competitive ratio*.

Traditional paging algorithms, like least recently used (LRU), evict the page currently

in cache that was least recently accessed, or first-in-first-out (FIFO), evict the page currently in the cache that was brought in the earliest, have an optimal competitive ratio equal to the cache size. Other optimal eviction policies are flush-when-full (FWF) and Clock (see [2] for definitions).

Paging in Multicore Caches

The paging problem described above can be extended to model several programs running simultaneously with a shared cache. For a multicore system with p cores sharing one cache, the multicore paging problem consists of a set r of p request sequences $r_1, \ldots r_p$ to be served with one shared cache of size k pages. At any timestep, at most p requests from different processors can arrive and must be served in parallel. A paging algorithm must decide which pages to evict when a fault occurs on a full cache.

The general model we consider for this problem was proposed by Hassidim [6]. This model defines the fetching time τ of a page as the ratio between a cache miss and a cache hit. A sequence of requests that suffers a page fault must wait τ timesteps for the page to be fetched into the cache, while other sequences that incur hits can continue to be served. In addition, paging algorithms can decide on the schedule of request sequences, choosing to serve a subset of the sequences and delay others. In this problem, the goal of a paging algorithm is to minimize the makespan. López-Ortiz and Salinger [8] proposed a slightly different model in which paging algorithms are not allowed to make scheduling decisions and must serve requests as they arrive. Furthermore, instead of minimizing the makespan, they propose two different goals: minimize the number of faults and decide if each of the sequences can be served with a number of faults below a given threshold. We consider both these settings here and the following problems:

Definition 1 (Min-Makespan) Given a set r of request sequences $r_1, \ldots, r_p$ to be served with a cache of size k, minimize the timestep at which the last request among all sequences is served.

Definition 2 (Min-Faults) Given a set r of requests $r_1, \ldots, r_p$ to be served with a cache of size k, minimize the total number of faults when serving r.

Definition 3 (Partial-Individual-Faults) Given a set r of requests $r_1, \ldots, r_p$ to be served with a cache of size k, a timestep t, and a bound b_i for each sequence, decide whether r can be served such that at time t the number of faults on r_i is at most b_i for all $1 \leq i \leq p$.

Key Results

Online Paging

For both the models of Hassidim and López-Ortiz and Salinger ,no online algorithm has been shown to be competitive, while traditional algorithms that are competitive in the classic paging setting are not competitive in the multicore setting. Hassidim shows that LRU and FIFO have a competitive ratio in the Min-Makespan problem of $\Theta(\tau)$, which is the worst possible for any online algorithm in this problem.

In the following, k is the size of the shared cache of an online algorithm, and h is the size of the shared cache of the optimal offline.

Theorem 1 ([6]) *For any $\alpha > 1$, the competitive ratio of LRU (or FIFO) is $\Theta(\tau/\alpha)$, when $h = k/\alpha$. In particular, if we give LRU a constant factor resource augmentation, the ratio is $\Theta(\tau)$. There is a setting with this ratio with just $\lceil \alpha \rceil + 1$ cores.*

The bad competitive ratio stems from the ability of the offline algorithm to schedule sequences one after the other one so that each sequence can use the entire cache. Meanwhile, LRU or FIFO will try to serve all sequences simultaneously, not having enough cache to satisfy the demands of any sequence. A similar result is shown in [8] for the Min-Faults problem, even in the case in which the optimal offline cannot explicitly schedule the input sequences. In this case, given a set of request sequences that alternate periods of high and low cache demand, the optimal offline algorithm can delay some sequences through faults in

order to align periods of high demands of some sequences with periods of low demands of others and with a total cache demand below capacity. As in the previous lower bound, traditional online algorithms will strive to serve all sequences simultaneously, incurring only faults in periods of high demand.

Theorem 2 ([8]) *Let A be any of LRU, FIFO, Clock, or FWF, let $p \geq 4$, let n be the total length of request sequences, and assume $\tau > 1$. The competitive ratio of A is at least $\Omega(\sqrt{n\tau/k})$ when the optimal offline's cache is $h \geq k/2 + 3p/2$. If A has no resource augmentation, the competitive ratio is at least $\Omega(\sqrt{n\tau p/k})$.*

These results give light about the characteristics required by online policies to achieve better competitive ratios. López-Ortiz and Salinger analyzed paging algorithms for Min-Faults , separating the cache partition and the eviction policy aspects. They defined partitioned strategies as those that give a portion of the cache to each core and serve the request sequences with a given eviction policy exclusively with the given part of the cache. The partition can be static or dynamic. They also define shared strategies as those in which all requests are served with one eviction policy using a global cache. The policies considered in Theorems 1 and 2 above are examples of shared strategies.

If a cache partition is determined externally by a scheduler or operating system, then traditional eviction policies can achieve a good performance when compared to the optimal eviction policy with the same partition. More formally,

Theorem 3 *Let A be any marking or conservative paging algorithm and B be any dynamically conservative algorithm [9] (these classes include LRU, FIFO, and Clock). Let S and D be any static and dynamic partition functions and let OPT_s and OPT_d denote the optimal eviction policies given S and D, respectively. Then, for all inputs r, $A(r) \leq k \cdot OPT_s(r)$ and $B(r) \leq pk \cdot OPT_d(r)$.*

The result above relies on a result by Peserico [9] which states that dynamically conservative policies are k-competitive when the size of the cache varies throughout the execution of the cache instance.

When considering a strategy as a partition plus eviction policy, it should not be a surprise that a strategy involving a static partition cannot be competitive. In fact, even a dynamic partition that does not change the sizes of the parts assigned to its cores often enough cannot be competitive. There are sequences for which the optimal static partition with the optimal paging policy in each part can incur a number of faults that is arbitrarily large compared to an online shared strategy using LRU. A similar result applies to dynamic partitions that change a sublinear number of times. These results suggest that in order to be competitive, an online strategy needs to be either shared or partitioned with a partition that changes often.

Offline Paging

We now consider the offline multicore paging problem. Hassidim shows that Min-Makespan is NP-hard for $k = p/3$ and [7] extends it to arbitrary k and p. In the model without scheduling of [8], Partial-Individual-Faults, a variant of the fault minimization problem, is also shown to be NP-hard. It is not known, however, whether Min-Faults is NP-hard as well. Interestingly, Partial-Individual-Faults remains NP-hard when $\tau = 1$ (and hence a fault does not delay the affected sequence with respect to other sequences). In contrast, in this case, Min-Faults can be solved simply by evicting the page that will be requested furthest in the future, as in classic paging. On the positive side, the following property holds for both Min-Makespan and Min-Faults (on disjoint sequences).

Theorem 4 *There exist optimal algorithms for Min-Makespan and Min-Faults that, upon a fault, evict the page that is the furthest in the future for some sequence.*

This result implies that multicore paging reduces to determining the optimal dynamic partition of the cache: upon a fault, the part of the cache of one sequence is reduced (unless this sequence is the same as the one which incurred the fault), and the page whose next request is

furthest is the future in this sequence should be evicted.

Finally, in the special case of a constant number of processors p and constant delay τ, Min-Makespan admits a polynomial time approximation scheme (PTAS), while Min-Faults and Partial-Individual-Faults admit exact polynomial time algorithms.

Theorem 5 ([6]) *There exists an algorithm that, given an instance of Min-Makespan with optimal makespan m, returns a solution with makespan $(1+\epsilon)m$. The running time is exponential on p, τ, and $1/\epsilon$.*

Theorem 6 ([8]) *An instance of Min-Faults with p requests of total length n, with $p = O(1)$ and $\tau = O(1)$ can be solved in time $O(n^{k+p}\tau^p)$.*

Theorem 7 ([8]) *An instance of Partial-Individual-Faults with p requests of total length n, with $p = O(1)$ and $\tau = O(1)$, can be solved in time $O(n^{k+2p+1}\tau^{p+1})$.*

Other Models

Paging with multiple sequences with a shared cache has also been studied in other models [1,3–5], even prior to multicores. In these models, request sequences may be interleaved; however, only one request is served at a time and all sequences must wait upon a fault affecting one sequence.

In the application-controlled model of Cao et al. [3], each process has full knowledge of its request sequence, while the offline algorithm also knows the interleaving of requests. As opposed to the models in [6, 8], the interleaving is fixed and does not depend on the decisions of algorithms. It has been shown that for p sequences and a cache of size k, no online deterministic algorithm can have a competitive ratio better than $p+1$ in the case where sequences are disjoint [1] and $\frac{p}{2}\log\left(\frac{4(k+1)}{3p}\right)$ otherwise [7]. On the other hand, there exist algorithms with competitive ratios $2(p+1)$ [1, 3] and $\max\{10, p+1\}$ [7] for the disjoint case, and $2p(\ln(ek/p)+1)$ [7] for the shared case.

Open Problems

Open problems in multicore paging are finding competitive online algorithms, determining the exact complexity of Min-Faults, obtaining approximation algorithms for Min-Makespan for a wider range of parameters, and obtaining faster exact offline algorithms for Min-Faults and Partial-Individual-Faults. Another challenge in multicore paging is concerned with modeling the right features of the multicore architecture while enabling the development of meaningful algorithms. Factors to consider are cache coherence, limited parallelism in other shared resources (such as bus bandwidth), different cache associativities, and others.

Cross-References

▸ Online Paging and Caching

Recommended Reading

1. Barve RD, Grove EF, Vitter JS (2000) Application-controlled paging for a shared cache. SIAM J Comput 29:1290–1303
2. Borodin A, El-Yaniv R (1998) Online computation and competitive analysis. Cambridge University Press, New York
3. Cao P, Felten EW, Li K (1994) Application-controlled file caching policies. In: Proceedings of the USENIX summer 1994 technical conference – volume 1 (USTC'94), Boston, pp 171–182
4. Feuerstein E, Strejilevich de Loma A (2002) On-line multi-threaded paging. Algorithmica 32(1):36–60
5. Fiat A, Karlin AR (1995) Randomized and multi-pointer paging with locality of reference. In: Proceedings of the 27th annual ACM symposium on theory of computing (STOC'95), Las Vegas. ACM, pp 626–634
6. Hassidim A (2010) Cache replacement policies for multicore processors. In: Yao ACC (ed) Innovations in computer science (ICS 2010), Tsinghua University, Beijing, 2010, pp 501–509
7. Katti AK, Ramachandran V (2012) Competitive cache replacement strategies for shared cache environments. In: International parallel and distributed processing symposium (IPDPS'12), Shanghai, pp 215–226
8. López-Ortiz A, Salinger A (2012) Paging for multicore shared caches. In: Proceedings of the 3rd innovations in theoretical computer science conference (ITCS'12), Cambridge. ACM, pp 113–127

9. Peserico E (2013) Elastic paging. In: SIGMETRICS, Pittsburgh. ACM, pp 349–350
10. Sleator DD, Tarjan RE (1985) Amortized efficiency of list update and paging rules. Commun ACM 28(2):202–208

Analyzing Cache Misses

Naila Rahman
University of Hertfordshire, Hertfordshire, UK

Keywords

Cache analysis

Years and Authors of Summarized Original Work

2003; Mehlhorn, Sanders

Problem Definition

The problem considered here is *multiple sequence access via cache memory*. Consider the following pattern of memory accesses. k sequences of data, which are stored in disjoint arrays and have a total length of N, are accessed as follows:

for $t := 1$ to N do
  select a sequence $s_i \in \{1, \ldots k\}$
  work on the current element of sequence s_i
  advance sequence s_i to the next element.

The aim is to obtain exact (not just asymptotic) closed form upper and lower bounds for this problem. Concurrent accesses to multiple sequences of data are ubiquitous in algorithms. Some examples of algorithms which use this paradigm are distribution sorting, k-way merging, priority queues, permuting, and FFT. This entry summarizes the analyses of this problem in [5, 8].

Caches, Models, and Cache Analysis

Modern computers have hierarchical memory which consists of registers, one or more levels of caches, main memory, and external memory devices such as disks and tapes. Memory size increases, but the speed decreases with distance from the CPU. Hierarchical memory is designed to improve the performance of algorithms by exploiting temporal and spatial locality in data accesses.

Caches are modeled as follows. A cache has m *blocks* each of which holds B data elements. The capacity of the cache is $M = mB$. Data is transferred between one level of cache and the next larger and slower memory in blocks of B elements. A cache is organized as $s = m/a$ *sets* where each set consists of a blocks. Memory at address xB, referred to as memory block x, can only be placed in a block in set $x \bmod s$. If $a = 1$, the cache is said to be *direct mapped*, and if $a = s$, it is said to be *fully associative*.

If memory block x is accessed and it is not in cache, then a *cache miss* occurs, and the data in memory block x is brought into cache, incurring a performance penalty. In order to accommodate block x, it is assumed that the least recently used (LRU) or the first used (FIFO) block from the cache set $x \bmod s$ is evicted, and this is referred to as the *replacement strategy*. Note that a block may be evicted from a set, even though there may be unoccupied blocks in other sets.

Cache analysis is performed for the number of cache misses for a problem with N data elements. To read or write N data elements, an algorithm must incur $\Omega(N/B)$ cache misses. These are the *compulsory* or *first reference misses*. In the multiple sequence access via cache memory problem, for given values of M and B, one aim is to find the largest k such that there are $O(N/B)$ cache misses for the N data accesses. It is interesting to analyze cache misses for the important case of direct mapped cache and for the general case of set-associative caches.

A large number of algorithms have been designed on the external memory model [11], and these algorithms optimize the number of data transfers between main memory and disk. It seems natural to exploit these algorithms

to minimize cache misses, but due to the limited associativity of caches, this is not straightforward. In the external memory model, data transfers are under programmer control, and the multiple sequence access problem has a trivial solution. The algorithm simply chooses $k \leq M_e/B_e$, where B_e is the block size and M_e is the capacity of the main memory in the external memory model. For $k \leq M_e/B_e$, there are $O(N/B_e)$ accesses to external memory. Since caches are hardware controlled, the problem becomes nontrivial. For example, consider the case where the starting addresses of $k > a$ equal length sequences map to the ith element of the same set, and the sequences are accessed in a round-robin fashion. On a cache with an LRU or FIFO replacement strategy, all sequence accesses will result in a cache miss. Such pathological cases can be overcome by randomizing the starting addresses of the sequences.

Related Problems

A very closely related problem is where accesses to the sequences are interleaved with accesses to a small working array. This occurs in applications such as distribution sorting or matrix multiplication.

Caches can emulate external memory with an optimal replacement policy [3, 10]; however, this requires some constant factor more memory. Since the emulation techniques are software controlled and require modification to the algorithm, rather than selection of parameters, they work well for fairly simple algorithms [6].

Key Results

Theorem 1 ([5]) *Given an a-way set-associative cache with m cache blocks, $s = m/a$ cache sets, cache blocks size B, and LRU or FIFO replacement strategy. Let U_a denote the expected number of cache misses in any schedule of N sequential accesses to k sequences with starting addresses that are at least $(a+1)$-wise independent:*

$$U_1 \leq k + \frac{N}{B}\left(1 + (B-1)\frac{k}{m}\right), \tag{1}$$

$$U_1 \geq \frac{N}{B}\left(1 + (B-1)\frac{k-1}{m+k-1}\right), \tag{2}$$

$$U_a \leq k + \frac{N}{B}\left(1 + (B-1)\left(\frac{k\alpha}{m}\right)^a + \frac{1}{m/(k\alpha)-1} + \frac{k-1}{s-1}\right) \tag{3}$$

$$\text{for } k \leq \frac{m}{\alpha},$$

$$U_a \leq k + \frac{N}{B}\left(1 + (B-1)\left(\frac{k\beta}{m}\right)^a + \frac{1}{m/(k\beta)-1}\right) \tag{4}$$

$$\text{for } k \leq \frac{m}{2\alpha},$$

$$U_a \geq \frac{N}{B}\left(1 + (B-1)P_{\text{tail}}\left(k-1, \frac{1}{s}, a\right)\right) - kM, \tag{5}$$

$$U_a \geq \frac{N}{B}\left(1 + (B-1)\left(\frac{(k-a)\alpha}{m}\right)^a \left(1 - \frac{1}{-}s\right)^k\right) - kM, \tag{6}$$

where $\alpha = \alpha(a) = a/(a!)^{1/a}$, $P_{\text{tail}}(n, p, a) = \sum_{i \geq a} \binom{n}{i} p^i (1-p)^{n-i}$ *is the cumulative binomial probability, and* $\beta := 1 + \alpha(\lceil ax \rceil)$ *where* $x = x(a) = \inf\{0 < z < 1 : z + z/\alpha(\lceil az \rceil) = 1\}$.

Here $1 \leq \alpha < e$ and $\beta(1) = 2$, $\beta(\infty) = 1 + e \approx 3.71$. This analysis assumes that an adversary schedules the accesses to the sequences. For the lower bound, the adversary initially advances sequence s_i for $i = 1 \ldots k$ by X_i elements, where the X_i is chosen uniformly and independently from $\{0, M-1\}$. The adversary then accesses the sequences in a round-robin manner.

The k in the upper bound accounts for a possible extra block that may be accessed due to randomization of the starting addresses. The $-kM$ term in the lower bound accounts for the fact that cache misses cannot be counted when the adversary initially winds forwards the sequences.

The bounds are of the form $pN + c$, where c does not depend on N and p is called the *cache miss probability*. Letting $r = k/m$, the ratio between the number of sequences and the number of cache blocks, the bounds for the cache miss probabilities in Theorem 1 become [5]

$$p_1 \leq (1/B)(1 + (B-1)r), \tag{7}$$

$$p_1 \geq (1/B)\left(1 + (B-1)\frac{r}{1+r}\right), \tag{8}$$

$$p_a \leq (1/B)(1 + (B-1)(r\alpha)^a + r\alpha + ar) \quad \text{for } r \leq \frac{1}{\alpha}, \tag{9}$$

$$p_a \leq (1/B)(1 + B - 1)(r\beta)^a + r\beta \text{ for } r \leq \frac{1}{2\beta} \tag{10}$$

$$p_a(1/B)\left(1 + (B-1)(r\alpha)^a\left(1 - \frac{1}{s}\right)^k\right). \tag{11}$$

The $1/B$ term accounts for the compulsory or first reference miss, which must be incurred in order to read a block of data from a sequence. The remaining terms account for *conflict misses*, which occur when a block of data is evicted from cache before all its elements have been scanned. Conflict misses can be reduced by restricting the number of sequences. As r approaches zero, the cache miss probabilities approach $1/B$. In general, inequality (4) states that the number of cache misses is $O(N/B)$ if $r \leq 1/(2\beta)$ and $(B-1)(r\beta)^a = O(1)$. Both of these conditions are satisfied if $k \leq m/\max(B^{1/a}, 2\beta)$. So, there are $O(N/B)$ cache misses provided $k = O(m/B^{1/a})$.

The analysis shows that for a direct-mapped cache, where $a = 1$, the upper bound is a factor of $r + 1$ above the lower bound. For $a \geq 2$, the upper bounds and lower bounds are close if $(1 - 1/s)^k \approx$ and $(\alpha + \alpha)r \ll 1$, and both these conditions are satisfied if $k \ll s$.

Rahman and Raman [8] obtain closer upper and lower bounds for average case cache misses assuming the sequences are accessed uniformly randomly on a direct-mapped cache. Sen and Chatterjee [10] also obtain upper and lower bounds assuming the sequences are randomly accessed. Ladner, Fix, and LaMarca have analyzed the problem on direct-mapped caches on the independent reference model [4].

Multiple Sequence Access with Additional Working Set

As stated earlier in many applications, accesses to sequences are interleaved with accesses to an additional data structure, a *working set*, which determines how a sequence element is to be treated. Assuming that the working set has size at most sB and is stored in contiguous memory locations, the following is an upper bound on the number of cache misses:

Theorem 2 ([5]) *Let* U_a *denote the bound on the number of cache misses in Theorem 1 and define* $U_0 = N$. *With the working set occupying* w *conflict-free memory blocks, the expected number of cache misses arising in the* N *accesses to the sequence data, and any number of accesses to the working set, is bounded by* $w + (1 - w/s)U_a + 2(w/s)U_{a-1}$.

On a direct-mapped cache, for $i = 1, \ldots, k$, if sequence i is accessed with probability p_i independently of all previous accesses and is followed by an access to element i of the working set, then the following are upper and lower bounds for the number of cache misses:

Theorem 3 ([8]) *In a direct-mapped cache with m cache blocks, each of B elements, if sequence i, for $i = , \ldots, k$, is accessed with probability p_i and block j of the working set, for $j = , \ldots, k/B$, is accessed with probability P_j, then the expected number of cache misses in N sequence accesses is at most $N(p_s + p_w) + k(1 + 1/B)$, where*

$$p_s \leq \frac{1}{B} + \frac{k}{mB} + \frac{B-1}{mB} \sum_{i=1}^{k} \left(\sum_{j=1}^{k/B} \frac{p_i P_j}{p_i + P_j} + \frac{B-1}{B} \sum_{j=1}^{k} \frac{p_i p_j}{p_i + p_j} \right),$$

$$p_w \leq \frac{k}{B^2 m} + \frac{B-1}{mB} \sum_{i=1}^{k/B} \sum_{j=1}^{k} \frac{P_i p_j}{P_i p_j}.$$

Theorem 4 ([8]) *In a direct-mapped cache with m cache blocks each of B elements, if sequence i, for $i = 1, \ldots, k$, is accessed with probability $p_i \geq 1/m$, then the expected number of cache misses in N sequence accesses is at least $Np_s + k$, where*

$$p_s \geq \frac{1}{B} + \frac{k(2m-k)}{2m^2} + \frac{k(k-3m)}{2Bm^2} - \frac{1}{2Bm} - \frac{k}{2B^2 m} + \frac{B(k-m) + 2m - 3k}{Bm^2} \sum_{i=1}^{k} \sum_{j=1}^{k} \frac{(p_i)^2}{p_i + p_j}$$
$$+ \frac{(B-1)^2}{B^3 m^2} \sum_{i=1}^{k} p_i \left[\sum_{j=1}^{k} \frac{p_i(1 - p_i - p_j)}{(p_i + p_j)^2} - \frac{B-1}{2} \sum_{j=1}^{k} \sum_{l=1}^{k} \frac{p_i}{p_i + p_j + p_l - p_j p_l} \right] - O(e^{-B}).$$

The lower bound ignores the interaction with the working set, since this can only increase the number of cache misses.

In Theorems 3 and 4, p_s is the probability of a cache miss for a sequence access, and in Theorem 3, p_w is the probability of a cache miss for an accesses to the working set.

If the sequences are accessed uniformly randomly, then using Theorems 3 and 4, the ratio between the upper and lower bound is $3/(3 - r)$, where $r = k/m$. So for uniformly random data, the lower bound is within a factor of about $3/2$ of the upper bound when $k \leq m$ and is much closer when $k \ll m$.

Applications

Numerous algorithms have been developed on the external memory model which access multiple sequences of data, such as merge sort, distribution sort, priority queues, and radix sorting. These analyses are important as they allow initial parameter choices to be made for cache memory algorithms.

Open Problems

The analyses assume that the starting addresses of the sequences are randomized, and current approaches to allocating random starting addresses waste a lot of virtual address space [5]. An open problem is to find a good online scheme to randomize the starting addresses of arbitrary length sequences.

Experimental Results

The cache model is a powerful abstraction of real caches; however, modern computer architectures have complex internal memory hierarchies, with

registers, multiple levels of caches, and *translation lookaside buffers* (TLB). Cache miss penalties are not of the same magnitude as the cost of disk accesses, so an algorithm may perform better by allowing conflict misses to increase in order to reduce computation costs and compulsory misses, by reducing the number of passes over the data. This means that in practice, cache analysis is used to choose an initial value of k which is then fine-tuned for the platform and algorithm [1,2,6,7,9,12,13].

For distribution sorting, in [6], a heuristic was considered for selecting k, and equations for approximate cache misses were obtained. These equations were shown to be very accurate in practice.

Cross-References

- Cache-Oblivious Model
- Cache-Oblivious Sorting
- External Sorting and Permuting
- I/O-Model

Recommended Reading

1. Bertasi P, Bressan M, Peserico E (2011) Psort, yet another fast stable sorting software. ACM J Exp Algorithmics 16:Article 2.4
2. Bingmann T, Sanders P (2013) Parallel string sample sort. In: Proceedings of the 21st European symposium on algorithms (ESA'13), Sophia Antipolis. Springer, pp 169–180
3. Frigo M, Leiserson CE, Prokop H, Ramachandran S (1999) Cache-oblivious algorithms. In: Proceedings of the 40th annual symposium on foundations of computer science (FOCS'99), New York. IEEE Computer Society, Washington, DC, pp 285–297
4. Ladner RE, Fix JD, LaMarca A (1999) Cache performance analysis of traversals and random accesses. In: Proceedings of the 10th annual ACM-SIAM symposium on discrete algorithms (SODA'99), Baltimore. Society for Industrial and Applied Mathematics, Philadelphia, pp 613–622
5. Mehlhorn K, Sanders P (2003) Scanning multiple sequences via cache memory. Algorithmica 35:75–93
6. Rahman N, Raman R (2000) Analysing cache effects in distribution sorting. ACM J Exp Algorithmics 5:Article 14
7. Rahman N, Raman R (2001) Adapting radix sort to the memory hierarchy. ACM J Exp Algorithmics 6:Article 7
8. Rahman N, Raman R (2007) Cache analysis of non-uniform distribution sorting algorithms. http://www.citebase.org/abstract?id=oai:arXiv.org:0706.2839. Accessed 13 Aug 2007 Preliminary version in: Proceedings of the 8th annual European symposium on algorithms (ESA'00), Saarbrücken. Lecture notes in computer science, vol 1879. Springer, Berlin/Heidelberg, pp 380–391 (2000)
9. Sanders P (2000) Fast priority queues for cached memory. ACM J Exp Algorithmics 5:Article 7
10. Sen S, Chatterjee S (2000) Towards a theory of cache-efficient algorithms. In: Proceedings of the 11th annual ACM-SIAM symposium on discrete algorithms (SODA'00), San Francisco. Society for Industrial and Applied Mathematics, pp 829–838
11. Vitter JS (2001) External memory algorithms and data structures: dealing with massive data. ACM Comput Surv **33**, 209–271
12. Wassenberg J, Sanders P (2011) Engineering a Multi-core Radix Sort, In: Proceedings of the 17th international conference, Euro-Par (2) 2011, Bordeaux. Springer, pp 160–169
13. Wickremesinghe R, Arge L, Chase JS, Vitter JS (2002) Efficient sorting using registers and caches. ACM J Exp Algorithmics 7:9

Applications of Geometric Spanner Networks

Joachim Gudmundsson[1,2], Giri Narasimhan[3,4], and Michiel Smid[5]
[1]DMiST, National ICT Australia Ltd, Alexandria, Australia
[2]School of Information Technologies, University of Sydney, Sydney, NSW, Australia
[3]Department of Computer Science, Florida International University, Miami, FL, USA
[4]School of Computing and Information Sciences, Florida International University, Miami, FL, USA
[5]School of Computer Science, Carleton University, Ottawa, ON, Canada

Keywords

Approximation algorithms; Cluster graphs; Dilation; Distance oracles; Shortest paths; Spanners

Years and Authors of Summarized Original Work

2002; Gudmundsson, Levcopoulos, Narasimhan, Smid
2005; Gudmundsson, Narasimhan, Smid
2008; Gudmundsson, Levcopoulos, Narasimhan, Smid

Problem Definition

Given a geometric graph in d-dimensional space, it is useful to preprocess it so that distance queries, exact or approximate, can be answered efficiently. Algorithms that can report distance queries in constant time are also referred to as "distance oracles." With unlimited preprocessing time and space, it is clear that exact distance oracles can be easily designed. This entry sheds light on the design of approximate distance oracles with limited preprocessing time and space for the family of geometric graphs with constant dilation.

Notation and Definitions

If p and q are points in $\mathcal{R}^d$, then the notation $|pq|$ is used to denote the Euclidean distance between p and q; the notation $\delta_G(p,q)$ is used to denote the Euclidean length of a shortest path between p and q in a geometric network G. Given a constant $t > 1$, a graph G with vertex set S is a t-spanner for S if $\delta_G(p,q) \leq t|pq|$ for any two points p and q of S. A t-spanner network is said to have *dilation* (or *stretch factor*) t. A $(1+\varepsilon)$-approximate shortest path between p and q is defined to be any path in G between p and q having length Δ, where $\delta_G(p,q) \leq \Delta \leq (1+\varepsilon)\delta_G(p,q)$. For a comprehensive overview of geometric spanners, see the book by Narasimhan and Smid [14].

All networks considered in this entry are simple and undirected. The model of computation used is the traditional algebraic computation tree model with the added power of indirect addressing. In particular, the algorithms presented here do not use the non-algebraic floor function as a unit-time operation. The problem is formalized below.

Problem 1 (Distance Oracle) Given an arbitrary real constant $\epsilon > 0$, and a geometric graph G in d-dimensional Euclidean space with constant dilation t, design a data structure that answers $(1+\epsilon)$-approximate shortest path length queries in constant time.

The data structure can also be applied to solve several other problems. These include (a) the problem of reporting approximate distance queries between vertices in a planar polygonal domain with "rounded" obstacles, (b) query versions of *closest pair* problems, and (c) the efficient computation of the approximate dilations of geometric graphs.

Survey of Related Research

The design of efficient data structures for answering distance queries for general (non-geometric) networks was considered by Thorup and Zwick [17] (unweighted general graphs), Baswanna and Sen [3] (weighted general graphs, i.e., arbitrary metrics), and Arikati et al. [2] and Thorup [16] (weighted planar graphs).

For the geometric case, variants of the problem have been considered in a number of papers (for a recent paper, see, e.g., Chen et al. [5]). Work on the approximate version of these variants can also be found in many articles (for a recent paper, see, e.g., Agarwal et al. [1]). The focus of this entry is the results reported in the work of Gudmundsson et al. [10–13]. Similar results on distance oracles were proved subsequently for *unit disk graphs* [7]. Practical implementations of distance oracles in geometric networks have also been investigated [15].

Key Results

The main result of this entry is the existence of approximate distance oracle data structures for geometric networks with constant dilation (see Theorem 4 below). As preprocessing, the network is "pruned" so that it only has a linear number of edges. The data structure consists of

a series of "cluster graphs" of increasing coarseness, each of which helps answer approximate queries for pairs of points with interpoint distances of different scales. In order to pinpoint the appropriate cluster graph to search in for a given query, the data structure uses the bucketing tool described below. The idea of using cluster graphs to speed up geometric algorithms was first introduced by Das and Narasimhan [6] and later used by Gudmundsson et al. [9] to design an efficient algorithm to compute $(1+\varepsilon)$-spanners. Similar ideas were explored by Gao et al. [8] for applications to the design of mobile networks.

Pruning

If the input geometric network has a superlinear number of edges, then the preprocessing step for the distance oracle data structure involves efficiently "pruning" the network so that it has only a linear number of edges. The pruning may result in a small increase of the dilation of the spanner. The following theorem was proved by Gudmundsson et al. [12].

Theorem 1 *Let $t > 1$ and $\varepsilon' > 0$ be real constants. Let S be a set of n points in $\mathcal{R}^d$, and let $G = (S, E)$ be a t-spanner for S with m edges. There exists an algorithm to compute in $O(m + n \log n)$ time, a $(1+\varepsilon')$-spanner of G having $O(n)$ edges and whose weight is $O(wt(MST(S)))$.*

The pruning step requires the following technical theorem proved by Gudmundsson et al. [12].

Theorem 2 *Let S be a set of n points in $\mathcal{R}^d$, and let $c \geq 7$ be an integer constant. In $O(n \log n)$ time, it is possible to compute a data structure $D(S)$ consisting of:*

1. *A sequence $L_1, L_2, \ldots, L_\ell$ of real numbers, where $\ell = O(n)$, and*
2. *A sequence $S_1, S_2, \ldots, S_\ell$ of subsets of S such that $\sum_{i=1}^{\ell} |S_i| = O(n)$,*

such that the following holds. For any two distinct points p and q of S, it is possible to compute in $O(1)$ time an index i with $1 \leq i \leq \ell$ and two points x and y in S_i such that (a) $L_i/n^{c+1} \leq |xy| < L_i$ and (b) both $|px|$ and $|qy|$ are less than $|xy|/n^{c-2}$.

Despite its technical nature, the above theorem is of fundamental importance to this work. In particular, it helps to deal with networks where the interpoint distances are not confined to a polynomial range, i.e., there are pairs of points that are very close to each other and very far from each other.

Bucketing

Since the model of computation assumed here does not allow the use of floor functions, an important component of the algorithm is a "bucketing tool" that allows (after appropriate preprocessing) constant-time computation of a quantity referred to as BINDEX, which is defined to be the floor of the logarithm of the interpoint distance between any pair of input points.

Theorem 3 *Let S be a set of n points in $\mathcal{R}^d$ that are contained in the hypercube $(0, n^k)^d$, for some positive integer constant k, and let ε be a positive real constant. The set S can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n)$, such that for any two points p and q of S, with $|pq| \geq 1$, it is possible to compute in constant time the quantity $\text{BINDEX}_\varepsilon(p, q) = \lfloor \log_{1+\varepsilon} |pq| \rfloor$.*

The constant-time computation mentioned in Theorem 3 is achieved by reducing the problem to one of answering least common ancestor queries for pairs of nodes in a tree, a problem for which constant-time solutions were devised most recently by Bender and Farach-Colton [4].

Main Results

Using the bucketing and the pruning tools, and using the algorithms described by Gudmundsson et al. [13], the following theorem can be proved.

Theorem 4 *Let $t > 1$ and $\varepsilon > 0$ be real constants. Let S be a set of n points in $\mathcal{R}^d$, and let $G = (S, E)$ be a t-spanner for S with m edges. The graph G can be preprocessed into a data structure of size $O(n \log n)$ in time $O(mn \log n)$,*

such that for any pair of query points $p, q \in S$, it is possible to compute a $(1+\varepsilon)$-approximation of the shortest path distance in G between p and q in $O(1)$ time. Note that all the big-Oh notations hide constants that depend on d, t, and ε.

Additionally, if the traditional algebraic model of computation (without indirect addressing) is assumed, the following weaker result can be proved.

Theorem 5 *Let S be a set of n points in $\mathcal{R}^d$, and let $G = (S, E)$ be a t-spanner for S, for some real constant $t > 1$, having m edges. Assuming the algebraic model of computation, in $O(m \log\log n + n \log^2 n)$ time, it is possible to preprocess G into a data structure of size $O(n \log n)$, such that for any two points p and q in S, a $(1+\varepsilon)$-approximation of the shortest-path distance in G between p and q can be computed in $O(\log\log n)$ time.*

Applications

As mentioned earlier, the data structure described above can be applied to several other problems. The first application deals with reporting distance queries for a planar domain with polygonal obstacles. The domain is further constrained to be t-rounded, which means that the length of the shortest obstacle-avoiding path between any two points in the input point set is at most t times the Euclidean distance between them. In other words, the visibility graph is required to be a t-spanner for the input point set.

Theorem 6 *Let $\mathcal{F}$ be a t-rounded collection of polygonal obstacles in the plane of total complexity n, where t is a positive constant. One can preprocess $\mathcal{F}$ in $O(n \log n)$ time into a data structure of size $O(n \log n)$ that can answer obstacle-avoiding $(1+\varepsilon)$-approximate shortest path length queries in time $O(\log n)$. If the query points are vertices of $\mathcal{F}$, then the queries can be answered in $O(1)$ time.*

The next application of the distance oracle data structure includes query versions of *closest pair* problems, where the queries are confined to specified subset(s) of the input set.

Theorem 7 *Let $G = (S, E)$ be a geometric graph on n points and m edges, such that G is a t-spanner for S, for some constant $t > 1$. One can preprocess G in time $O(m + n \log n)$ into a data structure of size $O(n \log n)$ such that given a query subset S' of S, a $(1+\varepsilon)$-approximate closest pair in S' (where distances are measured in G) can be computed in time $O(|S'| \log |S'|)$.*

Theorem 8 *Let $G = (S, E)$ be a geometric graph on n points and m edges, such that G is a t-spanner for S, for some constant $t > 1$. One can preprocess G in time $O(m + n \log n)$ into a data structure of size $O(n \log n)$ such that given two disjoint query subsets X and Y of S, a $(1+\varepsilon)$-approximate bichromatic closest pair (where distances are measured in G) can be computed in time $O((|X| + |Y|) \log(|X| + |Y|))$.*

The last application of the distance oracle data structure includes the efficient computation of the approximate dilations of geometric graphs.

Theorem 9 *Given a geometric graph on n vertices with m edges, and given a constant C that is an upper bound on the dilation t of G, it is possible to compute a $(1+\varepsilon)$-approximation to t in time $O(m + n \log n)$.*

Open Problems

Two open problems remain unanswered:

1. Improve the space utilization of the distance oracle data structure from $O(n \log n)$ to $O(n)$.
2. Extend the approximate distance oracle data structure to report not only the approximate distance but also the approximate shortest path between the given query points.

Cross-References

▸ Geometric Spanners
▸ Sparse Graph Spanners

Recommended Reading

1. Agarwal PK, Har-Peled S, Karia M (2000) Computing approximate shortest paths on convex polytopes. In: Proceedings of the 16th ACM symposium on computational geometry, Hong Kong, pp 270–279
2. Arikati S, Chen DZ, Chew LP, Das G, Smid M, Zaroliagis CD (1996) Planar spanners and approximate shortest path queries among obstacles in the plane. In: Proceedings of the 4th annual european symposium on algorithms, Barcelona. Lecture notes in computer science, vol 1136. Springer, Berlin, pp 514–528
3. Baswana S, Sen S (2004) Approximate distance oracles for unweighted graphs in $\tilde{O}(n^2)$ time. In: Proceedings of the 15th ACM-SIAM symposium on discrete algorithms, Philadelphia, pp 271–280
4. Bender MA, Farach-Colton M (2000) The LCA problem revisited. In: Proceedings of the 4th Latin American symposium on theoretical informatics, Punta del Este. Lecture notes in computer science, vol 1776. Springer, Berlin, pp 88–94
5. Chen DZ, Daescu O, Klenk KS (2001) On geometric path query problems. Int J Comput Geom Appl 11:617–645
6. Das G, Narasimhan G (1997) A fast algorithm for constructing sparse Euclidean spanners. Int J Comput Geom Appl 7:297–315
7. Gao J, Zhang L (2005) Well-separated pair decomposition for the unit-disk graph metric and its applications. SIAM J Comput 35(1):151–169
8. Gao J, Guibas LJ, Hershberger J, Zhang L, Zhu A (2003) Discrete mobile centers. Discret Comput Geom 30:45–63
9. Gudmundsson J, Levcopoulos C, Narasimhan G (2002) Fast greedy algorithms for constructing sparse geometric spanners. SIAM J Comput 31:1479–1500
10. Gudmundsson J, Levcopoulos C, Narasimhan G, Smid M (2002) Approximate distance oracles for geometric graphs. In: Proceedings of the 13th ACM-SIAM symposium on discrete algorithms, San Francisco, pp 828–837
11. Gudmundsson J, Levcopoulos C, Narasimhan G, Smid M (2002) Approximate distance oracles revisited. In: Proceedings of the 13th international symposium on algorithms and computation, Osaka. Lecture notes in computer science, vol 2518. Springer, Berlin, pp 357–368
12. Gudmundsson J, Narasimhan G, Smid M (2005) Fast pruning of geometric spanners. In: Proceedings of the 22nd symposium on theoretical aspects of computer science, Stuttgart. Lecture notes in computer science, vol 3404. Springer, Berlin, pp 508–520
13. Gudmundsson J, Levcopoulos C, Narasimhan G, Smid M (2008) Approximate distance oracles for geometric spanners. ACM Trans Algorithms 4(1):article 10
14. Narasimhan G, Smid M (2007) Geometric spanner networks. Cambridge University, Cambridge
15. Sankaranarayanan J, Samet H (2010) Query processing using distance oracles for spatial networks. IEEE Trans Knowl Data Eng 22(8):1158–1175
16. Thorup M (2004) Compact oracles for reachability and approximate distances in planar digraphs. J ACM 51:993–1024
17. Thorup M, Zwick U (2001) Approximate distance oracles. In: Proceedings of the 33rd annual ACM symposium on the theory of computing, Crete, pp 183–192

Approximate Dictionaries

Venkatesh Srinivasan
Department of Computer Science, University of Victoria, Victoria, BC, Canada

Keywords

Cell probe model; Data structures; Static membership

Years and Authors of Summarized Original Work

2002; Buhrman, Miltersen, Radhakrishnan, Venkatesh

Problem Definition

The Problem and the Model

A static data structure problem consists of a set of data D, a set of queries Q, a set of answers A, and a function $f : D \times Q \to A$. The goal is to store the data succinctly, so that any query can be answered with only a few probes to the data structure. *Static membership* is a well-studied problem in data structure design [2, 6, 9, 10, 16, 17, 23].

Definition 1 (Static Membership) In the static membership problem, one is given a subset S of at most n keys from a universe $U = \{1, 2, \ldots, m\}$. The task is to store S so that queries of the form "Is u in S?" can be answered by making few accesses to the memory.

A natural and general model for studying any data structure problem is the *cell probe model* proposed by Yao [23].

Definition 2 (Cell Probe Model) An (s, w, t) cell probe scheme for a static data structure problem $f : D \times Q \to A$ has two components: a storage scheme and a query scheme. The storage scheme stores the data $d \in D$ as a Table $T[d]$ of s cells, each cell of word size w bits. The storage scheme is deterministic. Given a query $q \in Q$, the query scheme computes $f(d, q)$ by making at most t probes to $T[d]$, where each probe reads one cell at a time, and the probes can be adaptive. In a deterministic cell probe scheme, the query scheme is deterministic. In a randomized cell probe scheme, the query scheme is randomized and is allowed to err with a small probability.

Buhrman et al. [3] study the complexity of the static membership problem in the *bitprobe* model. The bitprobe model is a variant of the cell probe model in which each cell holds just a single bit. In other words, the word size w is 1. Thus, in this model, the query algorithm is given bitwise access to the data structure. The study of the membership problem in the bitprobe model was initiated by Minsky and Papert in their book *Perceptrons* [16]. However, they were interested in *average*-case upper bounds for this problem, while this work studies *worst*-case bounds for the membership problem.

Observe that if a scheme is required to store sets of size at most n, then it must use at least $\lceil \log \sum_{i \le n} \binom{m}{i} \rceil$ number of bits. If $n \le m^{1-\Omega(1)}$, this implies that the scheme must store $\Omega(n \log m)$ bits (and therefore use $\Omega(n)$ cells). The goal in [3] is to obtain a scheme that answers queries, uses only constant number of bitprobes, and at the same time uses a table of $O(n \log m)$ bits.

Related Work

The static membership problem has been well studied in the cell probe model, where each cell is capable of holding one element of the universe. That is, $w = O(\log m)$. In a seminal paper, Fredman, Komlós, and Szemerédi [10] proposed a scheme for the static membership problem in the cell probe model with word size $O(\log m)$ that used a constant number of probes and a table of size $O(n)$. This scheme will be referred to as the FKS scheme. Thus, up to constant factors, the FKS scheme uses optimal space and number of cell probes. In fact, Fiat et al. [9], Brodnik and Munro [2], and Pagh [17] obtain schemes that use space (in bits) that is within a small additive term of $\lceil \log \sum_{i \le n} \binom{m}{i} \rceil$ and yet answer queries by reading at most a constant number of cells. Despite all these fundamental results for the membership problem in the cell probe model, very little was known about the bitprobe complexity of static membership prior to the work in [3].

Key Results

Buhrman et al. investigate the complexity of the static membership problem in the bitprobe model. They study

- Two-sided error randomized schemes that are allowed to err on positive instances as well as negative instances (i.e., these schemes can say "No" with a small probability when the query element u is in the set S and "Yes" when it is not).
- One-sided error randomized schemes where the errors are restricted to negative instances alone (i.e., these schemes never say "No" when the query element u is in the set S);
- Deterministic schemes in which no errors are allowed.

The main techniques used in [3] are based on 2-colorings of special set systems that are related to r-cover-free family of sets considered in [5, 7, 11]. The reader is referred to [3] for further details.

Randomized Schemes with Two-Sided Error

The main result in [3] shows that there are randomized schemes that use *just one bitprobe* and

yet use space close to the information theoretic lower bound of $\Omega(n \log m)$ bits.

Theorem 1 *For any $0 < \epsilon \leq \frac{1}{4}$, there is a scheme for storing subsets S of size at most n of a universe of size m using $O\left(\frac{n}{\epsilon^2} \log m\right)$ bits so that any membership query "Is $u \in S$?" can be answered with error probability at most ϵ by a randomized algorithm which probes the memory at just one location determined by its coin tosses and the query element u.*

Note that randomization is allowed only in the query algorithm. It is still the case that for each set S, there is exactly one associated data structure $T(S)$. It can be shown that deterministic schemes that answer queries using a single bitprobe need m bits of storage (see the remarks following Theorem 4). Theorem 1 shows that, by allowing randomization, this bound (for constant ϵ) can be reduced to $O(n \log m)$ bits. This space is within a constant factor of the information theoretic bound for n sufficiently small. Yet, the randomized scheme answers queries using a single bitprobe.

Unfortunately, the construction above does not permit us to have sub-constant error probability and still use optimal space. Is it possible to improve the result of Theorem 1 further and design such a scheme? Buhrman et al. [3] shows that this is not possible: if ϵ is made sub-constant, then the scheme must use more than $n \log m$ space.

Theorem 2 *Suppose $\frac{n}{m^{1/3}} \leq \epsilon \leq \frac{1}{4}$. Then, any two-sided ϵ-error randomized scheme which answers queries using one bitprobe must use space $\Omega\left(\frac{n}{\epsilon \log(1/\epsilon)} \log m\right)$.*

Randomized Schemes with One-Sided Error

Is it possible to have any savings in space if the query scheme is expected to make only one-sided errors? The following result shows it is possible if the error is allowed only on negative instances.

Theorem 3 *For any $0 < \epsilon \leq \frac{1}{4}$, there is a scheme for storing subsets S of size at most n of a universe of size m using $O\left(\left(\frac{n}{\epsilon}\right)^2 \log m\right)$ bits so that any membership query "Is $u \in S$?" can be answered with error probability at most ϵ by a randomized algorithm which makes a single bitprobe to the data structure. Furthermore, if $u \in S$, the probability of error is 0.*

Though this scheme does not operate with optimal space, it still uses significantly less space than a bitvector. However, the dependence on n is quadratic, unlike in the two-sided scheme where it was linear. Buhrman et al. [3] shows that this scheme is essentially optimal: there is necessarily a quadratic dependence on $\frac{n}{\epsilon}$ for any scheme with one-sided error.

Theorem 4 *Suppose $\frac{n}{m^{1/3}} \leq \epsilon \leq \frac{1}{4}$. Consider the static membership problem for sets S of size at most n from a universe of size m. Then, any scheme with one-sided error ϵ that answers queries using at most one bitprobe must use $\Omega\left(\frac{n^2}{\epsilon^2 \log(n/\epsilon)} \log m\right)$ bits of storage.*

Remark 1 One could also consider one-probe one-sided error schemes that only make errors on positive instances. That is, no error is made for query elements *not* in the set S. In this case, [3] shows that randomness does not help at all: such a scheme must use m bits of storage.

The following result shows that the space requirement can be reduced further in one-sided error schemes if more probes are allowed.

Theorem 5 *Suppose $0 < \delta < 1$. There is a randomized scheme with one-sided error $n^{-\delta}$ that solves the static membership problem using $O\left(n^{1+\delta} \log m\right)$ bits of storage and $O\left(\frac{1}{\delta}\right)$ bitprobes.*

Deterministic Schemes

In contrast to randomized schemes, Buhrman et al. show that deterministic schemes exhibit a time-space tradeoff behavior.

Theorem 6 *Suppose a deterministic scheme stores subsets of size n from a universe of size m using s bits of storage and answers membership queries with t bitprobes to memory. Then, $\binom{m}{n} \leq \max_{i \leq nt} \binom{2s}{i}$.*

This tradeoff result has an interesting consequence. Recall that the FKS hashing scheme is

a data structure for storing sets of size at most n from a universe of size m using $O(n \log m)$ bits, so that membership queries can be answered using $O(\log m)$ bitprobes. As a corollary of the tradeoff result, [3] show that the FKS scheme makes an optimal number of bitprobes, within a constant factor, for this amount of space.

Corollary 1 *Let $\epsilon > 0, c \geq 1$ be any constants. There is a constant $\delta > 0$ so that the following holds. Let $n \leq m^{1-\epsilon}$ and let a scheme for storing sets of size at most n of a universe of size m as data structures of at most $cn \log m$ bits be given. Then, any deterministic algorithm answering membership queries using this structure must make at least $\delta \log m$ bitprobes in the worst case.*

From Theorem 6, it also follows that any deterministic scheme that answers queries using t bitprobes must use space at least $\Omega\left(tm^{1/t}n^{1-1/t}\right)$ in the worst case. The final result shows the existence of schemes which almost match the lower bound.

Theorem 7 *1. There is a nonadaptive scheme that stores sets of size at most n from a universe of size m using $O\left(ntm^{\frac{2}{t+1}}\right)$ bits and answers queries using $2t+1$ bitprobes. This scheme is non-explicit.*

2. *There is an explicit adaptive scheme that stores sets of size at most n from a universe of size m using $O\left(m^{1/t}n \log m\right)$ bits and answers queries using $O(\log n + \log\log m) + t$ bitprobes.*

Power of Few Bitprobes

In this section, we highlight some of recent results for this problem subsequent to [3] and encourage the reader to read the corresponding references for more details. Most of these results focus on the power of deterministic schemes with a small number of bitprobes.

Let $S(m,n,t)$ denote the minimum number of bits of storage needed by a deterministic scheme that answers queries using t (adaptive) bitprobes. In [3], it was shown that $S(m,n,1) = m$ and $S(m,n,5) = o(m)$ for $n = o\left(m^{1/3}\right)$ (Theorem 7, Part 1). This leads us to a natural question: Is $S(m,n,t) = o(m)$ for $t = 2, 3$, and 4 and under what conditions on n?

Initial progress for the case $t = 2$ was made by [18] who considered the simplest case: $n = 2$. They showed that $S(m,2,2) = O\left(m^{2/3}\right)$. It was later shown in [19] that $S(m,2,2) = \Omega\left(m^{4/7}\right)$. The upper bound result of [18] was improved upon by the authors of [1] who showed that $S(m,n,2) = o(m)$ if $n = o(\log m)$. Interestingly, a matching lower bound was shown recently in [12]: $S(m,n,2) = o(m)$ only if $n = o(\log m)$.

Strong upper bounds were obtained by [1] for the case $t = 3$ and $t = 4$. They showed that $S(m,n,3) = o(m)$ whenever $n = o(m)$. They also showed that $S(m,n,4) = o(m)$ for $n = o(m)$ even if the four bitprobes are nonadaptive. Recently, it was shown in [14] that $S(m,2,3) \leq 7m^{2/5}$. This work focuses on explicit schemes for $n = 2$ and $t \geq 3$.

Finally, we end with two remarks. Our problem for the case $n = \Theta(m)$ has been studied by Viola [22]. A recent result of Chen, Grigorescu, and de Wolf [4] studies our problem in the presence of adversarial noise.

Applications

The results in [3] have interesting connections to questions in coding theory and communication complexity. In the framework of coding theory, the results in [3] can be viewed as constructing locally decodable source codes, analogous to the locally decodable channel codes of [13]. Theorems 1–4 can also be viewed as giving tight bounds for the following communication complexity problem (as pointed out in [15]): Alice gets $u \in \{1,\ldots,m\}$, Bob gets $S \subseteq \{1,\ldots,m\}$ of size at most n, and Alice sends a single message to Bob after which Bob announces whether $u \in S$. See [3] for further details.

Recommended Reading

1. Alon N and Feige U (2009) On the power of two, three and four probes. In: Proceedings of SODA'09, New York, pp 346–354

2. Brodnik A, Munro JI (1994) Membership in constant time and minimum space. In: Algorithms ESA'94: second annual European symposium, Utrecht. Lecture notes in computer science, vol 855, pp 72–81. Final version: Membership in constant time and almost-minimum space. SIAM J Comput 28(5):1627–1640 (1999)
3. Buhrman H, Miltersen PB, Radhakrishnan J, Venkatesh S (2002) Are bitvectors optimal? SIAM J Comput 31(6):1723–1744
4. Chen V, Grigorescu E, de Wolf R (2013) Error-correcting data structures. SIAM J Comput 42(1):84–111
5. Dyachkov AG, Rykov VV (1982) Bounds on the length of disjunctive codes. Problemy Peredachi Informatsii 18(3):7–13 [Russian]
6. Elias P, Flower RA (1975) The complexity of some simple retrieval problems. J Assoc Comput Mach 22:367–379
7. Erdős P, Frankl P, Füredi Z (1985) Families of finite sets in which no set is covered by the union of r others. Isr J Math 51:79–89
8. Fiat A, Naor M (1993) Implicit $O(1)$ probe search. SIAM J Comput 22:1–10
9. Fiat A, Naor M, Schmidt JP, Siegel A (1992) Non-oblivious hashing. J Assoc Comput Mach 31:764–782
10. Fredman ML, Komlós J, Szemerédi E (1984) Storing a sparse table with $O(1)$ worst case access time. J Assoc Comput Mach 31(3):538–544
11. Füredi Z (1996) On r-cover-free families. J Comb Theory Ser A 73:172–173
12. Garg M, Radhakrishnan J (2015) Set membership with a few bit probes. In: Proceedings of SODA'15, San Diego, pp 776–784
13. Katz J, Trevisan L (2000) On the efficiency of local decoding procedures for error-correcting codes. In: Proceedings of STOC'00, Portland, pp 80–86
14. Lewenstein M, Munro JI, Nicholson PK, Raman V (2014) Improved explicit data structures in the bit-probe model. In: Proceedings of ESA'14, Wroclaw, pp 630–641
15. Miltersen PB, Nisan N, Safra S, Wigderson A (1998) On data structures and asymmetric communication complexity. J Comput Syst Sci 57:37–49
16. Minsky M, Papert S (1969) Perceptrons. MIT, Cambridge
17. Pagh R (1999) Low redundancy in static dictionaries with $O(1)$ lookup time. In: Proceedings of ICALP '99, Prague. Lecture notes in computer science, vol 1644, pp 595–604
18. Radhakrishnan J, Raman V, Rao SS (2001) Explicit deterministic constructions for membership in the bitprobe model. In: Proceedings of ESA'01, Aarhus, pp 290–299
19. Radhakrishnan J, Shah S, Shannigrahi S (2010) Data structures for storing small sets in the bitprobe model. In: Proceedings of ESA'10, Liverpool, pp 159–170
20. Ruszinkó M (1984) On the upper bound of the size of r-cover-free families. J Comb Theory Ser A 66:302–310
21. Ta-Shma A (2002) Explicit one-probe storing schemes using universal extractors. Inf Process Lett 83(5):267–274
22. Viola E (2012) Bit-probe lower bounds for succinct data structures. SIAM J Comput 41(6):1593–1604
23. Yao ACC (1981) Should tables be sorted? J Assoc Comput Mach 28(3):615–628

Approximate Distance Oracles with Improved Query Time

Christian Wulff-Nilsen
Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

Keywords

Approximate distance oracle; Graphs; Query time; Shortest paths

Years and Authors of Summarized Original Work

2013; Wulff-Nilsen

Problem Definition

This problem is concerned with obtaining a compact data structure capable of efficiently reporting approximate shortest path distance queries in a given undirected edge-weighted graph $G = (V, E)$. If the query time is independent (or nearly independent) of the size of G, we refer to the data structure as an *approximate distance oracle* for G. For vertices u and v in G, we denote by $d_G(u, v)$ the shortest path distance between u and v in G. For a given *stretch* parameter $\delta \geq 1$, we call the oracle *δ-approximate* if for all vertices u and v in G, $d_G(u, v) \leq \tilde{d}_G(u, v) \leq \delta d_G(u, v)$, where $\tilde{d}_G(u, v)$ is the output of the query for u and v. Hence, we allow estimates to be stretched by a factor up to δ but not shrunk.

Key Results

A major result in the area of distance oracles is due to Thorup and Zwick [4]. They gave, for every integer $k \geq 1$, a $(2k-1)$-approximate distance oracle of size $O\left(kn^{1+1/k}\right)$ and query time $O(k)$, where n is the number of vertices of the graph. This is constant query time when k is constant. Corresponding approximate shortest paths can be reported in time proportional to their length. Mendel and Naor [3] asked the question of whether query time can be improved to a universal constant (independent also of k) while keeping both size and stretch small. They obtained $O\left(n^{1+1/k}\right)$ size and $O(1)$ query time at the cost of a constant-factor increase in stretch to $128k$. Unlike the oracle of Thorup and Zwick, Mendel and Naor's oracle is not path reporting.

In [5], it is shown how to improve the query time of Thorup-Zwick to $O(\log k)$ without increasing space or stretch. This is done while keeping essentially the same data structure but applying binary instead of linear search in so-called bunch structures that were introduced by Thorup and Zwick [4]; the formal definition will be given below. Furthermore, it is shown in [5] how to improve the stretch of the oracle of Mendel and Naor to $(2+\epsilon)k$ for an arbitrarily small constant $\epsilon > 0$ while keeping query time constant (bounded by $1/\epsilon$). This improvement is obtained without an increase in space except for large values of k close to $\log n$ (only values of k less than $\log n$ are interesting since the Mendel-Naor oracle has optimal $O(n)$ space and $O(1)$ query time for larger values). Below, we sketch the main ideas in the improvement of Thorup-Zwick and of Mendel-Naor, respectively.

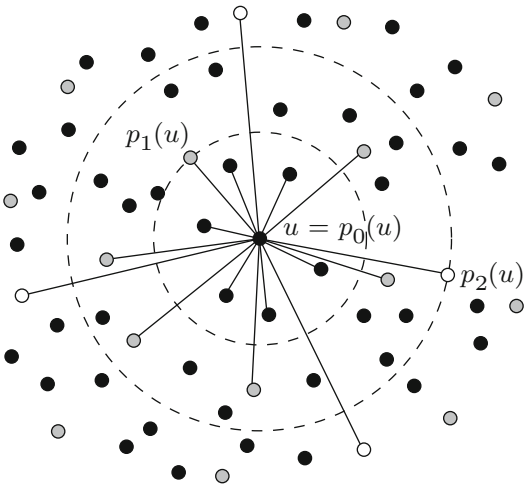
Oracle with $O(\log k)$ Query Time

The oracle of Thorup and Zwick keeps a hierarchy of sets of sampled vertices $V = A_0 \supseteq A_1 \supseteq A_2 \ldots \supseteq A_k = \emptyset$, where for $i = 1, \ldots, k-1$, A_i is obtained by picking each element of A_{i-1} independently with probability $n^{-1/k}$. Define $p_i(u)$ as the vertex in A_i closest to u. The oracle precomputes and stores for each vertex $u \in V$ the *bunch* B_u, defined as

$$B_u = \bigcup_{i=0}^{k-1} \{v \in A_i \setminus A_{i+1} | d_G(u,v) < d_G(u, p_{i+1}(u))\}.$$

See Fig. 1 for an illustration of a bunch. The distance $d_G(u,v)$ for each $v \in B_u$ is precomputed as well.

Now, to answer a query for a vertex pair (u,v), the oracle performs a linear search through bunches B_u and B_v. Pseudocode is given in Fig. 2. It is clear that query time is $O(k)$, and it can be shown that the estimate output in line 6 has stretch $2k-1$.



Approximate Distance Oracles with Improved Query Time, Fig. 1 A bunch B_u in a complete Euclidean graph with $k = 3$. *Black vertices* belong to A_0, *grey vertices* to A_1, and *white vertices* to A_2. Line segments connect u to vertices of B_u

```
Algorithm dist_k(u, v)
1. w ← p_0(u); j ←0
2. while w ∉ B_v
3.     j ← j + 1
4.     (u, v) ← (v, u)
5.     w ← p_j(u)
6. return d_G(w, u) + d_G(w, v)
```

Approximate Distance Oracles with Improved Query Time, Fig. 2 Answering a distance query, starting at sample level i

We can improve query time to $O(\log k)$ by instead doing binary search in the bunch structures. A crucial property of the Thorup-Zwick oracle is that every time the test in line 2 succeeds, $d_G(u, p_j(u))$ increases by at most $d_G(u, v)$, and this is sufficient to prove $2k - 1$ stretch. In particular, if the test succeeds two times in a row, $d_G(u, p_{j+2}(u)) - d_G(u, p_j(u)) \leq 2d_G(u, v)$, where j is even. If we can check that $d_G(u, p_{j'+2}(u)) - d_G(u, p_{j'}(u)) \leq 2d_G(u, v)$ for all smaller even indices j', we may start the query algorithm at index j instead of index 0. Since we would like to apply binary search, pick j to be (roughly) $k/2$. It suffices to check only one inequality, namely, the one with the largest value $d_G(u, p_{j'+2}(u)) - d_G(u, p_{j'}(u))$. Note that this value depends only on u and k, so we can precompute the index j' with this largest value. In the query phase, we can check in $O(1)$ time whether $d_G(u, p_{j'+2}(u)) - d_G(u, p_{j'}(u)) \leq 2d_G(u, v)$. If the test succeeds, we can start the query at j, and hence, we can recurse on indices between j and $k - 1$. Conversely, if the test fails, it means that the test in line 2 fails for either j' or $j' + 1$. Hence, the query algorithm of Thorup and Zwick terminates no later than at index $j' + 1$, and we can recurse on indices between 0 and $j' + 1$. In both cases, the number of remaining indices is reduced by a factor of at least 2. Since each recursive call takes $O(1)$ time, we thus achieve $O(\log k)$ query time.

Since the improved oracle is very similar to the Thorup-Zwick oracle, it is path reporting, i.e., it can report approximate paths in time proportional to their length.

Oracle with Constant Query Time

The second oracle in [5] can be viewed as a hybrid between the oracles of Thorup-Zwick and of Mendel-Naor. An initial estimate is obtained by querying the Mendel-Naor oracle. This estimate has stretch at most $128k$, and it is refined in subsequent iterations until the desired stretch $(2 + \epsilon)k$ is obtained. In each iteration, the current estimate is reduced by a small constant factor greater than 1 (depending on ϵ). Note that after a constant number of iterations, the estimate will be below the desired stretch, but it needs to be ensured that it is not below the shortest path distance.

In each iteration, the hybrid algorithm attempts to start the Thorup-Zwick query algorithm at a step corresponding to this estimate. If this can be achieved, only a constant number of steps of this query algorithm need to be executed before the desired stretch is obtained. Conversely, if the hybrid algorithm fails to access the Thorup-Zwick oracle in any iteration, then by a property of the bunch structures, it is shown that the current estimate is not below the shortest path distance. Hence, the desired stretch is again obtained.

An important property of the Mendel-Naor oracle needed above is that the set d_{MN} of all different values the oracle can output has size bounded by $O\left(n^{1+1/k}\right)$. This is used in the hybrid algorithm as follows. In a preprocessing step, values from d_{MN} are ordered in a list $\mathcal{L}$ together with additional values corresponding to the intermediate estimates that the hybrid algorithm can consider in an iteration. Updating the estimate in each iteration then corresponds to a linear traversal of part of $\mathcal{L}$. Next, each vertex p_i of each bunch structure B_u of the Thorup-Zwick oracle is associated with the value in the list closest to $d_G(u, p_i)$. For each element of $\mathcal{L}$, a hash table is kept for the bunch vertices associated with that element. It can be shown that this way of linking the oracle of Thorup-Zwick and Mendel-Naor achieves the desired.

Applications

The practical need for efficient algorithms to answer the shortest path (distance) queries in graphs has increased significantly over the years, in large part due to emerging GPS navigation technology and other route planning software. Classical algorithms like Dijkstra do not scale well as they may need to explore the entire graph just to answer a single query. As road maps are typically of considerable size, obtaining compact distance oracles has received a great deal of attention from the research community.

Open Problems

A widely believed girth conjecture of Erdős [2] implies that an oracle with stretch $2k-1$, size $O\left(n^{1+1/k}\right)$, and query time $O(1)$ would be optimal. Obtaining such an oracle (preferably one that is path reporting) is a main open problem in the area. Some progress has recently been made: Chechik [1] gives an oracle (not path reporting) with stretch $2k-1$, size $O\left(kn^{1+1/k}\right)$, and $O(1)$ query time.

Recommended Reading

1. Chechik S (2014) Approximate distance oracles with constant query time. In: STOC, New York, pp 654–663
2. Erdős P (1964) Extremal problems in graph theory. In: Theory of graphs and its applications (Proceedings of the symposium on smolenice, 1963). Czechoslovak Academy of Sciences, Prague, pp 29–36
3. Mendel M, Naor A (2007) Ramsey partitions and proximity data structures. J Eur Math Soc 9(2):253–275. See also FOCS'06
4. Thorup M, Zwick U (2005) Approximate distance oracles. J Assoc Comput Mach 52:1–24
5. Wulff-Nilsen C (2013) Approximate distance oracles with improved query time. In: Proceedings of the 24th ACM-SIAM symposium on discrete algorithms (SODA), New Orleans, pp 202–208

Approximate Matching

Ran Duan
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

Keywords

Approximation algorithms; Maximum matching

Years and Authors of Summarized Original Work

1973; Hopcroft, Karp
1980; Micali, Vazirani
2014; Duan, Pettie

Problem Definition

In graph theory, a *matching* in a graph is a set of edges without common vertices, while a *perfect matching* is one in which all vertices are associated with matching edges. In a graph $G = (V, E, w)$, where V is the set of vertices, E is the set of edges, and $w : E \to \mathbb{R}$ is the edge weight function, the *maximum matching* problem determines the matching M in G which maximizes $w(M) = \sum_{e \in M} w(e)$. Note that a maximum matching is not necessarily perfect. The *maximum cardinality matching (MCM)* problem means the maximum matching problem for $w(e) = 1$ for all edges. Otherwise, it is called the *maximum weight matching (MWM)*.

Algorithms for Exact MWM

Although the maximum matching problem has been studied for decades, the computational complexity of finding an optimal matching remains quite open. Most algorithms for graph matchings use the concept of augmenting paths. An alternating path (or cycle) is one whose edges alternate between M and $E \backslash M$. An alternating path P is *augmenting* if P begins and ends at free vertices, that is, $M \oplus P \stackrel{\text{def}}{=} (M \backslash P) \cup (P \backslash M)$ is a matching with cardinality $|M \oplus P| = |M| + 1$. Therefore, the basic algorithm finds the maximum cardinality matching by finding an augmenting path in the graph and adding it the matching each time, until no more augmenting paths exist. The running time for the basic algorithm will be $O(mn)$ where $m = |E|$ and $n = |V|$. The major improvement over this for bipartite graphs is the Hopcroft-Karp algorithm [10]. It finds a maximal set of vertex disjoint shortest augmenting paths in each step and shows that the length of shortest augmenting paths will increase each time. The running time of the Hopcroft-Karp algorithm is $O(m\sqrt{n})$. Its corresponding algorithm for general graphs is given by Micali and Vazirani [14].

For the maximum weight matching (MWM) and maximum weight perfect matching (MWPM), the most classical algorithm is the Hungarian algorithm [12] for bipartite graphs and the Edmonds algorithm for general graphs [6, 7].

For fast implementations, Gabow and Tarjan [8] gave bit-scaling algorithms for MWM in bipartite graphs running in $O(m\sqrt{n}\log(nN))$ time, where the edge weights are integers in $[-N,\ldots,N]$. Then, they also gave its corresponding algorithm for general graphs [9]. Extending [15], Sankowski [18] gave an $O(Nn^{\omega})$ MWM algorithm for bipartite graphs (here, $\omega \leq 2.373$ denotes the exponential of the complexity of fast matrix multiplication (FMM) [2, 20]), while Huang and Kavitha [11] obtained a similar time bound for general graphs. We can see these time complexities are still far from linear, which shows the importance of fast approximation algorithms.

Approximate Matching

Let a δ-MWM be a matching whose weight is at least a δ fraction of the maximum weight matching, where $0 < \delta \leq 1$, and let δ-MCM be defined analogously.

It is well known that the *greedy* algorithm – iteratively chooses the maximum weight edge not incident to previously chosen edges – produces a $\frac{1}{2}$-MWM. A straightforward implementation of this algorithm takes $O(m \log n)$ time. Preis [3,17] gave a $\frac{1}{2}$-MWM algorithm running in linear time. Vinkemeier and Hougardy [19] and Pettie and Sanders [16] proposed several $\left(\frac{2}{3}-\epsilon\right)$-MWM algorithms (see also [13]) running in $O(m \log \epsilon^{-1})$ time; each is based on iteratively improving a matching by identifying sets of short weight-augmenting paths and cycles.

Key Results

Approximate Maximum Cardinality Matching

In fact, the Hopcroft-Karp algorithm [10] for bipartite graphs and Micali-Vazirani [14] algorithm for general graphs both imply a $(1-\epsilon)$-MCM algorithm in $O(\epsilon^{-1}m)$ time. We can search for a maximal set of vertex disjoint shortest augmenting paths for k steps, and the matching obtained is a $\left(1-\frac{1}{k+1}\right)$-MCM.

Theorem 1 ([10, 14]) *In a general graph G, the $(1-\epsilon)$-MCM algorithm can be found in time $O(\epsilon^{-1}m)$.*

Approximate Maximum Weighted Matching

In 2014, Duan and Pettie [5] give the first $(1-\epsilon)$-MWM algorithm for arbitrary weighted graphs whose running time is linear. In particular, we show that such a matching can be found in $O(m\epsilon^{-1}\log\epsilon^{-1})$ time, improving a preliminary result of $O(m\epsilon^{-2}\log^3 n)$ running time by the authors in 2010 [4]. This result leaves little room for improvement. The main results are given in the following two theorems:

Theorem 2 ([5]) *In a general graph G with integer edge weights between $[0, N]$, a $(1-\epsilon)$-MWM can be computed in time $O(m\epsilon^{-1}\log N)$.*

Theorem 3 ([5]) *In a general graph G with real edge weights, a $(1-\epsilon)$-MWM can be computed in time $O(m\epsilon^{-1}\log\epsilon^{-1})$.*

Unlike previous algorithms of approximation ratios of $1/2$ [3, 17] or $2/3$ [16, 19], the new algorithm does not find weight-augmenting paths and cycles directly, but follows a primal-dual relaxation on the linear programming formulation of MWM. This relaxed complementary slackness approach relaxes the constraint of the dual variables by a small amount, so that the iterative process of the dual problem will converge to an approximate solution much more quickly. While it takes $O(\sqrt{n})$ iterations of augmenting to achieve a perfect matching, we proved that we only need $O(\log N/\epsilon)$ iterations to achieve a $(1-\epsilon)$-approximation. Also, we make the relaxation "dynamic" by shrinking the relaxation when the dual variables decrease by one half, so that finally the relaxation is at most ϵ times the edge weight on each matching edge and very small on each nonmatching edge, which gives an approximate solution.

Applications

Graph matching is a fundamental combinatorial problem that has a wide range of applications in

many fields, and it can also be building blocks of other algorithms, such as the Christofides algorithm [1] for approximate traveling salesman problem. The approximate algorithm for maximum weight matching described above has linear running time, much faster than the Hungarian algorithm [12] and Edmonds [6,7] algorithm. It is also much simpler than the Gabow-Tarjan scaling algorithms [8,9] of $\tilde{O}(m\sqrt{n})$ running time. Thus, it has a great impact both in theory and in real-world applications.

Cross-References

- ▸ Assignment Problem
- ▸ Maximum Matching

Recommended Reading

1. Christofides N, GROUP CMUPPMSR (1976) Worst-case analysis of a new heuristic for the travelling salesman problem. Defense Technical Information Center, http://books.google.de/books?id=2A7eygAACAAJ
2. Coppersmith D, Winograd T (1987) Matrix multiplication via arithmetic progressions. In: Proceedings of 19th ACM symposium on the theory of computing (STOC), New York, pp 1–6
3. Drake D, Hougardy S (2003) A simple approximation algorithm for the weighted matching problem. Inf Process Lett 85:211–213
4. Duan R, Pettie S (2010) Approximating maximum weight matching in near-linear time. In: Proceedings 51st IEEE symposium on foundations of computer science (FOCS), Las Vegas, pp 673–682
5. Duan R, Pettie S (2014) Linear-time approximation for maximum weight matching. J ACM 61(1):1:1–1:23. doi:10.1145/2529989, http://doi.acm.org/10.1145/2529989
6. Edmonds J (1965) Maximum matching and a polyhedron with 0, 1-vertices. J Res Nat Bur Stand Sect B 69B:125–130
7. Edmonds J (1965) Paths, trees, and flowers. Can J Math 17:449–467
8. Gabow HN, Tarjan RE (1989) Faster scaling algorithms for network problems. SIAM J Comput 18(5):1013–1036
9. Gabow HN, Tarjan RE (1991) Faster scaling algorithms for general graph-matching problems. J ACM 38(4):815–853
10. Hopcroft JE, Karp RM (1973) An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J Comput 2:225–231
11. Huang CC, Kavitha T (2012) Efficient algorithms for maximum weight matchings in general graphs with small edge weights. In: Proceedings of the twenty-third annual ACM-SIAM symposium on discrete algorithms, SODA'12, Kyoto. SIAM, pp 1400–1412. http://dl.acm.org/citation.cfm?id=2095116.2095226
12. Kuhn HW (1955) The Hungarian method for the assignment problem. Nav Res Logist Q 2:83–97
13. Mestre J (2006) Greedy in approximation algorithms. In: Proceedings of the 14th conference on annual European symposium, Zurich, vol 14. Springer, London, pp 528–539. doi:10.1007/11841036_48, http://portal.acm.org/citation.cfm?id=1276191.1276239
14. Micali S, Vazirani V (1980) An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In: Proceedings of 21st IEEE symposium on foundations of computer science (FOCS), Syracuse, pp 17–27
15. Mucha M, Sankowski P (2004) Maximum matchings via Gaussian elimination. In: Proceedings of 45th symposium on foundations of computer science (FOCS), Rome, pp 248–255
16. Pettie S, Sanders P (2004) A simpler linear time $2/3 - \epsilon$ approximation to maximum weight matching. Inf Process Lett 91(6):271–276
17. Preis R (1999) Linear time $1/2$-approximation algorithm for maximum weighted matching in general graphs. In: Proceedings of 16th symposium on theoretical aspects of computer science (STACS), Trier. LNCS, vol 1563, pp 259–269
18. Sankowski P (2006) Weighted bipartite matching in matrix multiplication time. In: Proceedings of 33rd international symposium on automata, languages, and programming (ICALP), Venice, pp 274–285
19. Vinkemeier DED, Hougardy S (2005) A linear-time approximation algorithm for weighted matchings in graphs. ACM Trans Algorithms 1(1):107–122
20. Williams VV (2012) Multiplying matrices faster than Coppersmith-Winograd. In: Proceedings of the 44th symposium on theory of computing, STOC'12, New York. ACM, New York, pp 887–898. doi:10.1145/2213977.2214056, http://doi.acm.org/10.1145/2213977.2214056

Approximate Regular Expression Matching

Gonzalo Navarro
Department of Computer Science, University of Chile, Santiago, Chile

Keywords

Regular expression matching allowing errors or differences

Years and Authors of Summarized Original Work

1995; Wu, Manber, Myers

Problem Definition

Given a *text string* $T = t_1 t_2 \ldots t_n$ and a *regular expression* R of length m denoting language, $\mathcal{L}(R)$ over an alphabet Σ of size σ, and given a *distance function* among strings d and a *threshold* k, the *approximate regular expression matching (AREM)* problem is to find all the text positions that finish a so-called approximate occurrence of R in T, that is, compute the set $\{j, \exists i, 1, \leq i \leq j, \exists P \in \mathcal{L}(R), d(P, t_i, \ldots, t_j) \leq k\}$ T, R, and k are given together, whereas the algorithm can be tailored for a specific d.

This entry focuses on the so-called weighted edit distance, which is the minimum sum of weights of a sequence of operations converting one string into the other. The operations are insertions, deletions, and substitutions of characters. The weights are positive real values associated to each operation and characters involved. The weight of deleting a character c is written $w(c \rightarrow \varepsilon)$, that of inserting c is written $w(\varepsilon \rightarrow c)$, and that of substituting c by $c \neq c'$ is written $w(c \rightarrow c')$. It is assumed $w(c \rightarrow c) = 0$ for all $c \in \Sigma \cup \varepsilon$ and the triangle inequality, that is, $w(x \rightarrow y) + w(y \rightarrow z) \geq w(x \rightarrow z)$ for any $x, y, z, \in \Sigma \cup \{\varepsilon\}$. As the distance may be asymmetric, it is also fixed that $d(A,B)$ is the cost of converting A into B. For simplicity and practicality, $m = o(n)$ is assumed in this entry.

Key Results

The most versatile solution to the problem [3] is based on a graph model of the distance computation process. Assume the regular expression R is converted into a nondeterministic finite automaton (NFA) with $O(m)$ states and transitions using Thompson's method [8]. Take this automaton as a directed graph $G(V,E)$ where edges are labeled by elements in $\Sigma \cup \{\varepsilon\}$. A directed and weighted graph $\mathcal{G}$ is built to solve the AREM problem. $\mathcal{G}$ is formed by putting $n+1$ copies of $G, G_0, G_1, \ldots, G_n$ and connecting them with weights so that the distance computation reduces to finding shortest paths in $\mathcal{G}$.

More formally, the nodes of $\mathcal{G}$ are $\{v_i, v \in V, 0 \leq i \leq n\}$, so that v_i is the copy of node $v \in V$ in graph G_i . For each edge $u \xrightarrow{c} v$ in $E, c \in \Sigma \cup \varepsilon$, the following edges are added to graph $\mathcal{G}$:

$$u_i \rightarrow v_i, \quad \text{with weight } w(c \rightarrow \varepsilon), \quad 0 \leq i \leq n.$$

$$u_i \rightarrow u_{i+1}, \quad \text{with weight } w(\varepsilon \rightarrow t_{i+1}), \quad 0 \leq i \leq n.$$

$$u_i \rightarrow v_{i+1}, \quad \text{with weight } w(c \rightarrow t_{i+1}), \quad 0 \leq i \leq n.$$

Assume for simplicity that G has initial state s and a unique final state f (this can always be arranged). As defined, the shortest path in $\mathcal{G}$ from s_0 to f_n gives the smallest distance between T and a string in $\mathcal{L}(R)$. In order to adapt the graph to the AREM problem, the weights of the edges between s_i and s_{i+1} are modified to be zero.

Then, the AREM problem is reduced to computing shortest paths. It is not hard to see that $\mathcal{G}$ can be topologically sorted so that all the paths to nodes in G_i are computed before all those to G_{i+1}. This way, it is not hard to solve this shortest path problem in $O(mn\log m)$ time and $O(m)$ space. Actually, if one restricts the problem to the particular case of *network expressions*, which are regular expressions without Kleene closure, then G has no loops and the shortest path computation can be done in $O(mn)$ time, and even better on average [2].

The most delicate part in achieving $O(mn)$ time for general regular expressions [3] is to prove that, given the types of loops that arise in the NFAs of regular expressions, it is possible to compute the distances correctly within each G_i by (a) computing them in a topological order of G_i without considering the *back edges* introduced by Kleene closures, (b) updating path costs by using the back edges once, and (c) updating path

costs once more in topological order ignoring back edges again.

Theorem 1 (Myers and Miller [3]) *There exists an $O(mn)$ worst-case time solution to the AREM problem under weighted edit distance.*

It is possible to do better when the weights are integer-valued, by exploiting the unit-cost RAM model through a four-Russian technique [10]. The idea is as follows. Take a small subexpression of R, which produces an NFA that will translate into a small subgraph of each G_i . At the time of propagating path costs within this automaton, there will be a counter associated to each node (telling the current shortest path from s_0). This counter can be reduced to a number in $[0, k + 1]$, where $k + 1$ means "more than k." If the small NFA has r states, $r\lceil \log_2(k + 2)\rceil$ bits are needed to fully describe the counters of the corresponding subgraph of G_i. Moreover, given an initial set of values for the counters, it is possible to precompute all the propagation that will occur within the same subgraph of G_i, in a table having $2^{r\lceil \log_2(k+2)\rceil}$ entries, one per possible configuration of counters. It is sufficient that $r < \alpha \log_{k+2} n$ for some $\alpha < 1$ to make the construction and storage cost of those tables $o(n)$. With the help of those tables, all the propagation within the subgraph can be carried out in constant time. Similarly, the propagation of costs to the same subgraph at G_{i+1} can also be precomputed in tables, as it depends only on the current counters in G_i and on text character t_{i+1}, for which there are only σ alternatives.

Now, take all the subtrees of R of maximum size not exceeding r and preprocess them with the technique above. Convert each such subtree into a leaf in R labeled by a special character a_A, associated to the corresponding small NFA A. Unless there are consecutive Kleene closures in R, which can be simplified as $R^{**} = R^*$, the size of R after this transformation is $O(m/r)$. Call R' the transformed regular expression. One essentially applies the technique of Theorem 1 to R', taking care of how to deal with the special leaves that correspond to small NFAs. Those leaves are converted by Thompson's construction into two nodes linked by an edge labeled a_A. When the path cost propagation process reaches the source node of an edge labeled a_A with cost c, one must update the counter of the initial state of NFA A to c (or $k + 1$ if $c > k$). One then uses the four-Russians table to do all the cost propagation within A in constant time and finally obtain, at the counter of the final state of A, the new value for the target node of the edge labeled a_A in the top-level NFA. Therefore, all the edges (normal and special) of the top-level NFA can be traversed in constant time, so the costs at G_i can be obtained in $O(mn/r)$ time using Theorem 1. Now one propagates the costs to G_{i+1}, using the four-Russians tables to obtain the current counter values of each subgraph A in G_{i+1}.

Theorem 2 (Wu et al. [10]) *There exists an $O(n + mn/\log_{k+2} n)$ worst-case time solution to the AREM problem under weighted edit distance if the weights are integer numbers.*

Applications

The problem has applications in computational biology, to find certain types of motifs in DNA and protein sequences. See [1] for a more detailed discussion. In particular, PROSITE patterns are limited regular expressions rather popular to search protein sequences. PROSITE patterns can be searched for with faster algorithms in practice [7]. The same occurs with other classes of complex patterns [6] and network expressions [2].

Open Problems

The worst-case complexity of the AREM problem is not fully understood. It is of course $\Omega(n)$, which has been achieved for $m\log(k + 2) = O(\log n)$, but it is not known how much can this be improved.

Experimental Results

Some experiments are reported in [5]. For small m and k, and assuming all the weights are 1

(except $w(c \rightarrow c) = 0$), bit-parallel algorithms of worst-case complexity $O(k\,n(m/\log n)^2)$ [4,9] are the fastest (the second is able to skip some text characters, depending on R). For arbitrary integer weights, the best choice is a more complex bit-parallel algorithm [5] or the four-Russians based one [10] for larger m and k. The original algorithm [3] is slower, but it is the only one supporting arbitrary weights.

URL to Code

A recent and powerful software package implementing AREM is *TRE* (http://laurikari.net/tre), which supports edit distance with different costs for each type of operation. Older packages offering efficient AREM are *agrep* [9] (https://github.com/Wikinaut/agrep) for simplified weight choices and *nrgrep* [4] (http://www.dcc.uchile.cl/~gnavarro/software).

Cross-References

- Approximate String Matching is a simplification of this problem, and the relation between graph $\mathcal{G}$ here and matrix C there should be apparent.
- Regular Expression Matching is the simplified case where exact matching with strings in $\mathcal{L}(R)$ is sought.

Recommended Reading

1. Gusfield D (1997) Algorithms on strings, trees and sequences. Cambridge University Press, Cambridge
2. Myers EW (1996) Approximate matching of network expressions with spacers. J Comput Biol 3(1):33–51
3. Myers EW, Miller W (1989) Approximate matching of regular expressions. Bull Math Biol 51:7–37
4. Navarro G (2001) Nr-grep: a fast and flexible pattern matching tool. Softw Pract Exp 31:1265–1312
5. Navarro G (2004) Approximate regular expression searching with arbitrary integer weights. Nord J Comput 11(4):356–373
6. Navarro G, Raffinot M (2002) Flexible pattern matching in strings – practical on-line search algorithms for texts and biological sequences. Cambridge University Press, Cambridge
7. Navarro G, Raffinot M (2003) Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. J Comput Biol 10(6):903–923
8. Thompson K (1968) Regular expression search algorithm. Commun ACM 11(6):419–422
9. Wu S, Manber U (1992) Fast text searching allowing errors. Commun ACM 35(10):83–91
10. Wu S, Manber U, Myers EW (1995) A subquadratic algorithm for approximate regular expression matching. J Algorithms 19(3):346–360

Approximate String Matching

Gonzalo Navarro
Department of Computer Science, University of Chile, Santiago, Chile

Keywords

Inexact string matching; Semiglobal or semilocal sequence similarity; String matching allowing errors or differences

Years and Authors of Summarized Original Work

1980; Sellers
1989; Landau, Vishkin
1999; Myers
2003; Crochemore, Landau, Ziv-Ukelson
2004; Fredriksson, Navarro

Problem Definition

Given a *text string* $T = t_1 t_2 \ldots t_n$ and a *pattern string* $P = p_1 p_2 \ldots p_m$, both being sequences over an alphabet Σ of size σ, and given a *distance function* among strings d and a *threshold* k, the *approximate string matching (ASM)* problem is to find all the text positions that finish the so-called approximate occurrence of P in T, that is, compute the set $\{j, \exists i, 1 \leq i \leq j, d(P, t_i \ldots t_j) \leq k\}$. In the sequential version of the problem, T, P,

and k are given together, whereas the algorithm can be tailored for a specific d.

The solutions to the problem vary widely depending on the distance d used. This entry focuses on a very popular one, called *Levenshtein distance* or *edit distance*, defined as the minimum number of character insertions, deletions, and substitutions necessary to convert one string into the other. It will also pay some attention to other common variants such as *indel distance*, where only insertions and deletions are permitted and is the dual of the *longest common subsequence lcs* ($d(A,B) = |A| + |B| - 2 \cdot lcs(A,B)$), and *Hamming distance*, where only substitutions are permitted.

A popular generalization of all the above is the *weighted edit distance*, where the operations are given positive real-valued weights and the distance is the minimum sum of weights of a sequence of operations converting one string into the other. The weight of deleting a character c is written $w(c \rightarrow \varepsilon)$, that of inserting c is written $w(\varepsilon \rightarrow c)$, and that of substituting c by $c' \neq c$ is written $w(c \rightarrow c')$. It is assumed $w(c \rightarrow c) = 0$ and the triangle inequality, that is, $w(x \rightarrow y) + w(y \rightarrow z) \geq w(x \rightarrow z)$ for any $x, y, z, \in \sum \cup \{\varepsilon\}$. As the distance may now be asymmetric, it is fixed that $d(A,B)$ is the cost of converting A into B. Of course, any result for weighted edit distance applies to edit, Hamming, and indel distances (collectively termed *unit-cost edit distances*) as well, but other reductions are not immediate.

Both worst- and average-case complexity are considered. For the latter, one assumes that pattern and text are randomly generated by choosing each character uniformly and independently from Σ. For simplicity and practicality, $m = o(n)$ is assumed in this entry.

Key Results

The most ancient and versatile solution to the problem [13] builds over the process of computing weighted edit distance. Let $A = a_1 a_2 \ldots a_m$ and $B = b_1 b_2 \ldots b_n$ be two strings. Let $C[0 \ldots m, 0 \ldots n]$ be a matrix such that $C[i,j] = d(a_1 \ldots a_i, b_1 \ldots b_j)$. Then, it holds $C[0,0] = 0$ and

$$C[i,j] = \min(C[i-1,j] + w(a_i \rightarrow \varepsilon), C[i,j-1] + w(\varepsilon \rightarrow b_j), C[i-1,j-1] + w(a_i \rightarrow b_j)),$$

where $C[i,-1] = C[-1,j] = \infty$ is assumed. This matrix is computed in $O(mn)$ time and $d(A,B) = C[m,n]$. In order to solve the approximate string matching problem, one takes $A = P$ and $B = T$ and sets $C[0,j] = 0$ for all j, so that the above formula is used only for $i > 0$.

Theorem 1 (Sellers 1980 [13]) *There exists an $O(mn)$ worst-case time solution to the ASM problem under weighted edit distance.*

The space is $O(m)$ if one realizes that C can be computed column-wise and only column $j-1$ is necessary to compute column j. As explained, this immediately implies that searching under unit-cost edit distances can be done in $O(mn)$ time as well. In those cases, it is quite easy to compute only part of matrix C so as to achieve $O(kn)$ average-time algorithms [14].

Yet, there exist algorithms with lower worst-case complexity for weighted edit distance. By applying a Ziv-Lempel parsing to P and T, it is possible to identify regions of matrix C corresponding to substrings of P and T that can be computed from other previous regions corresponding to similar substrings of P and T [5].

Theorem 2 (Crochemore et al. 2003 [5]) *There exists an $O(n + mn/\log_\sigma n)$ worst-case time solution to the ASM problem under weighted edit distance. Moreover, the time is $O(n + mnh/\log n)$, where $0 \leq h \leq \log \sigma$ is the entropy of T.*

This result is very general, also holding for computing weighted edit distance and local similarity (see section on "Applications"). For the case of edit distance and exploiting the unit-cost RAM model, it is possible to do better. On one hand, one can apply a four-Russian technique: All the possible blocks (submatrices of C) of

size $t \times t$, for $t = O(\log_\sigma n)$, are precomputed, and matrix C is computed block-wise [9]. On the other hand, one can represent each cell in matrix C using a constant number of bits (as it can differ from neighboring cells by ± 1) so as to store and process several cells at once in a single machine word [10]. This latter technique is called *bit-parallelism* and assumes a machine word of $\Theta(\log n)$ bits.

Theorem 3 (Masek and Paterson 1980 [9]; Myers 1999 [10]) *There exist* $O(n + mn/(\log_\sigma n)^2)$ *and* $O(n + mn/\log n)$ *worst-case time solutions to the ASM problem under edit distance.*

Both complexities are retained for indel distance, yet not for Hamming distance.

For unit-cost edit distances, the complexity can depend on k rather than on m, as $k < m$ for the problem to be nontrivial, and usually k is a small fraction of m (or even $k = o(m)$). A classic technique [8] computes matrix C by processing in constant time diagonals $C[i + d, j + d], 0 \leq d \leq s$, along which cell values do not change. This is possible by preprocessing the suffix trees of T and P for lowest common ancestor queries.

Theorem 4 (Landau and Vishkin 1989 [8]) *There exists an* $O(kn)$ *worst-case time solution to the ASM problem under unit-cost edit distances.*

Other solutions exist which are better for small k, achieving time $O(n(1 + k^4/m))$ [4]. For the case of Hamming distance, one can achieve improved results using convolutions [1].

Theorem 5 (Amir et al. 2004 [1]) *There exist* $O(n\sqrt{k \log k})$ *and* $O(n(1+k^3/m)\log k)$ *worst-case time solution to the ASM problem under Hamming distance.*

The last result for edit distance [4] achieves $O(n)$ time if k is small enough ($k = O(m^{1/4})$). It is also possible to achieve $O(n)$ time on unit-cost edit distances at the expense of an exponential additive term on m or k: The number of different columns in C is independent of n, so the transition from every possible column to the next can be precomputed as a finite-state machine.

Theorem 6 (Ukkonen 1985 [14]) *There exists an* $O(n+m\min(3^m, m(2m\sigma)^k))$ *worst-case time solution to the ASM problem under edit distance.*

Similar results apply for Hamming and indel distance, where the exponential term reduces slightly according to the particularities of the distances.

The worst-case complexity of the ASM problem is of course $\Omega(n)$, but it is not known if this can be attained for any m and k. Yet, the average-case complexity of the problem is known.

Theorem 7 (Chang and Marr 1994 [3]) *The average-case complexity of the ASM problem is* $\Theta(n(k + \log_\sigma m)/m)$ *under unit-cost edit distances.*

It is not hard to prove the lower bound as an extension to Yao's bound for exact string matching [15]. The lower bound was reached in the same paper [3], for $k/m < 1/3 - O\left(1/\sqrt{\sigma}\right)$. This was improved later to $k/m < 1/2 - O\left(1/\sqrt{\sigma}\right)$ [6] using a slightly different idea. The approach is to precompute the minimum distance to match every possible text substring (block) of length $O(\log_\sigma m)$ inside P. Then, a text window is scanned backwards, block-wise, adding up those minimum precomputed distances. If they exceed k before scanning all the window, then no occurrence of P with k errors can contain the scanned blocks, and the window can be safely slid over the scanned blocks, advancing in T. This is an example of a *filtration* algorithm, which discards most text areas and applies an ASM algorithm only over those areas that cannot be discarded.

Theorem 8 (Fredriksson and Navarro 2004 [6]) *There exists an optimal-on-average solution to the ASM problem under edit distance, for any* $k/m \leq \frac{1-e/\sqrt{\sigma}}{2-e/\sqrt{\sigma}} = 1/2 - O\left(1/\sqrt{\sigma}\right)$.

The result applies verbatim to indel distance. The same complexity is achieved for Hamming distance, yet the limit on k/m improves to $1 - 1/\sigma$. Note that, when the limit k/m is reached, the average complexity is already $\Theta(n)$. It is not clear up to which k/m limit could one achieve linear time on average.

Applications

The problem has many applications in computational biology (to compare DNA and protein sequences, recovering from experimental errors, so as to spot mutations or predict similarity of structure or function), text retrieval (to recover from spelling, typing, or automatic recognition errors), signal processing (to recover from transmission and distortion errors), and several others. See a survey [11] for a more detailed discussion.

Many extensions of the ASM problem exist, particularly in computational biology. For example, it is possible to substitute whole substrings by others (called *generalized edit distance*), swap characters in the strings (*string matching with swaps or transpositions*), reverse substrings (*reversal distance*), have variable costs for insertions/deletions when they are grouped (*similarity with gap penalties*), and look for any pair of substrings of both strings that are sufficiently similar (*local similarity*). See, for example, Gusfield's book [7], where many related problems are discussed.

Open Problems

The worst-case complexity of the problem is not fully understood. For unit-cost edit distances, it is $\Theta(n)$ if $m = O(\min(\log n, (\log_\sigma n)^2))$ or $k = O(\min(m^{1/4}, \log_{m\sigma} n))$. For weighted edit distance, the complexity is $\Theta(n)$ if $m = O(\log_\sigma n)$. It is also unknown up to which k/m value can one achieve $O(n)$ average time; up to now this has been achieved up to $k/m = 1/2 - O\left(1/\sqrt{\sigma}\right)$.

Experimental Results

A thorough survey on the subject [11] presents extensive experiments. Nowadays, the fastest algorithms for edit distance are in practice filtration algorithms [6, 12] combined with bit-parallel algorithms to verify the candidate areas [2, 10]. Those filtration algorithms work well for small enough k/m; otherwise, the bit-parallel algorithms should be used stand-alone. Filtration algorithms are easily extended to handle multiple patterns searched simultaneously.

URL to Code

Well-known packages offering efficient ASM are *agrep* (https://github.com/Wikinaut/agrep) and *nrgrep* (http://www.dcc.uchile.cl/~gnavarro/software).

Cross-References

- Approximate Regular Expression Matching is the more complex case where P can be a regular expression
- Indexed Approximate String Matching refers to the case where the text can be preprocessed
- String Matching is the simplified version where no errors are permitted

Recommended Reading

1. Amir A, Lewenstein M, Porat E (2004) Faster algorithms for string matching with k mismatches. J Algorithms 50(2):257–275
2. Baeza-Yates R, Navarro G (1999) Faster approximate string matching. Algorithmica 23(2):127–158
3. Chang W, Marr T (1994) Approximate string matching and local similarity. In: Proceedings of the 5th annual symposium on combinatorial pattern matching (CPM'94), Asilomar. LNCS, vol 807. Springer, Berlin, pp 259–273
4. Cole R, Hariharan R (2002) Approximate string matching: a simpler faster algorithm. SIAM J Comput 31(6):1761–1782
5. Crochemore M, Landau G, Ziv-Ukelson M (2003) A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM J Comput 32(6):1654–1673
6. Fredriksson K, Navarro G (2004) Average-optimal single and multiple approximate string matching. ACM J Exp Algorithms 9(1.4)
7. Gusfield D (1997) Algorithms on strings, trees and sequences. Cambridge University Press, Cambridge
8. Landau G, Vishkin U (1989) Fast parallel and serial approximate string matching. J Algorithms 10:157–169
9. Masek W, Paterson M (1980) A faster algorithm for computing string edit distances. J Comput Syst Sci 20:18–31

10. Myers G (1999) A fast bit-vector algorithm for approximate string matching based on dynamic progamming. J ACM 46(3):395–415
11. Navarro G (2001) A guided tour to approximate string matching. ACM Comput Surv 33(1):31–88
12. Navarro G, Baeza-Yates R (1999) Very fast and simple approximate string matching. Inf Proc Lett 72:65–70
13. Sellers P (1980) The theory and computation of evolutionary distances: pattern recognition. J Algorithms 1:359–373
14. Ukkonen E (1985) Finding approximate patterns in strings. J Algorithms 6:132–137
15. Yao A (1979) The complexity of pattern matching for a random string. SIAM J Comput 8:368–387

Approximate Tandem Repeats

Gregory Kucherov[1] and Dina Sokol[2]
[1]CNRS/LIGM, Université Paris-Est, Marne-la-Vallée, France
[2]Department of Computer and Information Science, Brooklyn College of CUNY, Brooklyn, NY, USA

Keywords

Approximate periodicities; Approximate repetitions

Years and Authors of Summarized Original Work

2001; Landau, Schmidt, Sokol
2003; Kolpakov, Kucherov

Problem Definition

Identification of periodic structures in words (variants of which are known as *tandem repeats, repetitions, powers*, or *runs*) is a fundamental algorithmic task (see entry ▸ Squares and Repetitions). In many practical applications, such as DNA sequence analysis, considered repetitions admit a certain variation between copies of the repeated pattern. In other words, repetitions under interest are *approximate tandem repeats* and not necessarily exact repeats only.

The simplest instance of an approximate tandem repeat is an *approximate square*. An approximate square in a word w is a subword uv, where u and v are within a given distance k according to some distance measure between words, such as Hamming distance or edit (also called Levenshtein) distance. There are several ways to define approximate tandem repeats as successions of approximate squares, i.e., to generalize to the approximate case the notion of arbitrary periodicity (see entry ▸ Squares and Repetitions). In this entry, we discuss three different definitions of approximate tandem repeats. The first two are built upon the Hamming distance measure, and the third one is built upon the edit distance.

Let $h(\cdot,\cdot)$ denote the Hamming distance between two words of equal length.

Definition 1 A word $r[1\ldots n]$ is called a K-*repetition* of period p, $p \le n/2$, iff $h(r[1\ldots n-p], r[p+1\ldots n]) \le K$.

Equivalently, a word $r[1\ldots n]$ is a K-repetition of period p, if the number of mismatches, i.e., the number of i such that $r[i] \neq r[i+p]$, is at most K. For example, *ataa atta ctta ct* is a 2-repetition of period 4. *atc atc atc atg atg atg atg atg* is a 1-repetition of period 3, but *atc atc atc att atc atc atc att* is not.

Definition 2 A word $r[1\ldots n]$ is called a K-*run*, of period p, $p \le n/2$, iff for every $i \in [1\ldots n-2p+1]$, we have $h(r[i\ldots i+p-1], r[i+p, i+2p-1]) \le K$.

A K-run can be seen as a sequence of approximate squares uv such that $|u| = |v| = p$ and u and v differ by at most K mismatches. The total number of mismatches in a K-run is not bounded.

Let $ed(\cdot,\cdot)$ denote the edit distance between two strings.

Definition 3 A word r is a K-*edit* repeat if it can be partitioned into consecutive subwords, $r = v' w_1 w_2 \ldots w_\ell v''$, $\ell \ge 2$, such that

$$ed(v', w_1') + \sum_{i=1}^{\ell-1} ed(w_i, w_{i+1}) + ed(w_\ell'', v'') \le K,$$

where w'_1 is some suffix of w_1 and w''_ℓ is some prefix of w_ℓ.

A K-edit repeat is a sequence of "evolving" copies of a pattern such that there are at most K insertions, deletions, and mismatches, overall, between all consecutive copies of the repeat. For example, the word $r = caagct\ cagct\ ccgct$ is a 2-edit repeat.

When looking for tandem repeats occurring in a word, it is natural to consider *maximal* repeats. Those are the repeats extended to the right and left as much as possible provided that the corresponding definition is still verified. Note that the notion of maximality applies to K-repetitions, to K-runs, and to K-edit repeats.

Under the Hamming distance, K-runs provide the weakest "reasonable" definition of approximate tandem repeats, since it requires that every square it contains cannot contain more than K mismatch errors, which seems to be a minimal reasonable requirement. On the other hand, K-repetition is the strongest such notion as it limits by K the *total* number of mismatches. This provides an additional justification that finding these two types of repeats is important as they "embrace" other intermediate types of repeats. Several intermediate definitions have been discussed in [9, Section 5].

In general, each K-repetition is a part of a K-run of the same period, and every K-run is the union of all K-repetitions it contains. Observe that a K-run can contain as many as a linear number of K-repetitions with the same period. For example, the word $(000\ 100)^n$ of length $6n$ is a 1-run of period 3, which contains $(2n - 1)$ 1-repetitions. In general, a K-run r contains $(s - K + 1)$ K-repetitions of the same period, where s is the number of mismatches in r.

Example 1 The following Fibonacci word contains three 3-runs of period 6. They are shown in regular font, in positions aligned with their occurrences. Two of them are identical and contain each four 3-repetitions, shown in italic for the first run only. The third run is a 3-repetition in itself.

```
010010  100100  101001  010010  010100  1001

 10010  100100  101001
 10010  100100  10
  0010  100100  101
    10  100100  10100
     0  100100  101001
                 1001  010010  010100  1
                       10      010100  1001
```

Key Results

Given a word w of length n and an integer K, it is possible to find all K-runs, K-repetitions, and K-edit repeats within w in the following time and space bounds:

- K **-runs** can be found in time $O(nK \log K + S)$ (S the output size) and working space $O(n)$ [9].
- K **-repetitions** can be found in time $O(nK \log K + S)$ and working space $O(n)$ [9].
- K **-edit repeats** can be found in time $O(nK \log K \log(n/K) + S)$ and working space $O(n + K^2)$ [14, 19].

All three algorithms are based on similar algorithmic tools that generalize corresponding techniques for the exact case [4, 15, 16] (see [10] for a systematic presentation). The first basic tool is a generalization of the *longest extension functions* [16] that, in the case of Hamming distance, can be exemplified as follows. Given a word w, we want to compute, for each position p and each $k \leq K$, the quantity $\max\{j \mid h(w[1 \ldots j], w[p \ldots p + j - 1]) \leq k\}$. Computing all those values can be done in time $O(nK)$ using a method based on the suffix tree and the computation of the *lowest common ancestor* described in [7].

The second tool is the Lempel-Ziv factorization used in the well-known compression method. Different variants of the Lempel-Ziv factorization of a word can be computed in linear time [7, 18].

The algorithm for computing K-repetitions from [9] can be seen as a direct generalization of

the algorithm for computing maximal repetitions (runs) in the exact case [8, 15]. Although based on the same basic tools and ideas, the algorithm [9] for computing K-runs is much more involved and uses a complex "bootstrapping" technique for assembling runs from smaller parts.

The algorithm for finding the K-edit repeats uses both the recursive framework and the idea of the *longest extension functions* of [16]. The longest common extensions, in this case, allow up to K edit operations. Efficient methods for computing these extensions are based upon a combination of the results of [12] and [13]. The K-edit repeats are derived by combining the longest common extensions computed in the forward direction with those computed in the reverse direction.

Applications

Tandemly repeated patterns in DNA sequences are involved in various biological functions and are used in different practical applications.

Tandem repeats are known to be involved in regulatory mechanisms, e.g., to act as binding sites for regulatory proteins. Tandem repeats have been shown to be associated with recombination hotspots in higher organisms. In bacteria, a correlation has been observed between certain tandem repeats and virulence and pathogenicity genes.

Tandem repeats are responsible for a number of inherited diseases, especially those involving the central nervous system. Fragile X syndrome, Kennedy disease, myotonic dystrophy, and Huntington's disease are among the diseases that have been associated with triplet repeats.

Examples of different genetic studies illustrating abovementioned biological roles of tandem repeats can be found in introductive sections of [1, 6, 11]. Even more than just genomic elements associated with various biological functions, tandem repeats have been established to be a fundamental mutational mechanism in genome evolution [17].

A major practical application of short tandem repeats is based on the interindividual variability in copy number of certain repeats occurring at a single locus. This feature makes tandem repeats a convenient tool for genetic profiling of individuals. The latter, in turn, is applied to pedigree analysis and establishing phylogenetic relationships between species, as well as to forensic medicine [3].

Open Problems

The definition of K-edit repeats is similar to that of K-repetitions (for the Hamming distance case). It would be interesting to consider other definitions of maximal repeats over the edit distance. For example, a definition similar to the K-run would allow up to K edits between each pair of neighboring periods in the repeat. Other possible definitions would allow K errors between *any* pair of copies of a repeat, or between *all pairs* of copies, or between some *consensus* and each copy.

In general, a *weighted* edit distance scheme is necessary for biological applications. Known algorithms for tandem repeats based on a weighted edit distance scheme are not feasible, and thus, only heuristics are currently used.

URL to Code

The algorithms described in this entry have been implemented for DNA sequences and are publicly available. The Hamming distance algorithms (K-runs and K-repetitions) are part of the *mreps* software package, available at http://mreps.univ-mlv.fr/ [11]. The K-edit repeat software, *TRED*, is available at http://tandem.sci.brooklyn.cuny.edu/ [19]. The implementations of the algorithms are coupled with postprocessing filters, necessary due to the nature of biological sequences.

In practice, software based on heuristic and statistical methods is largely used. Among them, TRF (http://tandem.bu.edu/trf/trf.html) [1] is the most popular program used by the bioinformatics community. Other programs include ATRHunter (http://bioinfo.cs.technion.ac.il/atrhunter/) [20]

and TandemSWAN (http://favorov.bioinfolab.net/swan/) [2]. STAR (http://atgc.lirmm.fr/star/) [5] is another software, based on an information-theoretic approach, for computing approximate tandem repeats of a prespecified pattern.

Cross-References

▸ Squares and Repetitions

Acknowledgments This work was supported in part by the National Science Foundation Grant DB&I 0542751.

Recommended Reading

1. Benson G (1999) Tandem repeats finder: a program to analyze DNA sequences. Nucleic Acids Res 27:573–580
2. Boeva VA, Régnier M, Makeev VJ (2004) SWAN: searching for highly divergent tandem repeats in DNA sequences with the evaluation of their statistical significance. In: Proceedings of JOBIM 2004, Montreal, p 40
3. Butler JM (2001) Forensic DNA typing: biology and technology behind STR markers. Academic Press, San Diego
4. Crochemore M (1983) Recherche linéaire d'un carré dans un mot. C R Acad Sci Paris Sér I Math 296:781–784
5. Delgrange O, Rivals E (2004) STAR – an algorithm to search for tandem approximate repeats. Bioinformatics 20:2812–2820
6. Gelfand Y, Rodriguez A, Benson G (2007) TRDB – the tandem repeats database. Nucleic Acids Res 35(suppl. 1):D80–D87
7. Gusfield D (1997) Algorithms on strings, trees, and sequences. Cambridge University Press, Cambridge/New York
8. Kolpakov R, Kucherov G (1999) Finding maximal repetitions in a word in linear time. In: 40th symposium foundations of computer science (FOCS), New York, pp 596–604. IEEE Computer Society Press
9. Kolpakov R, Kucherov G (2003) Finding approximate repetitions under Hamming distance. Theor Comput Sci 33(1):135–156
10. Kolpakov R, Kucherov G (2005) Identification of periodic structures in words. In: Berstel J, Perrin D (eds) Applied combinatorics on words. Encyclopedia of mathematics and its applications. Lothaire books, vol 104, pp 430–477. Cambridge University Press, Cambridge
11. Kolpakov R, Bana G, Kucherov G (2003) *mreps*: efficient and flexible detection of tandem repeats in DNA. Nucleic Acids Res 31(13):3672–3678
12. Landau GM, Vishkin U (1988) Fast string matching with k differences. J Comput Syst Sci 37(1):63–78
13. Landau GM, Myers EW, Schmidt JP (1998) Incremental string comparison. SIAM J Comput 27(2):557–582
14. Landau GM, Schmidt JP, Sokol D (2001) An algorithm for approximate tandem repeats. J Comput Biol 8:1–18
15. Main M (1989) Detecting leftmost maximal periodicities. Discret Appl Math 25:145–153
16. Main M, Lorentz R (1984) An $O(n \log n)$ algorithm for finding all repetitions in a string. J Algorithms 5(3):422–432
17. Messer PW, Arndt PF (2007) The majority of recent short DNA insertions in the human genome are tandem duplications. Mol Biol Evol 24(5):1190–1197
18. Rodeh M, Pratt V, Even S (1981) Linear algorithm for data compression via string matching. J Assoc Comput Mach 28(1):16–24
19. Sokol D, Benson G, Tojeira J (2006) Tandem repeats over the edit distance. Bioinformatics 23(2):e30–e35
20. Wexler Y, Yakhini Z, Kashi Y, Geiger D (2005) Finding approximate tandem repeats in genomic sequences. J Comput Biol 12(7):928–942

Approximating Fixation Probabilities in the Generalized Moran Process

George B. Mertzios
School of Engineering and Computing Sciences, Durham University, Durham, UK

Keywords

Approximation algorithm; Evolutionary dynamics; Fixation probability; Markov-chain Monte Carlo; Moran process

Years and Authors of Summarized Original Work

2014; Diaz, Goldberg, Mertzios, Richerby, Serna, Spirakis

Problem Definition

Population and evolutionary dynamics have been extensively studied, usually with the assumption that the evolving population has no spatial structure. One of the main models in this area is the Moran process [17]. The initial population

contains a single "mutant" with fitness $r > 0$, with all other individuals having fitness 1. At each step of the process, an individual is chosen at random, with probability proportional to its fitness. This individual reproduces, replacing a second individual, chosen uniformly at random, with a copy of itself.

Lieberman, Hauert, and Nowak introduced a generalization of the Moran process, where the members of the population are placed on the vertices of a connected graph which is, in general, directed [13, 19]. In this model, the initial population again consists of a single mutant of fitness $r > 0$ placed on a vertex chosen uniformly at random, with each other vertex occupied by a nonmutant with fitness 1. The individual that will reproduce is chosen as before, but now one of its neighbors is randomly selected for replacement, either uniformly or according to a weighting of the edges. The original Moran process can be recovered by taking the graph to be an unweighted clique.

Several similar models describing particle interactions have been studied previously, including the SIR and SIS epidemic models [8, Chapter 21], the voter model, the antivoter model, and the exclusion process [1, 7, 14]. Related models, such as the decreasing cascade model [12, 18], have been studied in the context of influence propagation in social networks and other models have been considered for dynamic monopolies [2]. However, these models do not consider different fitnesses for the individuals.

In general, the Moran process on a finite, connected, directed graph may end with all vertices occupied by mutants or with no vertex occupied by a mutant – these cases are referred to as *fixation* and *extinction*, respectively – or the process may continue forever. However, for undirected graphs and strongly connected digraphs, the process terminates almost surely, either at fixation or extinction. At the other extreme, in a directed graph with two sources, neither fixation nor extinction is possible. In this work we consider finite undirected graphs. The *fixation probability* for a mutant of fitness r in a graph G is the probability that fixation is reached and is denoted $f_{G,r}$.

Key Results

The fixation probability can be determined by standard Markov chain techniques. However, doing so for a general graph on n vertices requires solving a set of 2^n linear equations, which is not computationally feasible, even numerically. As a result, most prior work on computing fixation probabilities in the generalized Moran process has either been restricted to small graphs [6] or graph classes where a high degree of symmetry reduces the size of the set of equations – for example, paths, cycles, stars, and complete graphs [3–5] – or has concentrated on finding graph classes that either encourage or suppress the spread of the mutants [13, 16].

Because of the apparent intractability of exact computation, we turn to approximation. Using a potential function argument, we show that, with high probability, the Moran process on an undirected graph of order n reaches absorption (either fixation or extinction) within $\mathcal{O}(n^6)$ steps if $r = 1$ and $\mathcal{O}(n^4)$ and $\mathcal{O}(n^3)$ steps when $r > 1$ and $r < 1$, respectively. Taylor et al. [20] studied absorption times for variants of the generalized Moran process, but, in our setting, their results only apply to the process on regular graphs, where it is equivalent to a biased random walk on a line with absorbing barriers. The absorption time analysis of Broom et al. [3] is also restricted to cliques, cycles, and stars. In contrast to this earlier work, our results apply to all connected undirected graphs.

Our bound on the absorption time, along with polynomial upper and lower bounds for the fixation probability, allows the estimation of the fixation and extinction probabilities by Monte Carlo techniques. Specifically, we give a *fully polynomial randomized approximation scheme* (FPRAS) for these quantities. An FPRAS for a function $f(X)$ is a polynomial-time randomized algorithm g that, given input X and an error bound ε, satisfies $(1-\varepsilon)f(X) \leqslant g(X) \leqslant (1+\varepsilon)f(X)$ with probability at least $\frac{3}{4}$ and runs in time polynomial in the length of X and $\frac{1}{\varepsilon}$ [11].

For the case $r < 1$, there is no polynomial lower bound on the fixation probability so only the extinction probability can be approximated

by this technique. Note that, when $f \ll 1$, computing $1 - f$ to within a factor of $1 \pm \varepsilon$ does not imply computing f to within the same factor.

Bounding the Fixation Probability

In the next two lemmas, we provide polynomial upper and lower bounds for the fixation probability of an arbitrary undirected graph G. Note that the lower bound of Lemma 1 holds only for $r \geqslant 1$. Indeed, for example, the fixation probability of the complete graph K_n is given by $f_{K_n,r} = (1 - \frac{1}{r})/(1 - \frac{1}{r^n})$ [13, 19], which is exponentially small for any $r < 1$.

Lemma 1 *Let $G = (V, E)$ be an undirected graph with n vertices. Then $f_{G,r} \geqslant \frac{1}{n}$ for any $r \geqslant 1$.*

Lemma 2 *Let $G = (V, E)$ be an undirected graph with n vertices. Then $f_{G,r} \leqslant 1 - \frac{1}{n+r}$ for any $r > 0$.*

Bounding the Absorption Time

In this section, we show that the Moran process on a connected graph G of order n is expected to reach absorption in a polynomial number of steps. To do this, we use the potential function given by

$$\phi(S) = \sum_{x \in S} \frac{1}{\deg x}$$

for any state $S \subseteq V(G)$ and we write $\phi(G)$ for $\phi(V(G))$. Note that $1 < \phi(G) < n$ and that $\phi(\{x\}) = 1/\deg x \leqslant 1$ for any vertex $x \in V$.

First, we show that the potential strictly increases in expectation when $r > 1$ and strictly decreases in expectation when $r < 1$.

Lemma 3 *Let $(X_i)_{i \geqslant 0}$ be a Moran process on a graph $G = (V, E)$ and let $\emptyset \subset S \subset V$. If $r \geqslant 1$, then*

$$\mathbb{E}[\phi(X_{i+1}) - \phi(X_i) \mid X_i = S] \geqslant \left(1 - \frac{1}{r}\right) \cdot \frac{1}{n^3},$$

with equality if and only if $r = 1$. For $r < 1$,

$$\mathbb{E}[\phi(X_{i+1}) - \phi(X_i) \mid X_i = S] < \frac{r-1}{n^3}.$$

To bound the expected absorption time, we use martingale techniques. It is well known how to bound the expected absorption time using a potential function that decreases in expectation until absorption. This has been made explicit by Hajek [9] and we use the following formulation based on that of He and Yao [10]. The proof is essentially theirs but is modified to give a slightly stronger result.

Theorem 1 *Let $(Y_i)_{i \geqslant 0}$ be a Markov chain with state space Ω, where Y_0 is chosen from some set $I \subseteq \Omega$. If there are constants $k_1, k_2 > 0$ and a nonnegative function $\psi : \Omega \to \mathbb{R}$ such that*

- *$\psi(S) = 0$ for some $S \in \Omega$,*
- *$\psi(S) \leqslant k_1$ for all $S \in I$ and*
- *$\mathbb{E}[\psi(Y_i) - \psi(Y_{i+1}) \mid Y_i = S] \geqslant k_2$ for all $i \geqslant 0$ and all S with $\psi(S) > 0$,*

then $\mathbb{E}[\tau] \leqslant k_1/k_2$, where $\tau = \min\{i : \psi(Y_i) = 0\}$.

Using Theorem 1, we can prove the following upper bounds for the absorption time τ in the cases where $r < 1$ and $r > 1$, respectively.

Theorem 2 *Let $G = (V, E)$ be a graph of order n. For $r < 1$ and any $S \subseteq V$, the absorption time τ of the Moran process on G satisfies*

$$\mathbb{E}[\tau \mid X_0 = S] \leqslant \frac{1}{1-r} n^3 \phi(S).$$

Theorem 3 *Let $G = (V, E)$ be a graph of order n. For $r > 1$ and any $S \subseteq V$, the absorption time τ of the Moran process on G satisfies*

$$\begin{aligned}\mathbb{E}[\tau \mid X_0 = S] &\leqslant \frac{r}{r-1} n^3 \big(\phi(G) - \phi(S)\big) \\ &\leqslant \frac{r}{r-1} n^4.\end{aligned}$$

The case $r = 1$ is more complicated as Lemma 3 shows that the expectation is constant. However, this allows us to use standard martingale techniques and the proof of the following is partly adapted from the proof of Lemma 3.4 in [15].

Theorem 4 *The expected absorption time for the Moran process* $(X_i)_{i\geqslant 0}$ *with* $r = 1$ *on a graph* $G = (V, E)$ *is at most* $n^4(\phi(G)^2 - \mathbb{E}[\phi(X_0)^2])$.

Approximation Algorithms

We now have all the components needed to present our fully polynomial randomized approximation schemes (FPRAS) for the problem of computing the fixation probability of a graph, where $r \geqslant 1$, and for computing the extinction probability for all $r > 0$. In the following two theorems, we give algorithms whose running times are polynomial in n, r, and $\frac{1}{\varepsilon}$. For the algorithms to run in time polynomial in the length of the input and thus meet the definition of FPRAS, r must be encoded in unary.

Theorem 5 *There is an FPRAS for* MORAN FIXATION, *for* $r \geqslant 1$.

Proof (sketch) The algorithm is as follows. If $r = 1$ then we return $\frac{1}{n}$. Otherwise, we simulate the Moran process on G for $T = \lceil \frac{8r}{r-1} Nn^4 \rceil$ steps, $N = \lceil \frac{1}{2}\varepsilon^{-2} n^2 \ln 16 \rceil$ times and compute the proportion of simulations that reached fixation. If any simulation has not reached absorption (fixation or extinction) after T steps, we abort and immediately return an error value.

Note that each transition of the Moran process can be simulated in $\mathcal{O}(1)$ time. Maintaining arrays of the mutant and nonmutant vertices allows the reproducing vertex to be chosen in constant time, and storing a list of each vertex's neighbors allows the same for the vertex where the offspring is sent. Therefore, the total running time is $\mathcal{O}(NT)$ steps, which is polynomial in n and $\frac{1}{\varepsilon}$, as required.

For $i \in \{1, \ldots, N\}$, let $X_i = 1$ if the ith simulation of the Moran process reaches fixation and $X_i = 0$ otherwise. Assuming all simulation runs reach absorption, the output of the algorithm is $p = \frac{1}{N}\sum_i X_i$. □

Note that this technique fails for disadvantageous mutants ($r < 1$) because there is no analogue of Lemma 1 giving a polynomial lower bound on $f_{G,r}$. As such, an exponential number of simulations may be required to achieve the desired error probability. However, we can give an FPRAS for the extinction probability for all $r > 0$. Although the extinction probability is just $1 - f_{G,r}$, there is no contradiction because a small relative error in $1 - f_{G,r}$ does not translate into a small relative error in $f_{G,r}$ when $f_{G,r}$ is, itself, small.

Theorem 6 *There is an FPRAS for* MORAN EXTINCTION *for all* $r > 0$.

Proof (sketch) The algorithm and its correctness proof are essential as in the previous theorem. If $r = 1$, we return $1 - \frac{1}{n}$. Otherwise, we run $N = \lceil \frac{1}{2}\varepsilon^{-2}(r + n)^2 \ln 16 \rceil$ simulations of the Moran process on G for $T(r)$ steps each, where

$$T(r) = \begin{cases} \lceil \frac{8r}{r-1} Nn^4 \rceil & \text{if } r > 1 \\ \lceil \frac{8}{1-r} Nn^3 \rceil & \text{if } r < 1. \end{cases}$$

If any simulation has not reached absorption within $T(r)$ steps, we return an error value; otherwise, we return the proportion p of simulations that reached extinction. □

It remains open whether other techniques could lead to an FPRAS for MORAN FIXATION when $r < 1$.

Recommended Reading

1. Aldous DJ, Fill JA (2002) Reversible Markov chains and random walks on graphs. Monograph in preparation. Available at http://www.stat.berkeley.edu/aldous/RWG/book.html
2. Berger E (2001) Dynamic monopolies of constant size. J Comb Theory Ser B 83:191–200
3. Broom M, Hadjichrysanthou C, Rychtář J (2010) Evolutionary games on graphs and the speed of the evolutionary process. Proc R Soc A 466(2117):1327–1346
4. Broom M, Hadjichrysanthou C, Rychtář J (2010) Two results on evolutionary processes on general non-directed graphs. Proc R Soc A 466(2121):2795–2798
5. Broom M, Rychtář J (2008) An analysis of the fixation probability of a mutant on special classes of non-directed graphs. Proc R Soc A 464(2098):2609–2627
6. Broom M, Rychtář J, Stadler B (2009) Evolutionary dynamics on small order graphs. J Interdiscip Math 12:129–140
7. Durrett R (1988) Lecture notes on particle systems and percolation. Wadsworth Publishing Company, Pacific Grove
8. Easley D, Kleinberg J (2010) Networks, crowds, and markets: reasoning about a highly connected world. Cambridge University Press, New York

9. Hajek B (1982) Hitting-time and occupation-time bounds implied by drift analysis with applications. Adv Appl Probab 14(3):502–525
10. He J, Yao X (2001) Drift analysis and average time complexity of evolutionary algorithms. Artif Intell 127:57–85
11. Karp RM, Luby M (1983) Monte-Carlo algorithms for enumeration and reliability problems. In: Proceedings of 24th annual IEEE symposium on foundations of computer science (FOCS), Tucson, pp 56–64
12. Kempel D, Kleinberg J, Tardos E (2005) Influential nodes in a diffusion model for social networks. In: Proceedings of the 32nd international colloquium on automata, languages and programming (ICALP), Lisbon. Lecture notes in computer science, vol 3580, pp 1127–1138. Springer
13. Lieberman E, Hauert C, Nowak MA (2005) Evolutionary dynamics on graphs. Nature 433:312–316
14. Liggett TM (1985) Interacting particle systems. Springer, New York
15. Luby M, Randall D, Sinclair A (2001) Markov chain algorithms for planar lattice structures. SIAM J Comput 31(1):167–192
16. Mertzios GB, Nikoletseas S, Raptopoulos C, Spirakis PG (2013) Natural models for evolution on networks. Theor Comput Sci 477:76–95
17. Moran PAP (1958) Random processes in genetics. Proc Camb Philos Soc 54(1):60–71
18. Mossel E, Roch S (2007) On the submodularity of influence in social networks. In: Proceedings of the 39th annual ACM symposium on theory of computing (STOC), San Diego, pp 128–134
19. Nowak MA (2006) Evolutionary dynamics: exploring the equations of life. Harvard University Press, Cambridge
20. Taylor C, Iwasa Y, Nowak MA (2006) A symmetry of fixation times in evolutionary dynamics. J Theor Biol 243(2):245–251

Approximating Metric Spaces by Tree Metrics

Jittat Fakcharoenphol[1], Satish Rao[2], and Kunal Talwar[3]
[1]Department of Computer Engineering, Kasetsart University, Bangkok, Thailand
[2]Department of Computer Science, University of California, Berkeley, CA, USA
[3]Microsoft Research, Silicon Valley Campus, Mountain View, CA, USA

Keywords

Embedding general metrics into tree metrics

Years and Authors of Summarized Original Work

1996; Bartal, Fakcharoenphol, Rao, Talwar
2004; Bartal, Fakcharoenphol, Rao, Talwar

Problem Definition

This problem is to construct a random tree metric that probabilistically approximates a given arbitrary metric well. A solution to this problem is useful as the first step for numerous approximation algorithms because usually solving problems on trees is easier than on general graphs. It also finds applications in on-line and distributed computation.

It is known that tree metrics approximate general metrics badly, e.g., given a cycle C_n with n nodes, any tree metric approximating this graph metric has distortion $\Omega(n)$ [17]. However, Karp [15] noticed that a random spanning tree of C_n approximates the distances between any two nodes in C_n well in expectation. Alon, Karp, Peleg, and West [1] then proved a bound of $\exp(O(\sqrt{\log n \log\log n}))$ on an average distortion for approximating any graph metric with its spanning tree.

Bartal [2] formally defined the notion of probabilistic approximation.

Notations

A graph $G = (V, E)$ with an assignment of nonnegative weights to the edges of G defines a metric space (V, d_G) where for each pair $u, v \in V$, $d_G(u, v)$ is the shortest path distance between u and v in G. A metric (V, d) is a *tree metric* if there exists some tree $T = (V', E')$ such that $V \subseteq V'$ and for all $u, v \in V$, $d_T(u, v) = d(u, v)$. The metric (V, d) is also called a metric induced by T.

Given a metric (V, d), a distribution $\mathcal{D}$ over tree metrics over V *α-probabilistically approximates* d if every tree metric $d_T \in \mathcal{D}$, $d_T(u, v) \geq d(u, v)$ and $\mathrm{E}_{d_T \in \mathcal{D}}[d_T(u, v)] \leq \alpha \cdot d(u, v)$, for every $u, v \in V$. The quantity α is referred to as the *distortion* of the approximation.

Although the definition of probabilistic approximation uses a distribution $\mathcal{D}$ over tree metrics, one is interested in a procedure that constructs a random tree metric distributed according to $\mathcal{D}$, i.e., an algorithm that produces a random tree metric that probabilistically approximates a given metric. The problem can be formally stated as follows.

Problem (APPROX-TREE)
INPUT: a metric (V, d)
OUTPUT: a tree metric (V, d_T) sampled from a distribution $\mathcal{D}$ over tree metrics that α-probabilistically approximates (V, d).

Bartal then defined a class of tree metrics, called hierarchically well-separated trees (HST), as follows. A *k-hierarchically well-separated tree* (k-HST) is a rooted weighted tree satisfying two properties: the edge weight from any node to each of its children is the same, and the edge weights along any path from the root to a leaf are decreasing by a factor of at least k. These properties are important to many approximation algorithms.

Bartal showed that any metric on n points can be probabilistically approximated by a set of k-HST's with $O(\log^2 n)$ distortion, an improvement from $\exp(O(\sqrt{\log n \log\log n}))$ in [1]. Later Bartal [3], following the same approach as in Seymour's analysis on the Feedback Arc Set problem [18], improved the distortion down to $O(\log n \log\log n)$. Using a rounding procedure of Calinescu, Karloff, and Rabani [5], Fakcharoenphol, Rao, and Talwar [9] devised an algorithm that, in expectation, produces a tree with $O(\log n)$ distortion. This bound is tight up to a constant factor.

Key Results

A tree metric is closely related to graph decomposition. The randomized rounding procedure of Calinescu, Karloff, and Rabani [5] for the 0-extension problem decomposes a graph into pieces with bounded diameter, cutting each edge with probability proportional to its length and a ratio between the numbers of nodes at certain distances. Fakcharoenphol, Rao, and Talwar [9] used the CKR rounding procedure to decompose the graph recursively and obtained the following theorem.

Theorem 1 *Given an n-point metric (V, d), there exists a randomized algorithm, which runs in time $O(n^2)$, that samples a tree metric from the distribution $\mathcal{D}$ over tree metrics that $O(\log n)$-probabilistically approximates (V, d). The tree is also a 2-HST.*

The bound in Theorem 1 is tight, as Alon et al. [1] proved the bound of an $\Omega(\log n)$ distortion when (V, d) is induced by a grid graph. Also note that it is known (as folklore) that even embedding a line metric onto a 2-HST requires distortion $\Omega(\log n)$.

If the tree is required to be a k-HST, one can apply the result of Bartal, Charikar, and Raz [4] which states that any 2-HST can be $O(k/\log k)$-probabilistically approximated by k-HST, to obtain an expected distortion of $O(k \log n/\log k)$.

Finding a distribution of tree metrics that probabilistically approximates a given metric has a dual problem that is to find a single tree T with small average weighted stretch. More specifically, given weight c_{uv} on edges, find a tree metric d_T such that for all $u, v \in V d_T(u, v) \geq d(u, v)$ and $\sum_{u,v\in V} c_{uv} \cdot d_T(u, v) \leq \alpha \sum_{u,v\in V} c_{uv} \cdot d(u, v)$.

Charikar, Chekuri, Goel, Guha, and Plotkin [6] showed how to find a distribution of $O(n \log n)$ tree metrics that α-probabilistically approximates a given metric, provided that one can solve the dual problem. The algorithm in Theorem 1 can be derandomized by the method of conditional expectation to find the required tree metric with $\alpha = O(\log n)$. Another algorithm based on modified region growing techniques is presented in [9], and independently by Bartal.

Theorem 2 *Given an n-point metric (V, d), there exists a polynomial-time deterministic algorithm that finds a distribution $\mathcal{D}$ over $O(n \log n)$ tree metrics that $O(\log n)$-probabilistically approximates (V, d).*

Note that the tree output by the algorithm contains Steiner nodes, however Gupta [10] showed how to find another tree metric without Steiner nodes while preserving all distances within a constant factor.

Applications

Metric approximation by random trees has applications in on-line and distributed computation, since randomization works well against oblivious adversaries, and trees are easy to work with and maintain. Alon et al. [1] first used tree embedding to give a competitive algorithm for the k-server problem. Bartal [3] noted a few problems in his paper: metrical task system, distributed paging, distributed k-server problem, distributed queuing, and mobile user.

After the paper by Bartal in 1996, numerous applications in approximation algorithms have been found. Many approximation algorithms work for problems on tree metrics or HST metrics. By approximating general metrics with these metrics, one can turn them into algorithms for general metrics, while, usually, losing only a factor of $O(\log n)$ in the approximation factors. Sample problems are metric labeling, buy-at-bulk network design, and group Steiner trees. Recent applications include an approximation algorithm to the Unique Games [12], information network design [13], and oblivious network design [11].

The SIGACT News article [8] is a review of the metric approximation by tree metrics with more detailed discussion on developments and techniques. See also [3, 9], for other applications.

Open Problems

Given a metric induced by a graph, some application, e.g., solving a certain class of linear systems, does not only require a tree metric, but a tree metric induced by a spanning tree of the graph. Elkin, Emek, Spielman, and Teng [7] gave an algorithm for finding a spanning tree with average distortion of $O(\log^2 n \log\log n)$. It remains open if this bound is tight.

Cross-References

- ▸ Metrical Task Systems
- ▸ Sparse Graph Spanners

Recommended Reading

1. Alon N, Karp RM, Peleg D, West D (1995) A graph-theoretic game and its application to the k-server problem. SIAM J Comput 24:78–100
2. Bartal Y (1996) Probabilistic approximation of metric spaces and its algorithmic applications. In: FOCS '96: proceedings of the 37th annual symposium on foundations of computer science, Washington, DC. IEEE Computer Society, pp 184–193
3. Bartal Y (1998) On approximating arbitrary metrices by tree metrics. In: STOC '98: proceedings of the thirtieth annual ACM symposium on theory of computing. ACM Press, New York, pp 161–168
4. Bartal Y, Charikar M, Raz D (2001) Approximating min-sum k-clustering in metric spaces. In: STOC '01: proceedings of the thirtythird annual ACM symposium on theory of computing. ACM Press, New York, pp 11–20
5. Calinescu G, Karloff H, Rabani Y (2001) Approximation algorithms for the 0-extension problem. In: SODA '01: proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, pp 8–16
6. Charikar M, Chekuri C, Goel A, Guha S (1998) Rounding via trees: deterministic approximation algorithms for group Steiner trees and k-median. In: STOC '98: proceedings of the thirtieth annual ACM symposium on theory of computing. ACM Press, New York, pp 114–123
7. Elkin M, Emek Y, Spielman DA, Teng S-H (2005) Lower-stretch spanning trees. In: STOC '05: proceedings of the thirty-seventh annual ACM symposium on theory of computing. ACM Press, New York, pp 494–503
8. Fakcharoenphol J, Rao S, Talwar K (2004) Approximating metrics by tree metrics. SIGACT News 35:60–70
9. Fakcharoenphol J, Rao S, Talwar K (2004) A tight bound on approximating arbitrary metrics by tree metrics. J Comput Syst Sci 69:485–497
10. Gupta A (2001) Steiner points in tree metrics don't (really) help. In: SODA '01: proceedings of the twelfth annual ACM-SIAM symposium on discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, pp 220–227
11. Gupta A, Hajiaghayi MT, Räcke H (2006) Oblivious network design. In: SODA '06: proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithm. ACM Press, New York, pp 970–979

A

12. Gupta A, Talwar K (2006) Approximating unique games. In: SODA '06: proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithm, New York. ACM Press, New York, pp 99–106
13. Hayrapetyan A, Swamy C, Tardos É (2005) Network design for information networks. In: SODA '05: proceedings of the sixteenth annual ACM-SIAM symposium on discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, pp 933–942
14. Indyk P, Matousek J (2004) Low-distortion embeddings of finite metric spaces. In: Goodman JE, O'Rourke J (eds) Handbook of discrete and computational geometry. Chapman&Hall/CRC, Boca Raton, chap. 8
15. Karp R (1989) A 2*k*-competitive algorithm for the circle. Manuscript
16. Matousek J (2002) Lectures on discrete geometry. Springer, New York
17. Rabinovich Y, Raz R (1998) Lower bounds on the distortion of embedding finite metric spaces in graphs. Discret Comput Geom 19:79–94
18. Seymour PD (1995) Packing directed circuits fractionally. Combinatorica 15:281–288

Approximating the Diameter

Liam Roditty
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Keywords

Diameter; Graph algorithms; Shortest paths

Years and Authors of Summarized Original Work

1999; Aingworth, Chekuri, Indyk, Motwani
2013; Roditty, Vassilevska Williams
2014; Chechik, Larkin, Roditty, Schoenebeck, Tarjan, Vassilevska Williams

Problem Definition

The diameter of a graph is the largest distance between its vertices. Closely related to the diameter is the radius of the graph. The center of a graph is a vertex that minimizes the maximum distance to all other nodes, and the radius is the distance from the center to the node furthest from it. Being able to compute the diameter, center, and radius of a graph efficiently has become an increasingly important problem in the analysis of large networks [11]. For general weighted graphs the only known way to compute the exact diameter and radius is by solving the all-pairs shortest paths problem (APSP). Therefore, a natural question is whether it is possible to get faster diameter and radius algorithms by settling for an approximation. For a graph G with diameter D, a c-approximation of D is a value $\hat{D}$ such that $\hat{D} \in [D/c, D]$. The question is whether a c-approximation can be computed in sub-cubic time.

Key Results

For *sparse* directed or undirected unweighted graphs, the best-known algorithm (ignoring polylogarithmic factors) for APSP, diameter, and radius does breadth-first search (BFS) from every node and hence runs in $O(mn)$ time, where m is the number of edges in the graph. For dense directed *unweighted* graphs, it is possible to compute both the diameter and the radius using fast matrix multiplication (this is folklore; for a recent simple algorithm, see [5]), thus obtaining $\tilde{O}(n^{\omega})$ time algorithms, where $\omega < 2.38$ is the matrix multiplication exponent [4, 9, 10] and n is the number of nodes in the graph.

A 2-approximation for both the diameter and the radius of an undirected graph can be obtained in $O(m + n)$ time using BFS from an arbitrary node. For APSP, Dor et al. [6] show that any $(2 - \epsilon)$-approximation algorithm in unweighted undirected graphs running in $T(n)$ time would imply an $O(T(n))$ time algorithm for Boolean matrix multiplication (BMM). Hence a priori it could be that $(2 - \epsilon)$-approximating the diameter and radius of a graph may also require solving BMM.

Aingworth et al. [1] showed that this is not the case by presenting a sub-cubic $(2 - \epsilon)$-approximation algorithm for the diameter in both directed and undirected graphs that does not use fast matrix multiplication. Their algorithm

computes in $\tilde{O}(m\sqrt{n} + n^2)$ time an estimate $\hat{D}$ such that $\hat{D} \in [\lfloor 2D/3 \rfloor, D]$. Berman and Kasiviswanathan [2] showed that for the radius problem the approach of Aingworth et al. can be used to obtain in $\tilde{O}(m\sqrt{n} + n^2)$ time an estimate $\hat{r}$ that satisfies $r \in [\hat{r}, 3/2r]$, where r is the radius of the graph. For weighted graphs the algorithm of Aingworth et al. [1] guarantees that the estimate $\hat{D}$ satisfies $\hat{D} \in [\lfloor \frac{2}{3} \cdot D \rfloor - (M-1), D]$, where M is the maximum edge weight in the graph.

Roditty and Vassilevska Williams [8] gave a Las Vegas algorithm running in expected $\tilde{O}(m\sqrt{n})$ time that has the same approximation guarantee as Aingworth et al. for the diameter and the radius. They also showed that obtaining a $(\frac{3}{2} - \epsilon)$-approximation algorithm running in $O(n^{2-\delta})$ time in sparse undirected and unweighted graphs for constant $\epsilon, \delta > 0$ would be difficult, as it would imply a fast algorithm for CNF Satisfiability, violating the widely believed Strong Exponential Time Hypothesis of Impagliazzo, Paturi, and Zane [7].

Chechik et al. [3] showed that it is possible to remove the additive error while still keeping the running time (in terms of n) subquadratic for sparse graphs. They present two *deterministic* algorithms with $\frac{3}{2}$-approximation for the diameter, one running in $\tilde{O}(m^{\frac{3}{2}})$ time and one running in $\tilde{O}(mn^{\frac{3}{2}})$ time.

Open Problems

The main open problem is to understand the relation between the diameter computation and the APSP problem. Is there a truly sub-cubic time algorithm for computing the exact diameter or can we show sub-cubic equivalence between the exact diameter computation and APSP problem?

Another important open problem is to find an algorithm that distinguishes between graphs of diameter two to graphs of diameter three in sub-cubic time. Alternatively, can we show that it is sub-cubic equivalent to the problem of exact diameter?

Recommended Reading

1. Aingworth D, Chekuri C, Indyk P, Motwani R (1999) Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM J Comput 28(4):1167–1181
2. Berman P, Kasiviswanathan SP (2007) Faster approximation of distances in graphs. In: Proceedings of the WADS, Halifax, pp 541–552
3. Chechik S, Larkin D, Roditty L, Schoenebeck G, Tarjan RE, Williams VV (2014) Better approximation algorithms for the graph diameter. In: SODA, Portland, pp 1041–1052
4. Coppersmith D, Winograd S (1990) Matrix multiplication via arithmetic progressions. J Symb Comput 9(3):251–280
5. Cygan M, Gabow HN, Sankowski P (2012) Algorithmic applications of Baur-strassen's theorem: shortest cycles, diameter and matchings. In: Proceedings of the FOCS, New Brunswick
6. Dor D, Halperin S, Zwick U (2000) All-pairs almost shortest paths. SIAM J Comput 29(5):1740–1759
7. Impagliazzo R, Paturi R, Zane F (2001) Which problems have strongly exponential complexity? J Comput Syst Sci 63(4):512–530
8. Roditty L, Vassilevska Williams V (2013) Fast approximation algorithms for the diameter and radius of sparse graphs. In: Proceedings of the 45th annual ACM symposium on theory of computing, STOC '13, Palo Alto. ACM, New York, pp 515–524. doi:10.1145/2488608.2488673, http://doi.acm.org/10.1145/2488608.2488673
9. Stothers A (2010) On the complexity of matrix multiplication. PhD thesis, University of Edinburgh
10. Vassilevska Williams V (2012, to appear) Multiplying matrices faster than Coppersmith-Winograd. In: Proceedings of the STOC, New York
11. Watts DJ, Strogatz SH (1998) Collective dynamics of 'small-world' networks. Nature 393:440–442

Approximating the Partition Function of Two-Spin Systems

Pinyan Lu[1] and Yitong Yin[2]
[1]Microsoft Research Asia, Shanghai, China
[2]Nanjing University, Jiangsu, Nanjing, Gulou, China

Keywords

Approximate counting; Partition function; Two-state spin systems

Years and Authors of Summarized Original Work

1993; Jerrum, Sinclair
2003; Goldberg, Jerrum, Paterson
2006; Weitz
2012; Sinclair, Srivastava, Thurley
2013; Li, Lu, Yin
2015; Sinclair, Srivastava, Štefankovič, Yin

Problem Definition

Spin systems are well-studied objects in statistical physics and applied probability. An instance of a spin system is an undirected graph $G = (V, E)$ of n vertices. A *configuration* of a two-state spin system, or simply just *two-spin system* on G, is an assignment $\sigma : V \to \{0, 1\}$ of two *spin states* "0" and "1" (sometimes called "–" and "+" or seen as two colors) to the vertices of G. Let $\mathbf{A} = \begin{bmatrix} A_{0,0} & A_{0,1} \\ A_{1,0} & A_{1,1} \end{bmatrix}$ be a nonnegative symmetric matrix which specifies the local interactions between adjacent vertices and $\mathbf{b} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$ a nonnegative vector which specifies preferences of individual vertices over the two spin states. For each configuration $\sigma \in \{0, 1\}^V$, its weight is then given by the following product:

$$w(\sigma) = \prod_{\{u,v\} \in E} A_{\sigma(u),\sigma(v)} \prod_{v \in V} b_{\sigma(v)}.$$

The *partition function* $Z_{\mathbf{A},\mathbf{b}}(G)$ of a two-spin system on G is defined to be the following exponential summation over all possible configurations:

$$Z_{\mathbf{A},\mathbf{b}}(G) = \sum_{\sigma \in \{0,1\}^V} w(\sigma).$$

Up to normalization, $\mathbf{A}$ and $\mathbf{b}$ can be described by three parameters, so that one can assume that $\mathbf{A} = \begin{bmatrix} \beta & 1 \\ 1 & \gamma \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} \lambda \\ 1 \end{bmatrix}$, where $\beta, \gamma \geq 0$ are the *edge activities* and $\lambda > 0$ is the *external field*. Since the roles of the two spin states are symmetric, it can be further assumed that $\beta \leq \gamma$ without loss of generality. Therefore, a two-spin system is completely specified by the three parameters (β, γ, λ) where it holds that $0 \leq \beta \leq \gamma$ and $\lambda > 0$. The resulting partition function is written as $Z_{(\beta,\gamma,\lambda)}(G) = Z_{\mathbf{A},\mathbf{b}}(G)$ and as $Z(G)$ for short if the parameters are clear from the context.

The two-spin systems are classified according to their parameters into two families with distinct physical and computational properties: the *ferromagnetic* two-spin systems ($\beta\gamma > 1$) in which neighbors favor agreeing spin states and the *antiferromagnetic* two-spin systems ($\beta\gamma < 1$) in which neighbors favor disagreeing spin states. Two-spin systems with $\beta\gamma = 1$ are trivial in both physical and computational senses and thus are usually not considered. The model of two-spin systems covers some of the most extensively studied statistical physics models as special cases, as well as being accepted in computer science as a framework for counting problems, for examples:

- When $\beta = 0$, $\gamma = 1$, and $\lambda = 1$, the $Z_{(\beta,\gamma,\lambda)}(G)$ gives the number of independent sets (or vertex covers) of G.
- When $\beta = 0$ and $\gamma = 1$, the $Z_{(\beta,\gamma,\lambda)}(G)$ is the partition function of the hardcore model with fugacity λ on G.
- When $\beta = \gamma$, the $Z_{(\beta,\gamma,\lambda)}(G)$ is the partition function of the Ising model with edge activity β and external field λ on G.

Given a set of parameters (β, γ, λ), the computational problem Two-Spin(β, γ, λ) is the problem of computing the value of the partition function $Z_{(\beta,\gamma,\lambda)}(G)$ when the graph G is given as input. This problem is known to be #P-hard except for the trivial cases where $\beta\gamma = 1$ or $\beta = \gamma = 0$ [1]. Therefore, the main focus here is the efficient approximation algorithms for Two-Spin(β, γ, λ). Formally, a fully polynomial-time approximation scheme (FPTAS) is an algorithm which takes G and any $\varepsilon > 0$ as input and outputs a number $\hat{Z}$ satisfying $Z(G)\exp(-\varepsilon) \leq \hat{Z} \leq Z(G)\exp(\varepsilon)$ within time polynomial in n and $1/\varepsilon$; and a fully

polynomial-time randomized approximation scheme (FPRAS) is its randomized relaxation in which randomness is allowed and the above accuracy of approximation is required to be satisfied with high probability.

For many important two-spin systems (e.g., independent sets, antiferromagnetic Ising model), it is NP-hard to approximate the partition function on graphs of unbounded degrees. In these cases, the problem is further refined to consider the approximation algorithms for TWO-SPIN(β, γ, λ) on graphs with *bounded maximum degree*. In addition, in order to study the approximation algorithms on graphs which has bounded average degree or on special classes of lattice graphs, the approximation of partition function is studied on classes of graphs with bounded *connective constant*, a natural and well-studied notion of average degree originated from statistical physics.

Therefore, the main problem of interest is to characterize the regimes of parameters (β, γ, λ) for which there exist efficient approximation algorithms for TWO-SPIN(β, γ, λ) on classes of graphs with bounded maximum degree $\Delta_{\max}$, or on classes of graphs with bounded connective constant Δ, or on all graphs.

Key Results

Given a two-spin system on graph $G = (V, E)$, a natural probability distribution μ over all configurations $\sigma \in \{0, 1\}^V$, called the *Gibbs measure*, can be defined by $\mu(\sigma) = \frac{w(\sigma)}{Z(G)}$, where $w(\sigma) = \prod_{\{u,v\}\in E} A_{\sigma_u,\sigma_v} \prod_{v\in V} b_{\sigma_v}$ is the weight of σ and the normalizing factor $Z(G)$ is the partition function.

The Gibbs measure defines a marginal distribution at each vertex. Suppose that a configuration σ is sampled according to the Gibbs measure μ. Let p_v denote the probability of vertex v having spin state "0" in σ; and for a fixed configuration $\tau_\Lambda \in \{0, 1\}^\Lambda$ partially specified over vertices in $\Lambda \subset V$, let $p_v^{\tau_\Lambda}$ denote the probability of vertex v having spin state "0" conditioning on that the configuration of vertices in Λ in σ is as specified by τ_Λ.

The marginal probability plays a key role in computing the partition function. Indeed, the marginal probability $p_v^{\tau_\Lambda}$ itself is a quantity of main interest in many applications such as probabilistic inference. In addition, due to the standard procedure of self-reduction, an FPTAS for the partition function $Z(G)$ can be obtained if the value of $p_v^{\tau_\Lambda}$ can be approximately computed with an additive error ε in time polynomial in both n and $1/\varepsilon$. This reduces the problem of approximating the partition function (with multiplicative errors) to approximating the marginal probability (with additive errors), which is achieved either by rapidly mixing random walks or by recursions exhibiting a decay of correlation.

Ferromagnetic Two-Spin Systems

For the ferromagnetic case, the problem TWO-SPIN(β, γ, λ) is considered for $\beta\gamma > 1$ and without loss of generality for $\beta \le \gamma$.

In a seminal work [3], Jerrum and Sinclair gave an FPRAS for approximately computing the partition function of the ferromagnetic Ising model, which is the TWO-SPIN(β, γ, λ) problem with $\beta = \gamma > 1$.

The algorithm uses the Markov chain Monte Carlo (MCMC) method; however very interestingly, it does not directly apply the random walk over configurations of two-spin system since such random walk might have a slow mixing time. Instead, it first transforms the two-spin system into configurations of the so-called "subgraphs world": each such configuration is a subgraph of G. A random walk over the subgraph configurations is applied and proved to be rapidly mixing for computing the new partition function defined over subgraphs, which is shown to be equal to the partition function $Z(G)$ of the two-spin system. This equivalence is due to that this transformation between the "spins world" and the "subgraphs world" is actually a holographic transformation, which is guaranteed to preserve the value of the partition function.

The result of [3] can be stated as the following theorem.

Theorem 1 *If $\beta = \gamma > 1$ and $\lambda > 0$, then there is an FPRAS for* TWO-SPIN(β, γ, λ).

The algorithm actually works for a stronger setting where the external fields are local (vertices have different external fields) as long as the external fields are homogeneous (all have the same preference over spin states).

For the two-spin system with general β and γ, one can translate it to the Ising model where $\beta = \gamma$ by delegating the effect of the general β, γ to the degree-dependent effective external fields. This extends the FPRAS for the ferromagnetic Ising model to certain regime of ferromagnetic two-spin systems, stated as follows.

Theorem 2 ([2, 6]) *If $\beta < \gamma$, $\beta\gamma > 1$, and $\lambda \leq \gamma/\beta$, then there is an FPRAS for* TWO-SPIN(β, γ, λ).

If one is restricted to the deterministic algorithms for approximating the partition function, then a deterministic FPTAS is known for a strictly smaller regime, implicitly stated in the following theorem.

Theorem 3 ([7]) *There is a continuous monotonically increasing function $\Gamma(\gamma)$ defined on $[1, +\infty)$ satisfying (1) $\Gamma(1) = 1$, (2) $1 < \Gamma(\gamma) < \gamma$ for all $\gamma > 1$, and (3) $\lim_{\gamma \to +\infty} \frac{\Gamma(\gamma)}{\gamma} = 1$, such that there is an FPTAS for* TWO-SPIN(β, γ, λ) *if $\beta\gamma > 1$, $\beta \leq \Gamma(\gamma)$, and $\lambda \leq 1$.*

This deterministic FPTAS uses the same holographic transformation from two-spin systems to the "subgraphs world" as in [3], and it approximately computes the marginal probability defined in the subgraphs world by a recursion. The accuracy of the approximation is guaranteed by the decay of correlation. This technique is more extensively and successfully used for the antiferromagnetic two-spin systems.

On the other hand, assuming certain complexity assumptions, it is unlikely that for every ferromagnetic two-spin system its partition function is easy to approximate.

Theorem 4 ([6]) *For any $\beta < \gamma$ with $\beta\gamma > 1$, there is a λ_0 such that* TWO-SPIN(β, γ, λ) *is* #BIS*-hard for all $\lambda \geq \lambda_0$.*

Antiferromagnetic Two-Spin Systems

For the antiferromagnetic case, the problem TWO-SPIN(β, γ, λ) is considered for $\beta\gamma < 1$ and without loss of generality for $\beta \leq \gamma$.

In [2], a heatbath random walk over spin configurations is applied to obtain an FPRAS for TWO-SPIN(β, γ, λ) for a regime of antiferromagnetic two-spin systems.

The regime of antiferromagnetic two-spin systems whose partition function is efficiently approximable is characterized by the *uniqueness condition*.

Given parameters (β, γ, λ) and $d \geq 1$, the *tree recursion* $f(x)$ is given by

$$f(x) = \lambda \left(\frac{\beta x + 1}{x + \gamma} \right)^d. \tag{1}$$

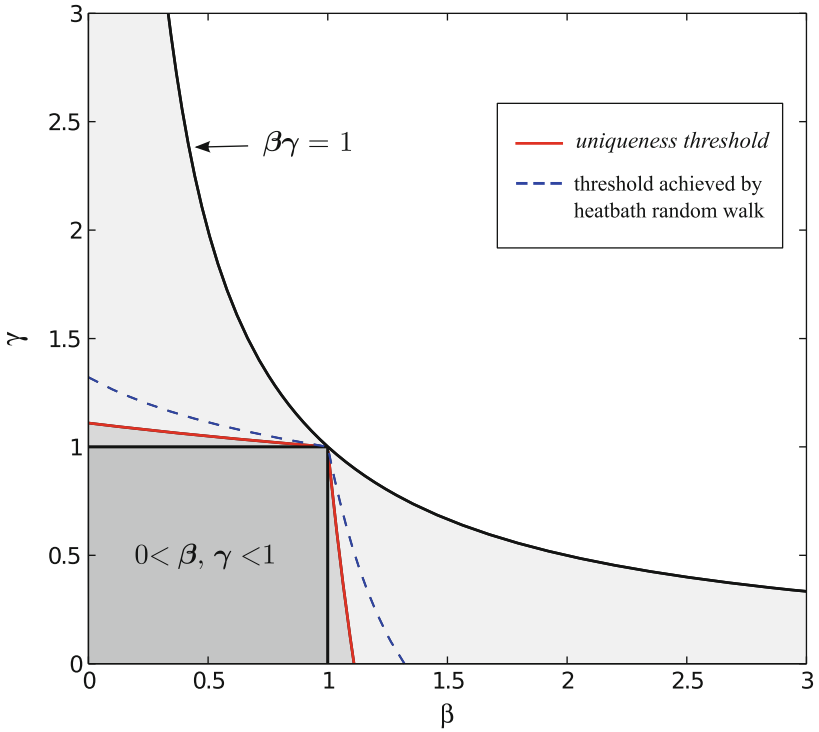
For antiferromagnetic (β, γ, λ), the function $f(x)$ is decreasing in x; thus, there is a unique positive fixed point $\hat{x}$ satisfying $\hat{x} = f(\hat{x})$. Consider the absolute derivative of $f(x)$ at the fixed point:

$$\left| f'(\hat{x}) \right| = \frac{d(1 - \beta\gamma)\hat{x}}{(\beta\hat{x} + 1)(\hat{x} + \gamma)}.$$

Definition 1 Let $0 \leq \beta \leq \gamma$, $\beta\gamma < 1$, and $d \geq 1$. The *uniqueness condition* UNIQUE$(\beta, \gamma, \lambda, d)$ is satisfied if $|f'(\hat{x})| < 1$; and the condition NON-UNIQUE$(\beta, \gamma, \lambda, d)$ is satisfied if $|f'(\hat{x})| > 1$.

The condition UNIQUE$(\beta, \gamma, \lambda, d)$ holds if and only if the dynamical system (1) converges to its unique fixed point $\hat{x}$ at an exponential rate. The name uniqueness condition is due to that UNIQUE$(\beta, \gamma, \lambda, d)$ implies the uniqueness of the Gibbs measure of two-spin system of parameters (β, γ, λ) on the Bethe lattice (i.e., the infinite d-regular tree) and NON-UNIQUE$(\beta, \gamma, \lambda, d)$ implies that there are more than one such measures (Fig. 1).

Efficient approximation algorithms for TWO-SPIN(β, γ, λ) are discovered for special cases of antiferromagnetic two-spin systems within the uniqueness regime, including the hardcore model [12], the antiferromagnetic Ising model [8], and the antiferromagnetic two-spin



Approximating the Partition Function of Two-Spin Systems, Fig. 1 The regime of (β, γ) for which the uniqueness condition UNIQUE$(\beta, \gamma, \lambda, d)$ holds for $\lambda = 1$ and for all integer $d \geq 1$

systems without external field [4], and finally for all antiferromagnetic two-spin systems within the uniqueness regime [5].

Theorem 5 ([5]) *For* $0 \leq \beta \leq \gamma$ *and* $\beta\gamma < 1$, *there is an FPTAS for* TWO-SPIN(β, γ, λ) *on graphs of maximum degree at most* $\Delta_{\max}$ *if* UNIQUE$(\beta, \gamma, \lambda, d)$ *holds for all integer* $1 \leq d \leq \Delta_{\max} - 1$.

This algorithmic result for graphs of bounded maximum degree can be extended to graphs of unbounded degrees.

Theorem 6 ([4, 5]) *For* $0 \leq \beta \leq \gamma$ *and* $\beta\gamma < 1$, *there is an FPTAS for* TWO-SPIN(β, γ, λ) *if* UNIQUE$(\beta, \gamma, \lambda, d)$ *holds for all integer* $d \geq 1$.

All these algorithms follow the framework introduced by Weitz in his seminal work [12]. In this framework, the marginal probability $p_v^{\tau\Lambda}$ is computed by applying the tree recursion (1) on the *tree of self-avoiding walks*, (In fact, (1) is the recursion for the ratio $p_v^{\tau\Lambda}/(1 - p_v^{\tau\Lambda})$ of marginal probabilities.) which enumerates all paths originated from vertex v. Then, a decay of correlation, also called the *spatial mixing* property, is verified, so that a truncated recursion tree of polynomial size is sufficient to provide the required accuracy for the estimation of the marginal probability. For graphs of unbounded degrees, a stronger notion of decay of correlation, called the *computationally efficient correlation decay* [4], is verified to enforce the same cost and accuracy even when the branching number of the recursion tree is unbounded.

On the other hand, for antiferromagnetic two-spin systems in the nonuniqueness regime, the partition function is hard to approximate.

Theorem 7 ([11]) *Let* $0 \leq \beta \leq \gamma$ *and* $\beta\gamma < 1$. *For any* $\Delta_{\max} \geq 3$, *unless* $NP = RP$, *there does not exist an FPRAS for* TWO-SPIN(β, γ, λ) *on graphs of maximum degree at most* $\Delta_{\max}$ *if* NON-UNIQUE$(\beta, \gamma, \lambda, d)$ *holds for some integer* $1 \leq d \leq \Delta_{\max} - 1$.

Altogether, this gives a complete classification of the approximability of partition function of antiferromagnetic two-spin systems except for the uniqueness threshold.

Algorithms for Graphs with Bounded Connective Constant

The *connective constant* is a natural and well-studied notion of the average degree of a graph, which, roughly speaking, measures the growth rate of the number of self-avoiding walks in the graph as their length grows. As a quantity originated from statistical physics, the connective constant has been especially well studied for various infinite regular lattices. In order to suit the algorithmic applications, the definition of connective constant was extended in [9] to families of finite graphs.

Given a vertex v in a graph G, let $N(v, l)$ denote the number of self-avoiding walks of length ℓ in G which start at v.

Definition 2 ([9]) Let $\mathcal{G}$ be a family of finite graphs. The connective constant of $\mathcal{G}$ is at most Δ if there exist constants a and c such that for any graph $G = (V, E)$ in $\mathcal{G}$ and any vertex v in G, it holds that $\sum_{i=1}^{\ell} N(v, i) \leq c\Delta^{\ell}$ for all $\ell \geq a \log |V|$.

The connective constant has a natural interpretation as the "average arity" of the tree of self-avoiding walks.

For certain antiferromagnetic two-spin systems, it is possible to establish the desirable decay of correlation on the tree of self-avoiding walks with bounded average arity instead of maximum arity, and hence the arity d in the uniqueness condition $\text{UNIQUE}(\beta, \gamma, \lambda, d)$ can be replaced with the connective constant Δ. The algorithmic implication of this is stated as the following theorem.

Theorem 8 ([10]) *For the following two cases:*

- *(The hardcore model) $\beta = 0$ and $\gamma = 1$;*
- *(The antiferromagnetic Ising model with zero field) $\beta = \gamma < 1$ and $\lambda = 1$;*

there exists an FPTAS for $\text{TWO-SPIN}(\beta, \gamma, \lambda)$ *on graphs of connective constant at most Δ if* $\text{UNIQUE}(\beta, \gamma, \lambda, \Delta)$ *holds.*

For the two-spin systems considered by this theorem, it holds that $\text{UNIQUE}(\beta, \gamma, \lambda, \Delta)$ implies $\text{UNIQUE}(\beta, \gamma, \lambda, d)$ for all $1 \leq d \leq \Delta$.

The connective constant of a graph of maximum degree $\Delta_{\max}$ is at most $\Delta_{\max} - 1$, but the connective constant of a family of graphs can be much smaller than this crude bound. For example, though the maximum degree of a graph drawn from the Erdös-Rényi model $G(n, d/n)$ is $\Theta(\log n / \log \log n)$ with high probability, the connective constant of such a graph is at most $d(1 + \varepsilon)$ with high probability for any fixed $\varepsilon > 0$. Therefore, for the considered two-spin systems, the algorithm in Theorem 8 works on strictly more general families of graphs than that of Theorem 5.

Cross-References

▸ Complexity Dichotomies for Counting Graph Homomorphisms

Recommended Reading

1. Bulatov AA, Grohe M (2005) The complexity of partition functions. Theor Comput Sci 348(2–3):148–186
2. Goldberg LA, Jerrum M, Paterson M (2003) The computational complexity of two-state spin systems. Random Struct Algorithms 23(2):133–154
3. Jerrum M, Sinclair A (1993) Polynomial-time approximation algorithms for the ising model. SIAM J Comput 22(5):1087–1116
4. Li L, Lu P, Yin Y (2012) Approximate counting via correlation decay in spin systems. In: Proceedings of SODA, Kyoto, pp 922–940
5. Li L, Lu P, Yin Y (2013) Correlation decay up to uniqueness in spin systems. In: Proceedings of SODA, New Orleans, pp 67–84
6. Liu J, Lu P, Zhang C (2014) The complexity of ferromagnetic two-spin systems with external fields. In: Proceedings of RANDOM, Barcelona. To appear
7. Lu P, Wang M, Zhang C (2014) Fptas for weighted fibonacci gates and its applications. In: Esparza J, Fraigniaud P, Husfeldt T, Koutsoupias E (eds) ICALP (1), Copenhagen. Lecture Notes in Computer Science, vol 8572. Springer, pp 787–799

8. Sinclair A, Srivastava P, Thurley M (2012) Approximation algorithms for two-state anti-ferromagnetic spin systems on bounded degree graphs. In: Proceedings of SODA, Kyoto, pp 941–953
9. Sinclair A, Srivastava P, Yin Y (2013) Spatial mixing and approximation algorithms for graphs with bounded connective constant. In: Proceedings of FOCS, Berkeley, pp 300–309
10. Sinclair A, Srivastava P, Štefankovič D, Yin Y (2015) Spatial mixing and the connective constant: Optimal bounds. In: Proceedings of SODA, San Diego. To appear
11. Sly A, Sun N (2012) The computational hardness of counting in two-spin models on d-regular graphs. In: Proceedings of FOCS, New Brunswick, pp 361–369
12. Weitz D (2006) Counting independent sets up to the tree threshold. In: Proceedings of STOC, Seattle, pp 140–149

Approximation Schemes for Bin Packing

Nikhil Bansal
Eindhoven University of Technology, Eindhoven, The Netherlands

Keywords

Cutting-stock problem

Years and Authors of Summarized Original Work

1982; Karmarker, Karp

Problem Definition

In the bin-packing problem, the input consists of a collection of items specified by their sizes. There are also identical bins, which without loss of generality can be assumed to be of size 1, and the goal is to pack these items using the minimum possible number of bins.

Bin packing is a classic optimization problem, and hundreds of its variants have been defined and studied under various settings such as average case analysis, worst-case off-line analysis, and worst-case online analysis. This note considers the most basic variant mentioned above under the off line model where all the items are given in advance. The problem is easily seen to be NP-hard by a reduction from the partition problem. In fact, this reduction implies that unless P = NP, it is impossible to determine in polynomial time whether the items can be packed into two bins or whether they need three bins.

Notations

The input to the bin-packing problem is a set of n items I specified by their sizes $s_1, \ldots, s_n$, where each s_i is a real number in the range (0,1]. A subset of items $S \subseteq I$ can be packed feasibly in a bin if the total size of items in S is at most 1. The goal is to pack all items in I into the minimum number of bins. Let $\mathrm{OPT}(I)$ denote the value of the optimum solution and $\mathrm{Size}(I)$ the total size of all items in I. Clearly, $\mathrm{OPT}(I) \geq \lceil \mathrm{Size}(I) \rceil$.

Strictly speaking, the problem does not admit a polynomial-time algorithm with an approximation guarantee better than 3/2. Interestingly, however, this does not rule out an algorithm that requires, say, $\mathrm{OPT}(I) + 1$ bins (unlike other optimization problems, making several copies of a small hard instance to obtain a larger hard instance does not work for bin packing). It is more meaningful to consider approximation guarantees in an asymptotic sense. An algorithm is called an asymptotic ρ approximation if the number of bins required by it is $\rho \cdot \mathrm{OPT}(I) + O(1)$.

Key Results

During the 1960s and 1970s, several algorithms with constant factor asymptotic and absolute approximation guarantees and very efficient running times were designed (see [1] for a survey). A breakthrough was achieved in 1981 by de la Vega and Lueker [3], who gave the first polynomial-time asymptotic approximation scheme.

Theorem 1 ([3]) *Given any arbitrary parameter $\epsilon > 0$, there is an algorithm that uses $(1 + \epsilon)OPT(I) + O(1)$ bins to pack I. The running*

time of this algorithm is $O(n \log n) + (1 + \epsilon)^{O(1/\epsilon)}$.

The main insight of de la Vega and Lueker [3] was to give a technique for approximating the original instance by a simpler instance where large items have only $O(1)$ distinct sizes. Their idea was simple. First, it suffices to restrict attention to large items, say, with size greater than ε. These can be called I_b. Given an (almost) optimum packing of I_b, consider the solution obtained by greedily filling up the bins with remaining small items, opening new bins only if needed. Indeed, if no new bins are needed, then the solution is still almost optimum since the packing for I_b was almost optimum. If additional bins are needed, then each bin, except possibly one, must be filled to an extent $(1 - \epsilon)$, which gives a packing using $\text{Size}(I)/(1 - \epsilon) + 1 \leq \text{OPT}(I)/(1 - \epsilon) + 1$ bins. So it suffices to focus on solving I_b almost optimally. To do this, the authors show how to obtain another instance I' with the following properties. First, I' has only $O(1/\epsilon^2)$ distinct sizes, and second, I' is an approximation of I_b in the sense that $\text{OPT}(I_b) \geq \text{OPT}(I')$, and moreover, any solution of I' implies another solution of I_b using $O(\epsilon \cdot \text{OPT}(I))$ additional bins. As I' has only $1/\epsilon^2$ distinct item sizes, and any bin can obtain at most $1/\epsilon$ such items, there are at most $O\left(1/\epsilon^2\right)^{1/\epsilon}$ ways to pack a bin. Thus, I' can be solved optimally by exhaustive enumeration (or more efficiently using an integer programming formulation described below).

Later, Karmarkar, and Karp [4] proved a substantially stronger guarantee.

Theorem 2 ([4]) *Given an instance I, there is an algorithm that produces a packing of I using* $\text{OPT}(I) + O(\log^2 \text{OPT}(I))$ *bins. The running time of this algorithm is* $O(n^8)$.

Observe that this guarantee is significantly stronger than that of [3] as the additive term is $O(\log^2 \text{OPT})$ as opposed to $o(\epsilon \cdot \text{OPT})$. Their algorithm also uses the ideas of reducing the number of distinct item sizes and ignoring small items, but in a much more refined way. In particular, instead of obtaining a rounded instance in a single step, their algorithm consists of a logarithmic number of steps where in each step they round the instance "mildly" and then solve it partially.

The starting point is an exponentially large linear programming (LP) relaxation of the problem commonly referred to as the configuration LP. Here, there is a variable x_S corresponding to each subset of items S that can be packed feasibly in a bin. The objective is to minimize $\sum_S x_S$ subject to the constraint that for each item i, the sum of x_S over all subsets S that contain i is at least 1. Clearly, this is a relaxation as setting $x_S = 1$ for each set S corresponding to a bin in the optimum solution is a feasible integral solution to the LP. Even though this formulation has exponential size, the separation problem for the dual is a knapsack problem, and hence the LP can be solved in polynomial time to any accuracy (in particular within an accuracy of 1) using the ellipsoid method. Such a solution is called a fractional packing. Observe that if there are n_i items each of size exactly s_i, then the constraints corresponding to i can be "combined" to obtain the following LP:

$$\begin{aligned} & \min \sum_S x_S \\ \text{s.t.} \quad & \sum_S a_{S,i} x_S \geq n_i \qquad \forall \text{item sizes } i \\ & x_S \geq 0 \qquad\qquad \forall \text{ feasible sets } S. \end{aligned}$$

Here, $a_{S,i}$ is the number of items of size s_i in the feasible S. Let $q(I)$ denote the number of distinct sizes in I. The number of nontrivial constraints in LP is equal to $q(I)$, which implies that there is a basic optimal solution to this LP that has only $q(I)$ variables set nonintegrally. Karmarkar and Karp exploit this observation in a very clever way. The following lemma describes the main idea.

Lemma 1 *Given any instance J, suppose there is an algorithmic rounding procedure to obtain another instance J' such that J' has* $\text{Size}(J)/2$ *distinct item sizes and J and J' are related in the following sense: given any fractional packing of J using ℓ bins gives a fractional packing of J' with at most ℓ bins, and given any packing of J' using ℓ' bins gives a packing of J using $\ell' + c$*

bins, where c is some fixed parameter. Then, J can be packed using $\mathrm{OPT}(J) + c \cdot \log(\mathrm{OPT}(J))$ *bins.*

Proof Let $I_0 = I$ and let I_1 be the instance obtained by applying the rounding procedure to I_0. By the property of the rounding procedure, $\mathrm{OPT}(I) \leq \mathrm{OPT}(I_1) + c$ and $\mathrm{LP}(I_1) \leq \mathrm{LP}(I)$. As I_1 has $\mathrm{Size}(I_0)/2$ distinct sizes, the LP solution for I_1 has at most $\mathrm{Size}(I_0)/2$ fractionally set variables. Remove the items packed integrally in the LP solution, and consider the residual instance I_1'. Note that $\mathrm{Size}(I_1') \leq \mathrm{Size}(I_0)/2$. Now, again apply the rounding procedure to I_1' to obtain I_2 and solve the LP for I_2. Again, this solution has at most $\mathrm{Size}(I_1')/2 \leq \mathrm{Size}(I_0)/4$ fractionally set variables and $\mathrm{OPT}(I_1') \leq \mathrm{OPT}(I_2) + c$ and $\mathrm{LP}(I_2) \leq \mathrm{LP}(I_1')$. The above process is repeated for a few steps. At each step, the size of the residual instance decreases by a factor of at least two, and the number of bins required to pack I_0 increases by additive c. After $\log(\mathrm{Size}(I_0))(\approx \log(\mathrm{OPT}(I)))$ steps, the residual instance has size $O(1)$ and can be packed into $O(1)$ additional bins. □

It remains to describe the rounding procedure. Consider the items in nondecreasing order $s_1 \geq s_2 \geq \cdots \geq s_n$ and group them as follows. Add items to current group until its size first exceeds 2. At this point, close the group and start a new group. Let $G_1, \ldots, G_k$ denote the groups formed and let $n_i = |G_i|$, setting $n_0 = 0$ for convenience. Define I' as the instance obtained by rounding the size of n_{i-1} largest items in G_i to the size of the largest item in G_i for $i = 1, \ldots, k$. The procedure satisfies the properties of Lemma 1 with $c = O(\log n_k)$ (left as an exercise to the reader). To prove Theorem 2, it suffices to show that $n_k = O(\mathrm{Size}(I))$. This is done easily by ignoring all items smaller than $1/\mathrm{Size}(I)$ and filling them in only in the end (as in the algorithm of de la Vega and Lueker).

In the case when the item sizes are not too small, the following corollary is obtained.

Corollary 1 *If all the item sizes are at least δ, it is easily seen that $c = O(\log 1/\delta)$, and the above algorithm implies a guarantee of* $\mathrm{OPT} + O(\log(1/\delta) \cdot \log \mathrm{OPT})$, *which is* $\mathrm{OPT} + O(\log\ \mathrm{OPT})$ *if δ is a constant.*

Recently, Rothvoss gave the first improvement to the result of Karmarkar and Karp and improve their additive guarantee from $\mathrm{O}(\log^2 \mathrm{Opt})$ to O(log Opt log log Opt). His algorithm also uses the configuration LP solution and is based on several new ideas and recent developments. First is the connection of bin packing to a problem in discrepancy theory known as the k-permutation problem. Second are the recently developed algorithmic approaches for addressing discrepancy minimization problems.

In addition to these, a key idea in Rothvoss' algorithm is to glue several small items contained in a configuration into a new large item. For more details, we refer the reader to [5].

Applications

The bin-packing problem is directly motivated from practice and has many natural applications such as packing items into boxes subject to weight constraints, packing files into CDs, packing television commercials into station breaks, and so on. It is widely studied in operations research and computer science. Other applications include the so-called cutting-stock problems where some material such as cloth or lumber is given in blocks of standard size from which items of certain specified size must be cut. Several variations of bin packing, such as generalizations to higher dimensions, imposing additional constraints on the algorithm and different optimization criteria, have also been extensively studied. The reader is referred to [1, 2] for excellent surveys.

Open Problems

Except for the NP-hardness, no other hardness results are known, and it is possible that a polynomial-time algorithm with guarantee OPT + 1 exists for the problem. Resolving this is a key open question. A promising approach seems to be via the configuration LP (considered above).

In fact, no instance is known for which the additive gap between the optimum configuration LP solution and the optimum integral solution is more than 1. It would be very interesting to design an instance that has an additive integrality gap of two or more.

Cross-References

- Bin Packing
- Knapsack

Recommended Reading

1. Coffman EG, Garey MR, Johnson DS (1996) Approximation algorithms for bin packing: a survey. In: Hochbaum D (ed) Approximation algorithms for NP-hard problems. PWS, Boston, pp 46–93
2. Csirik J, Woeginger G (1998) On-line packing and covering problems. In: Fiat A, Woeginger G (eds) Online algorithms: the state of the art. LNCS, vol 1442. Springer, Berlin, pp 147–177
3. Fernandez de la Vega W, Lueker G (1981) Bin packing can be solved within $1+\varepsilon$ in linear time. Combinatorica 1:349–355
4. Karmarkar N, Karp RM (1982) An efficient approximation scheme for the one-dimensional bin-packing problem. In: Proceedings of the 23rd IEEE symposium on foundations of computer science (FOCS), Chicago, pp 312–320
5. Rothvoss T (2013) Approximating bin packing withing O(log OPT log log OPT) bins. In: Proceedings of the 54th IEEE symposium on foundations of computer science (FOCS), Berkeley, pp 20–29

Approximation Schemes for Geometric Network Optimization Problems

Joseph S.B. Mitchell
Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY, USA

Keywords

Approximation algorithms; Computational geometry; Geometric networks; Optimization; Polynomial-time approximation scheme; Traveling salesperson problem

Years and Authors of Summarized Original Work

1998; Arora
1999: Mitchell
1998; Rao, Smith

Problem Definition

Geometric network optimization is the problem of computing a network in a geometric space (e.g., the Euclidean plane), based on an input of geometric data (e.g., points, disks, polygons/polyhedra) that is optimal according to an objective function that typically involves geometric measures, such as Euclidean length, perhaps in addition to combinatorial metrics, such as the number of edges in the network. The desired network is required to have certain properties, such as being connected (or k-connected), having a specific topology (e.g., forming a path/cycle), spanning at least a certain number of input objects, etc.

One of the most widely studied optimization problems is the traveling salesperson problem (TSP): given a set S of n *sites* (e.g., cities), and distances between each pair of sites, determine a route or *tour* of minimum length that visits every member of S. The (symmetric) TSP is often formulated in terms of a graph optimization problem on an edge-weighted complete graph K_n, and the goal is to determine a Hamiltonian cycle (a cycle visiting each vertex exactly once), or a *tour*, of minimum total weight. In geometric settings, the sites are often points in the plane with distances measured according to the Euclidean metric.

The TSP is known to be NP-complete in graphs and NP-hard in the Euclidean plane. Many methods of combinatorial optimization, as well as heuristics, have been developed and applied successfully to solving to optimality instances of TSP; see Cook [7]. Our focus here is on provable approximation algorithms.

In the context of the TSP, a minimization problem, a c-approximation algorithm is an algorithm guaranteed to yield a solution whose objective function value (length) is guaranteed to be at most c times that of an optimal solution. A *polynomial-time approximation scheme* (PTAS) is a family of c-approximation algorithms, with $c = 1 + \varepsilon$, that runs in polynomial (in input size) time for any fixed $\varepsilon > 0$. A *quasi-polynomial-time approximation scheme* (QPTAS) is an approximation scheme, with factor $c = 1 + \varepsilon$ for any fixed $\varepsilon > 0$, whose running time is *quasi-polynomial*, $2^{O((\log n)^C)}$, for some C.

In the Euclidean *Steiner minimum spanning tree* (SMST) problem, the objective is to compute a minimum total length tree that spans all of the input points S, allowing nodes of the tree to be at points of the Euclidean space other than S (such points are known as *Steiner points*). The Euclidean SMST is known to be NP-hard, even in the plane.

Key Results

A simple 2-approximation algorithm for TSP follows from a "doubling" of a minimum spanning tree, assuming that the distances obey the triangle inequality. By augmenting the minimum spanning tree with a minimum-weight matching on the odd-degree nodes of the tree, Christofides obtained a 1.5-approximation for TSP with triangle inequality. This is the currently best-known approximation for general metric spaces; an outstanding open conjecture is that a 4/3-approximation (or better) may be possible. It is known that the TSP in a general metric space is APX-complete, implying that, unless P = NP, no PTAS exists, in general.

Research has shown that "geometry helps" in network optimization problems. Geometric structure has played a key role in solving combinatorial optimization problems. There are problems that are NP-hard in their abstract generality, yet are solvable exactly in polynomial time in geometric settings (e.g., maximum TSP in polyhedral metrics), and there are problems for which we have substantially better, or more efficient, approximation algorithms in geometric settings (e.g., TSP).

As shown by Arora [1] and Mitchell [10] in papers originally appearing in 1996, geometric instances of TSP and SMST have special structure that allows for the existence of a PTAS. Arora [1] gives a randomized algorithm that, with probability 1/2, yields a $(1 + \varepsilon)$-approximate tour in time $n(\log n)^{O(\sqrt{d}/\varepsilon)^{d-1}}$ in Euclidean d-space. Rao and Smith [14] obtain a deterministic algorithm with running time $2^{(d/\varepsilon)^{O(d)}} n + (d/\varepsilon)^{O(d)} n \log n$. This $O(n \log n)$ bound (for fixed d, ε) matches the $\Omega(n \log n)$ lower bound in the decision tree bound. In the real RAM model, with atomic floor or mod function, Bartal and Gottlieb [3] give a randomized linear-time PTAS that, with probability $1 - e^{-O(n^{1/3d})}$, computes a $(1 + \varepsilon)$-approximation to an optimal tour in time $2^{(d/\varepsilon)^{O(d)}} n$. The exponential dependence on d in the PTAS bounds is essentially best possible, since Trevisan has shown that if $d \geq \log n$, it is NP-hard to obtain a $(1 + \varepsilon)$-approximation.

A key insight of Rao and Smith is the application of the concept of "spanners" to the approximation schemes. A connected subgraph G of the complete Euclidean graph, joining every pair of points in S (within Euclidean d-dimensional space), is said to be a *t-spanner* for S if all points of S are nodes of G and, for any points $u, v \in S$, the length of a shortest path in G from u to v is at most t times the Euclidean distance, $d_2(u, v)$. It is known that for any point set S and $t > 1$, t-spanners exist and can be calculated in time $O(n \log n)$, with the property that the t-spanner is *lightweight*, meaning that the sum of its edge lengths is at most a constant factor (depending on d and t) greater than the Euclidean length of a minimum spanning tree on S.

Overview of Methods

The PTAS techniques are based on structure theorems showing that an optimal solution can be "rounded" to a "nearby" solution, of length at most a factor $(1 + \varepsilon)$ longer, that falls within a special class of recursively "nice" solutions for which optimization via dynamic programming

can be done efficiently, because the interface between adjacent subproblems is "small" combinatorially. Arora's algorithm [1] is randomized, as is that of Rao and Smith [14]; both can be derandomized. The m-guillotine method (Mitchell [10]) is directly a deterministic method; however, the proof of its structure theorem is effectively an averaging argument.

Arora's Dissection Method

Arora [1, 2] gives a method based on geometric dissection using a quadtree (or its octtree analogue in d dimensions). On the boundary of each quadtree square are m equally spaced points ("portals"); a *portal-respecting tour* is one that crosses the boundaries of squares only at portals. Using an averaging argument based on a randomly shifted quadtree that contains the bounding square of S, Arora proves structure theorems, the simplest of which shows that, when $m > (\log n)/\epsilon$, the expected length of a shortest portal-respecting tour, T, is at most $(1+\epsilon)$ times the length of an optimal tour. Within a quadtree square, T consists of at most m disjoint paths that together visit all sites within the square. Since the interface data specifying a subproblem has size $2^{O(m)}$, dynamic programming computes a shortest portal-respecting tour in time $2^{(O(m)}$ per quadtree square, for overall time $2^{O((\log n)/\epsilon)} = n^{O(1/\epsilon)}$. An improved, near-linear (randomized) running time is obtained via a stronger structure theorem, based on "(m,k)-light" tours, which are portal respecting and enter/leave each square at most k times (with $k = O(1/\varepsilon)$). Rao and Smith's improvement uses the observation that it suffices to restrict the algorithm to use edges of a $(1+\varepsilon)$-spanner.

The m-Guillotine Method

The m-guillotine method of Mitchell [10] is based on the notion of an *m-guillotine structure*. A geometric graph G in the plane has the m-guillotine structure if the following holds: either (1) G has $O(m)$ edges or (2) there exists a *cut* by an axis-parallel line ℓ such that the intersection of ℓ with the edge set of G has $O(m)$ connected components and the subgraphs of G on each side of the cut recursively also have the m-guillotine structure. The m-guillotine structure is defined in dimensions $d > 2$ as well, using hyperplane cuts orthogonal to the coordinate axes.

The m-guillotine structure theorem in 2 dimensions states that, for any positive integer m, a set E of straight line segments in the plane is either m-guillotine already or is "close" to being m-guillotine, in that there exists a superset, $E_m \supset E$ that has m-guillotine structure, where E_m is obtained from E by adding a set of axis-parallel segments (*bridges*, or *m-spans*) of total length at most $O(\varepsilon|E|)$. The proof uses a simple charging scheme.

The m-guillotine method originally (1996) yielded a PTAS for TSP and related problems in the plane, with running time $n^{O(1/\epsilon)}$; this was improved (1997) to $n^{O(1)}$. With the injection of the idea of Rao and Smith [14] to employ spanners, the m-guillotine method yields a simple, deterministic $O(n \log n)$ time PTAS for TSP and related problems in fixed dimension $d \geq 2$. The steps are the following: (a) construct (in $O(n \log n)$ time) a spanner, T; (b) compute its m-guillotine superset, T_m, by standard sweep techniques (in time $O(n \log n)$); and (c) use dynamic programming (time $O(n)$) applied to the recursive tree associated with T_m, to optimize over spanning subgraphs of T_m.

Generalizations to Other Metrics

The PTAS techniques described above have been significantly extended to variants of the Euclidean TSP. While we do not expect that a PTAS exists for general metric spaces (because of APX-hardness), the methods can be extended to a very broad class of "geometric" metrics known as *doubling metrics*, or metric spaces of bounded *doubling dimension*. A metric space $\mathcal{X}$ is said to have *doubling constant* c_d if any ball of radius r can be covered by c_d balls of radius $r/2$; the logarithm of c_d is the *doubling dimension* of $\mathcal{X}$. Euclidean d-space has doubling dimension $O(d)$. Bartal, Gottlieb, and Krauthgamer [4] have given a PTAS for TSP in doubling metrics, improving on a prior QPTAS.

For the discrete metric space induced by an edge-weighted planar graph, the TSP has a linear-time PTAS. The *subset TSP* for edge-weighted

planar graphs, in which there is a subset $S \subseteq V$ of the vertex set V that must be visited, also has an efficient ($O(n \log n)$ time) PTAS; this implies a PTAS for the *geodesic metric* for TSP on a set S of sites in a polygonal domain in the plane, with distances given by the (Euclidean) lengths of geodesic shortest paths between pairs of sites.

Applications to Network Optimization

The approximation schemes we describe above have been applied to numerous geometric network optimization problems, including the list below. We do not give references for most of the results summarized below; see the surveys [2, 11, 12] and Har-Peled [9].

1. A PTAS for the Euclidean Steiner minimum spanning tree (SMST) problem.
2. A PTAS for the Euclidean minimum Steiner forest problem, in which one is to compute a minimum-weight forest whose connected components (Steiner trees) span given (disjoint) subsets $S_1, \ldots, S_K \subset S$ of the sites, allowing Steiner points.
3. A PTAS for computing a minimum-weight k-connected spanning graph of S in Euclidean d-space.
4. A PTAS for the *k-median problem*, in which one is to determine k *centers*, among S, in order to minimize the sum of the distances from the sites S to their nearest center points.
5. A PTAS for the *minimum latency problem* (MLP), also known as the *deliveryman problem* or the *traveling repairman problem*, in which one is to compute a tour on S that minimizes the sum of the "latencies" of all points, where the *latency* of a point p is the length of the tour from a given starting point to the point p. The PTAS of Sitters [15] runs in time $n^{O(1/\epsilon)}$, improving the prior QPTAS.
6. A PTAS for the k-TSP (and k-MST), in which one is to compute a shortest tour (tree) spanning at least k of the n sites of S.
7. A QPTAS for degree-bounded spanning trees in the plane.
8. A QPTAS for the capacitated vehicle routing problem (VRP) [8], in which one is to compute a minimum-length collection of tours, each visiting at most k sites of S. A PTAS is known for some values of k.
9. A PTAS for the *orienteering problem*, in which the goal is to maximize the number of sites visited by a length-bounded tour [6].
10. A PTAS for *TSP with Neighborhoods* (TSPN), in which each site of the input set S is a connected region of d-space (rather than a point), and the goal is to compute a tour that visits each region. The TSPN has a PTAS for regions in the plane that are "fat" and are weakly disjoint (no point lies in more than a constant number of regions) [13]. Chan and Elbassioni [5] give a QPTAS for fat, weakly disjoint regions in doubling metrics. For TSPN with disconnected regions, the problem is that of the "group TSP" (also known as "generalized TSP" or "one-of-a-set TSP"), which, in general metrics, is much harder than TSP; even in the Euclidean plane, the problem is NP-hard to approximate to within any constant factor for finite point sets and is NP-hard to approximate better than a fixed constant for visiting point pairs.
11. A PTAS for the *milling* and *lawnmowing* problems, in which one is to compute a shortest path/tour for a specified cutter so that all points of a given region R in the plane is swept over by the cutter head while keeping the cutter fully within the region R (milling), or allowing the cutter to sweep over points outside of region R (lawnmowing).
12. A PTAS for computing a minimum-length cycle that separates a given set of "red" points from a given set of "blue" points in the Euclidean plane.
13. A QPTAS for the *minimum-weight triangulation* (MWT) problem of computing a triangulation of the planar point set S in order to minimize the sum of the edge lengths. The MWT has been shown to be NP-hard.

14. A PTAS for the minimum-weight Steiner convex partition problem in the plane, in which one is to compute an embedded planar straight-line graph with convex faces whose vertex set contains the input set S.

Open Problems

A prominent open problem in approximation algorithms for network optimization is to determine if approximations better than factor 3/2 can be achieved for the TSP in general metric spaces.

Specific open problems for geometric network optimization problems include:

1. Is there a PTAS for minimum-weight triangulation (MWT) in the plane? (A QPTAS is known.)
2. Is there a PTAS for capacitated vehicle routing, for all k?
3. Is there a PTAS for Euclidean minimum spanning trees of bounded degree (3 or 4)? (A QPTAS is known for degree-3 trees.)
4. Is there a PTAS for TSP with Neighborhoods (TSPN) for connected disjoint regions in the plane?
5. Is there a PTAS for computing a minimum-weight t-spanner of a set of points in a Euclidean space?

Finally, can PTAS techniques be implemented to be competitive with other practical methods for TSP or related network optimization problems?

Cross-References

- Applications of Geometric Spanner Networks
- Euclidean Traveling Salesman Problem
- Metric TSP
- Minimum Geometric Spanning Trees
- Minimum k-Connected Geometric Networks
- Steiner Trees

Recommended Reading

1. Arora S (1998) Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J ACM 45(5):753–782
2. Arora S (2007) Approximation algorithms for geometric TSP. In: Gutin G, Punnen AP (eds) The traveling salesman problem and its variations. Combinatorial Optimization, vol 12. Springer, New York/Berlin, pp 207–221
3. Bartal Y, Gottlieb LA (2013) A linear time approximation scheme for Euclidean TSP. In: Proceedings of the 54th IEEE Foundations of Computer Science (FOCS). IEEE, Piscataway, pp 698–706
4. Bartal Y, Gottlieb LA, Krauthgamer R (2012) The traveling salesman problem: low-dimensionality implies a polynomial time approximation scheme. In: Proceedings of the 44th ACM Symposium on Theory of Computing. ACM, New York, pp 663–672
5. Chan TH, Elbassioni KM (2011) A QPTAS for TSP with fat weakly disjoint neighborhoods in doubling metrics. Discret Comput Geom 46(4):704–723
6. Chen K, Har-Peled S (2008) The Euclidean orienteering problem revisited. SIAM J Comput 38(1):385–397
7. Cook W (2011) In pursuit of the travelling salesman: mathematics at the limits of computation. Princeton University Press, Princeton
8. Das A, Mathieu C (2015) A quasipolynomial time approximation scheme for Euclidean capacitated vehicle routing. Algorithmica 73(1):115–142
9. Har-Peled S (2011) Geometric approximation algorithms, vol 173. American Mathematical Society, Providence
10. Mitchell JSB (1999) Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. SIAM J Comput 28(4):1298–1309
11. Mitchell JSB (2000) Geometric shortest paths and network optimization. In: Sack JR, Urrutia J (eds) Handbook of Computational Geometry. Elsevier Science, North-Holland/Amsterdam, pp 633–701
12. Mitchell JSB (2004) Shortest paths and networks. In: Goodman JE, O'Rourke J (eds) Handbook of Discrete and Computational Geometry, 2nd edn. Chapman & Hall/CRC, Boca Raton, chap 27, pp 607–641
13. Mitchell JSB (2007) A PTAS for TSP with neighborhoods among fat regions in the plane. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, New Orleans, pp 11–18
14. Rao SB, Smith WD (1998) Approximating geometrical graphs via spanners and banyans. In: Proceedings of the 30th ACM Symposium on Theory of Computing. ACM, New York, pp 540–550
15. Sitters R (2014) Polynomial time approximation schemes for the traveling repairman and other minimum latency problems. In: Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms, Portland, pp 604–616

Approximation Schemes for Makespan Minimization

Asaf Levin
Faculty of Industrial Engineering and Management, The Technion, Haifa, Israel

Keywords

Approximation scheme; Load balancing; Parallel machine scheduling

Years and Authors of Summarized Original Work

1987, 1988; Hochbaum, Shmoys

Problem Definition

Non-preemptive makespan minimization on m uniformly related machines is defined as follows. We are given a set $M = \{1, 2, \ldots, m\}$ of m machines where each machine i has a speed s_i such that $s_i > 0$. In addition we are given a set of jobs $J = \{1, 2, \ldots, n\}$, where each job j has a positive size p_j and all jobs are available for processing at time 0. The jobs need to be partitioned into m subsets $S_1, \ldots, S_m$, with S_i being the subset of jobs assigned to machine i, and each such (ordered) partition is a feasible solution to the problem. Processing job j on machine i takes $\frac{p_j}{s_i}$ time units. For such a solution (also known as a schedule), we let $L_i = (\sum_{j \in S_i} p_j)/s_i$ be the *completion time* or *load* of machine i. The *work* of machine i is $W_i = \sum_{j \in S_i} p_j = L_i \cdot s_i$, that is, the total size of the jobs assigned to i. The *makespan* of the schedule is defined as $\max\{L_1, L_2, \ldots, L_m\}$, and the goal is to find a schedule that minimizes the makespan. We also consider the problem on identical machines, that is, the special case of the above problem in which $s_i = 1$ for all i (in this special case, the work and the load of a given machine are always the same).

Key Results

A PTAS (polynomial-time approximation scheme) is a family of polynomial-time algorithms such that for all $\epsilon > 0$, the family has an algorithm such that for every instance of the makespan minimization problem, it returns a feasible solution whose makespan is at most $1 + \epsilon$ times the makespan of an optimal solution to the same instance. Without loss of generality, we can assume that $\epsilon < \frac{1}{5}$.

The Dual Approximation Framework and Common Preprocessing Steps

Using a guessing step of the optimal makespan, and scaling the sizes of all jobs by the value of the optimal makespan, we can assume that the optimal makespan is in the interval $[1, 1 + \epsilon)$ and it suffices to construct a feasible solution whose makespan is at most $1 + c\epsilon$ for a constant c (then scaling ϵ before applying the algorithm will give the claimed result). This assumption can be made since we can find in polynomial time two values LB and UB such that the optimal makespan is in the interval $[LB, UB]$ and $\frac{UB}{LB}$ is at most some constant (or even at most an exponential function of the length of the binary encoding of the input), then using a constant (or polynomial) number of iterations, we can find the minimum integer power of $1 + \epsilon$ for which the algorithm below will succeed to find a schedule with makespan at most $1 + c\epsilon$ times the optimal makespan. This approach is referred to as the *dual approximation method* [7,8].

From now on, we assume that the optimal makespan is in the interval $[1, 1 + \epsilon)$. The next step is to round up the size of each job to the next integer power of $1 + \epsilon$ and to round down the speed of each machine to the next integer power of $1 + \epsilon$. That is, the rounded size of job j is $p'_j = (1 + \epsilon)^{\lceil \log_{1+\epsilon} p_j \rceil}$ and the rounded speed of machine i is $s'_i = (1 + \epsilon)^{\lfloor \log_{1+\epsilon} s_i \rfloor}$. Note that this rounding does not decrease the makespan of any feasible solution and increase the optimal makespan by a multiplicative factor of at most $(1 + \epsilon)^2$. Thus, in the new instance that we call *the rounded instance*, the makespan

of an optimal solution is in the interval $[1, (1+\epsilon)^3)$. We observe that if the original instance to the makespan minimization problem was for the special case of identical machines, so does the rounded instance. The next steps differ between the PTAS for identical machines and its generalization for related machines.

The Case of Identical Machines

We define a job to be *small* if its rounded size is at most ϵ, and otherwise it is *large*. The large jobs instance is the instance we obtain from the rounded instance by removing all small jobs. The first observation is that it is sufficient to design an algorithm for finding a feasible solution to the large jobs instance whose makespan is at most $1 + c\epsilon$ where $c \geq 5$ is some constant. This is so, because we can apply this algorithm on the large jobs instance and obtain a schedule of these large jobs. Later, we add to the schedule the small jobs one by one using the list scheduling heuristic [5]. In the analysis, there are two cases. In the first one, adding the small jobs did not increase the makespan of the resulting schedule, and in this case our claim regarding the makespan of the output of the algorithm clearly holds. In the second case, the makespan increased by adding the small jobs, and we consider the last iteration in which such increase happened. In that last iteration, the load of one machine was increased by the size of the job assigned in this iteration, that is by at most ϵ, and before this iteration its load was smaller than $\frac{\sum_j p'_j}{m} \leq (1+\epsilon)^3$, where the inequality holds because the makespan of a feasible solution cannot be smaller than the average load of the machines. The claim now follows using $(1+\epsilon)^3 + \epsilon \leq 1 + 5\epsilon$ as $\epsilon < \frac{1}{5}$.

The large jobs instance has a compact representation. There are m identical machines and jobs of at most $O(\log_{1+\epsilon} \frac{1}{\epsilon})$ distinct sizes. Note that each machine has at most $\frac{2}{\epsilon}$ large jobs assigned to it (in any solution with makespan smaller than 2), and thus there are a constant number of different schedules of one machine when we consider jobs of the same size as identical jobs. A schedule of one machine in a solution to the large jobs instance is called the *configuration of the machine*. Now, we can either perform a dynamic programming algorithm that assigns large jobs to one machine after the other while recalling in each step the number of jobs of each size that still need to be assigned (as done in [7]) or use an integer program of fixed dimension [9] to solve the problem of assigning all large jobs to configurations of machines while having at most m machines in the solution and allowing only configurations corresponding to machines with load at most $(1+\epsilon)^3$ as suggested by Shmoys (see [6]).

We refer to [1, 2], and [10] for PTASs for other load balancing problems on identical machines.

The Case of Related Machines

Here, we still would like to consider separately the large jobs and the small jobs; however, a given job j can be large for one machine and small for another machine (it may even be too large for other machines, that is, processing it on such machine may take a period of time that is longer than the makespan of an optimal solution). Thus, for a given job j, we say that it is *huge* for machine i if $\frac{p_j}{s_i} > (1+\epsilon)^3$, it is *large* for machine i if $\epsilon < \frac{p_j}{s_i} \leq (1+\epsilon)^3$, and otherwise it is *small* for machine i. A configuration of machine i is the number of large jobs of each rounded size that are assigned to machine i (observe that similarly to the case of identical machines, the number of sizes of large jobs for a given machine is a constant) as well as approximate value of the total size of small jobs assigned to machine i, that is a value Δ_i such that the total size of small jobs assigned to machine i is in the interval $\left((\Delta_i - 1)\epsilon \cdot \frac{1}{s_i}, \Delta_i \epsilon \cdot \frac{1}{s_i}\right]$. Note that the vector of configurations of machines defines some information about the schedule, but it does not give a one-to-one assignment of small jobs to the machines.

Once again, [8] suggested to use dynamic programming for assigning jobs to the machines by traversing the machines from the slowest one to the fastest one and, for each machine, decide the number of large jobs of each size as well as an approximate value of the total size of small

jobs (for that machine) assigned to it. That is, the dynamic programming decides the configuration of each machine. To do so, it needs to recall the number of large jobs (with respect to the current machine) that are still not assigned, as well as the total size of small jobs (with respect to the current machine) that are still not assigned (this total size is a rounded value). At a postprocessing step, the jobs assigned as large jobs by the solution for the dynamic programming are scheduled accordingly, while the other jobs are assigned as small jobs as follows.

We assign the small jobs to the machines while traversing the machines from slowest to fastest and assigning the small jobs from the smallest to largest. At each time we consider the current machine i and the prefix of unassigned small jobs that are small with respect to the current machine. Denote by Δ_i the value of this parameter according to the solution of the dynamic programming. Due to the successive rounding of the total size of unassigned small jobs, we will allow assignments of a slightly larger total size of small jobs to machine i. So we will assign the small jobs one by one as long as their total size is at most $\frac{(\Delta_i+4)\epsilon}{s_i}$. If at some point there are no further unassigned small jobs that are small for machine i, we move to the next machine, otherwise we assign machine i small jobs (for machine i) of total size of at least $\frac{(\Delta_i+3)\epsilon}{s_i}$. This suffices to guarantee the feasibility of the resulting solution (i.e., all jobs are assigned) while increasing the makespan only by a small amount.

We refer to [3] and [4] for PTASs for other load balancing problems on related machines.

Cross-References

- Robust Scheduling Algorithms
- Vector Scheduling Problems

Recommended Reading

1. Alon N, Azar Y, Woeginger GJ, Yadid T (1997) Approximation schemes for scheduling. In: Proceedings of the 8th symposium on discrete algorithms (SODA), New Orleans, USA pp 493–500
2. Alon N, Azar Y, Woeginger GJ, Yadid T (1998) Approximation schemes for scheduling on parallel machines. J Sched 1(1):55–66
3. Azar Y, Epstein L (1998) Approximation schemes for covering and scheduling on related machines. In: Proceedings of the 1st international workshop on approximation algorithms for combinatorial optimization (APPROX), Aalborg, Denmark pp 39–47
4. Epstein L, Sgall J (2004) Approximation schemes for scheduling on uniformly related and identical parallel machines. Algorithmica 39(1):43–57
5. Graham RL (1966) Bounds for certain multiprocessing anomalies. Bell Syst Tech J 45(9):1563–1581
6. Hochbaum DS (1997) Various notions of approximations: Good, better, best and more. In: Hochbaum DS (ed) Approximation algorithms, PWS Publishing Company, Boston
7. Hochbaum DS, Shmoys DB (1987) Using dual approximation algorithms for scheduling problems: theoretical and practical results. J ACM 34(1):144–162
8. Hochbaum DS, Shmoys DB (1988) A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. SIAM J Comput 17(3):539–551
9. Lenstra HW Jr (1983) Integer programming with a fixed number of variables. Math Oper Res 8(4):538–548
10. Woeginger GJ (1997) A polynomial-time approximation scheme for maximizing the minimum machine completion time. Oper Res Lett 20(4):149–154

Approximation Schemes for Planar Graph Problems

Mohammad Taghi Hajiaghayi[1] and Erik D. Demaine[3]
[1]Department of Computer Science, University of Maryland, College Park, MD, USA
[2]MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA

Keywords

Approximation algorithms in planar graphs; Baker's approach; Lipton-Tarjan approach

Years and Authors of Summarized Original Work

1983; Baker
1994; Baker

Problem Definition

Many NP-hard graph problems become easier to approximate on planar graphs and their generalizations. (A graph is *planar* if it can be drawn in the plane (or the sphere) without crossings. For definitions of other related graph classes, see the entry on ▸ Bidimensionality (2004; Demaine, Fomin, Hajiaghayi, Thilikos).) For example, a *maximum independent set* asks to find a maximum subset of vertices in a graph that induce no edges. This problem is inapproximable in general graphs within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$ unless NP = ZPP (and inapproximable within $n^{1/2-\epsilon}$ unless P = NP), while for planar graphs, there is a 4-approximation (or simple 5-approximation) by taking the largest color class in a vertex 4-coloring (or 5-coloring). Another is *minimum dominating set*, where the goal is to find a minimum subset of vertices such that every vertex is either in or adjacent to the subset. This problem is inapproximable in general graphs within $\epsilon \log n$ for some $\epsilon > 0$ unless P = NP, but as we will see, for planar graphs, the problem admits a *polynomial-time approximation scheme* (PTAS): a collection of $(1 + \epsilon)$-approximation algorithms for all $\epsilon > 0$.

There are two main general approaches to designing PTASs for problems on planar graphs and their generalizations: the separator approach and the Baker approach.

Lipton and Tarjan [15, 16] introduced the first approach, which is based on planar separators. The first step in this approach is to find a separator of $O(\sqrt{n})$ vertices or edges, where n is the size of the graph, whose removal splits the graph into two or more pieces each of which is a constant fraction smaller than the original graph. Then, recurse in each piece, building a recursion tree of separators, and stop when the pieces have some constant size such as $1/\epsilon$. The problem can be solved on these pieces by brute force, and then it remains to combine the solutions up the recursion tree. The induced error can often be bounded in terms of the total size of all separators, which in turn can be bounded by ϵn. If the optimal solution is at least some constant factor times n, this approach often leads to a PTAS.

There are two limitations to this planar-separator approach. First, it requires that the optimal solution be at least some constant factor times n; otherwise, the cost incurred by the separators can be far larger than the desired optimal solution. Such a bound is possible in some problems after some graph pruning (linear kernelization), e.g., independent set, vertex cover, and forms of the traveling salesman problem. But, for example, Grohe [12] states that the dominating set is a problem "to which the technique based on the separator theorem does not apply." Second, the approximation algorithms resulting from planar separators are often impractical because of large constant factors. For example, to achieve an approximation ratio of just 2, the base case requires exhaustive solution of graphs of up to $2^{2,400}$ vertices.

Baker [1] introduced her approach to address the second limitation, but it also addresses the first limitation to a certain extent. This approach is based on decomposition into overlapping subgraphs of bounded outerplanarity, as described in the next section.

Key Results

Baker's original result [1] is a PTAS for a maximum independent set (as defined above) on planar graphs, as well as the following list of problems on planar graphs: maximum tile salvage, partition into triangles, maximum H-matching, minimum vertex cover, minimum dominating set, and minimum edge-dominating set.

Baker's approach starts with a planar embedding of the planar graph. Then it divides vertices into *layers* by iteratively removing vertices on the outer face of the graph: layer j consists of the vertices removed at the jth iteration. If one now removes the layers congruent to i modulo k, for any choice of i, the graph separates into connected components each with at most k consecutive layers, and hence the graph becomes k-outerplanar. Many NP-complete problems can be solved on k-outerplanar graphs for fixed k using dynamic programming (in particular, such graphs

have bounded treewidth). Baker's approximation algorithm computes these optimal solutions for each choice i of the congruence class of layers to remove and returns the best solution among these k solutions. The key argument for maximization problems considers the optimal solution to the full graph and argues that the removal of one of the k congruence classes of layers must remove at most a $1/k$ fraction of the optimal solution, so the returned solution must be within a $1 + 1/k$ factor of optimal. A more delicate argument handles minimization problems as well. For many problems, such as maximum independent set, minimum dominating set, and minimum vertex cover, Baker's approach obtains a $(1 + \epsilon)$-approximation algorithms with a running time of $2^{O(1/\epsilon)}n^{O(1)}$ on planar graphs.

Eppstein [10] generalized Baker's approach to a broader class of graphs called graphs of *bounded local treewidth*, i.e., where the treewidth of the subgraph induced by the set of vertices at a distance of at most r from any vertex is bounded above by some function $f(r)$ independent of n. The main differences in Eppstein's approach are replacing the concept of bounded outerplanarity with the concept of bounded treewidth, where dynamic programming can still solve many problems, and labeling layers according to a simple breadth-first search. This approach has led to PTASs for hereditary maximization problems such as maximum independent set and maximum clique, maximum triangle matching, maximum H-matching, maximum tile salvage, minimum vertex cover, minimum dominating set, minimum edge-dominating set, minimum color sum, and subgraph isomorphism for a fixed pattern [6, 8, 10]. Frick and Grohe [11] also developed a general framework for deciding any property expressible in first-order logic in graphs of bounded local treewidth.

The foundation of these results is Eppstein's characterization of minor-closed families of graphs with bounded local treewidth [10]. Specifically, he showed that a minor-closed family has bounded local treewidth if and only if it excludes some *apex graph*, a graph with a vertex whose removal leaves a planar graph. Unfortunately, the initial proof of this result brought Eppstein's approach back to the realm of impracticality, because his bound on local treewidth in a general apex-minor-free graph is doubly exponential in r : $2^{2O(r)}$. Fortunately, this bound could be improved to $2^{O(r)}$ [3] and even the optimal $O(r)$ [4]. The latter bound restores Baker's $2^{O(1/\epsilon)}n^{O(1)}$ running time for $(1 + \epsilon)$-approximation algorithms, now for all apex-minor-free graphs.

Another way to view the necessary decomposition of Baker's and Eppstein's approaches is that the vertices or edges of the graph can be split into any number k of pieces such that deleting any one of the pieces results in a graph of bounded treewidth (where the bound depends on k). Such decompositions in fact exist for arbitrary graphs excluding any fixed minor H [9], and they can be found in polynomial time [6]. This approach generalizes the Baker-Eppstein PTASs described above to handle general H-minor-free graphs.

This decomposition approach is effectively limited to *deletion-closed* problems, whose optimal solution only improves when deleting edges or vertices from the graph. Another decomposition approach targets *contraction-closed* problems, whose optimal solution only improves when contracting edges. These problems include classic problems such as dominating set and its variations, the traveling salesman problem, subset TSP, minimum Steiner tree, and minimum-weight c-edge-connected submultigraph. PTASs have been obtained for these problems in planar graphs [2, 13, 14] and in bounded-genus graphs [7] by showing that the edges can be decomposed into any number k of pieces such that contracting any one piece results in a bounded-treewidth graph (where the bound depends on k).

Applications

Most applications of Baker's approach have been limited to optimization problems arising from "local" properties (such as those definable in first-order logic). Intuitively, such local properties can be decided by locally checking every constant-

size neighborhood. In [5], Baker's approach is generalized to obtain PTASs for nonlocal problems, in particular, connected dominating set. This generalization requires the use of two different techniques. The first technique is to use an ε-fraction of a constant-factor (or even logarithmic-factor) approximation to the problem as a "backbone" for achieving the needed nonlocal property. The second technique is to use subproblems that overlap by $\Theta(\log n)$ layers instead of the usual $\Theta(1)$ in Baker's approach.

Despite this advance in applying Baker's approach to more general problems, the planar-separator approach can still handle some different problems. Recall, though, that the planar-separator approach was limited to problems in which the optimal solution is at least some constant factor times n. This limitation has been overcome for a wide range of problems [5], in particular obtaining a PTAS for feedback vertex set, to which neither Baker's approach nor the planar-separator approach could previously apply. This result is based on evenly dividing the optimum solution instead of the whole graph, using a relation between treewidth and the optimal solution value to bound the treewidth of the graph, thus obtaining an $O(\sqrt{\mathrm{OPT}})$ separator instead of an $O(\sqrt{n})$ separator. The $O(\sqrt{\mathrm{OPT}})$ bound on treewidth follows from the bidimensionality theory described in the entry on ▸ Bidimensionality (2004; Demaine, Fomin, Hajiaghayi, Thilikos). We can divide the optimum solution into roughly even pieces, without knowing the optimum solution, by using existing constant-factor (or even logarithmic-factor) approximations for the problem. At the base of the recursion, pieces no longer have bounded size but do have bounded treewidth, so fast fixed-parameter algorithms can be used to construct optimal solutions.

Open Problems

An intriguing direction for future research is to build a general theory for PTASs of subset problems. Although PTASs for subset TSP and Steiner tree have recently been obtained for planar graphs [2, 14], there remain several open problems of this kind, such as subset feedback vertex set, group Steiner tree, and directed Steiner tree.

Another instructive problem is to understand the extent to which Baker's approach can be applied to nonlocal problems. Again there is an example of how to modify the approach to handle the nonlocal problem of connected dominating set [5], but, for example, the only known PTAS for feedback vertex set in planar graphs follows the separator approach.

Cross-References

- ▸ Bidimensionality
- ▸ Separators in Graphs
- ▸ Treewidth of Graphs

Recommended Reading

1. Baker BS (1994) Approximation algorithms for NP-complete problems on planar graphs. J Assoc Comput Mach 41(1):153–180
2. Borradaile G, Kenyon-Mathieu C, Klein PN (2007) A polynomial-time approximation scheme for Steiner tree in planar graphs. In: Proceedings of the 18th annual ACM-SIAM symposium on discrete algorithms (SODA'07), New Orleans
3. Demaine ED, Hajiaghayi M (2004) Diameter and treewidth in minor-closed graph families, revisited. Algorithmica 40(3):211–215
4. Demaine ED, Hajiaghayi M (2004) Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In: Proceedings of the 15th ACM-SIAM symposium on discrete algorithms (SODA'04), New Orleans, pp 833–842
5. Demaine ED, Hajiaghayi M (2005) Bidimensionality: new connections between FPT algorithms and PTASs. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms (SODA'05), Vancouver, pp 590–601
6. Demaine ED, Hajiaghayi M, Kawarabayashi K-I (2005) Algorithmic graph minor theory: decomposition, approximation, and coloring. In: Proceedings of the 46th annual IEEE symposium on foundations of computer science, Pittsburgh, pp 637–646
7. Demaine ED, Hajiaghayi M, Mohar B (2007) Approximation algorithms via contraction decomposition. In: Proceedings of the 18th annual ACM-SIAM symposium on discrete algorithms (SODA'07), New Orleans, 7–9 Jan 2007, pp 278–287

8. Demaine ED, Hajiaghayi M, Nishimura N, Ragde P, Thilikos DM (2004) Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. J Comput Syst Sci 69(2):166–195
9. DeVos M, Ding G, Oporowski B, Sanders DP, Reed B, Seymour P, Vertigan D (2004) Excluding any graph as a minor allows a low tree-width 2coloring. J Comb Theory Ser B 91(1):25–41
10. Eppstein D (2000) Diameter and treewidth in minor-closed graph families. Algorithmica 27(3–4):275–291
11. Frick M, Grohe M (2001) Deciding first-order properties of locally tree-decomposable structures. J ACM 48(6):1184–1206
12. Grohe M (2003) Local tree-width, excluded minors, and approximation algorithms. Combinatorica 23(4):613–632
13. Klein PN (2005) A linear-time approximation scheme for TSP for planar weighted graphs. In: Proceedings of the 46th IEEE symposium on foundations of computer science, Pittsburgh, pp 146–155
14. Klein PN (2006) A subset spanner for planar graphs, with application to subset TSP. In: Proceedings of the 38th ACM symposium on theory of computing, Seattle, pp 749–756
15. Lipton RJ, Tarjan RE (1979) A separator theorem for planar graphs. SIAM J Appl Math 36(2):177–189
16. Lipton RJ, Tarjan RE (1980) Applications of a planar separator theorem. SIAM J Comput 9(3):615–627

Approximations of Bimatrix Nash Equilibria

Paul (Pavlos) Spirakis
Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece
Computer Science, University of Liverpool, Liverpool, UK
Computer Technology Institute (CTI), Patras, Greece

Keywords

ϵ-Nash equilibria; ϵ-Well-supported Nash equilibria

Years and Authors of Summarized Original Work

2003; Lipton, Markakis, Mehta
2006; Daskalaskis, Mehta, Papadimitriou
2006; Kontogiannis, Panagopoulou, Spirakis

Problem Definition

Nash [15] introduced the concept of Nash equilibria in noncooperative games and proved that any game possesses at least one such equilibrium. A well-known algorithm for computing a Nash equilibrium of a 2-player game is the Lemke-Howson algorithm [13]; however, it has exponential worst-case running time in the number of available pure strategies [18].

Daskalakis et al. [5] showed that the problem of computing a Nash equilibrium in a game with 4 or more players is *PPAD*-complete; this result was later extended to games with 3 players [8]. Eventually, Chen and Deng [3] proved that the problem is *PPAD*-complete for 2-player games as well.

This fact emerged the computation of *approximate* Nash equilibria. There are several versions of approximate Nash equilibria that have been defined in the literature; however, the focus of this entry is on the notions of ϵ-Nash equilibrium and ϵ-well-supported Nash equilibrium. An ϵ-Nash equilibrium is a strategy profile such that no deviating player could achieve a payoff higher than the one that the specific profile gives her, plus ϵ. A stronger notion of approximate Nash equilibria is the *ϵ-well-supported Nash equilibria*; these are strategy profiles such that each player plays only approximately best-response pure strategies with nonzero probability. These are *additive* notions of approximate equilibria; the problem of computing approximate equilibria of bimatrix games using a relative notion of approximation is known to be *PPAD*-hard even for constant approximations.

Notation

For a $n \times 1$ vector x, denote by $x_1, \ldots, x_n$ the components of x and by x^T the transpose of x. Denote by e_i the column vector with 1 at the ith coordinate and 0 elsewhere. For an $n \times m$ matrix A, denote a_{ij} the element in the i-th row

and j-th column of A. Let $\mathbb{P}^n$ be the set of all probability vectors in n dimensions: $\mathbb{P}^n = \left\{ z \in \mathbb{R}^n_{\geq 0} : \sum_{i=1}^{n} z_i = 1 \right\}$.

Bimatrix Games

Bimatrix games [16] are a special case of 2-player games such that the payoff functions can be described by two real $n \times m$ matrices A and B. The n rows of A, B represent the *action set* of the first player (the *row player*), and the m columns represent the action set of the second player (the *column player*). Then, when the row player chooses action i and the column player chooses action j, the former gets payoff a_{ij}, while the latter gets payoff b_{ij}. Based on this, bimatrix games are denoted by $\Gamma = \langle A, B \rangle$.

A *strategy* for a player is any probability distribution on his/her set of actions. Therefore, a strategy for the row player can be expressed as a probability vector $x \in \mathbb{P}^n$, while a strategy for the column player can be expressed as a probability vector $y \in \mathbb{P}^m$. Each extreme point $e_i \in \mathbb{P}^n (e_j \in \mathbb{P}^m)$ that corresponds to the strategy assigning probability 1 to the i-th row (j-th column) is called a *pure strategy* for the row (column) player. A *strategy profile* (x, y) is a combination of (mixed in general) strategies, one for each player. In a given strategy profile (x, y), the players get *expected payoffs* $x^T Ay$ (row player) and $x^T By$ (column player).

If both payoff matrices belong to $[0, 1]^{m \times n}$, then the game is called a [0,1]-bimatrix (or else, *positively normalized*) game. The special case of bimatrix games in which all elements of the matrices belong to $\{0, 1\}$ is called a $\{0, 1\}$-bimatrix (or else, *win-lose*) game. A bimatrix game $\langle A, B \rangle$ is called *zero sum* if $B = -A$.

Approximate Nash Equilibria

Definition 1 (ϵ-Nash equilibrium) For any $\epsilon > 0$, a strategy profile (x, y) is an *ϵ-Nash equilibrium* for the $n \times m$ bimatrix game $\Gamma = \langle A, B \rangle$ if

1. For all pure strategies $i \in \{1, \ldots, n\}$ of the row player, $e_i^T Ay \leq x^T Ay + \epsilon$.
2. For all pure strategies $j \in \{1, \ldots, m\}$ of the column player, $x^T Be_j \leq x^T By + \epsilon$.

Definition 2 (ϵ-well-supported Nash equilibrium) For any $\epsilon > 0$, a strategy profile (x, y) is an *ϵ-well-supported Nash equilibrium* for the $n \times m$ bimatrix game $\Gamma = \langle A, B \rangle$ if

1. For all pure strategies $i \in \{1, \ldots, n\}$ of the row player,

$$x_i > 0 \Rightarrow e_i^T Ay \geq e_k^T Ay - \epsilon \ \forall k \in \{1, \ldots, n\}$$

2. For all pure strategies $j \in \{1, \ldots, m\}$ of the column player,

$$y_i > 0 \Rightarrow x^T Be_j \geq x^T Be_k - \epsilon \ \forall k \in \{1, \ldots, m\}.$$

Note that both notions of approximate equilibria are defined with respect to an additive error term ϵ. Although (exact) Nash equilibria are known not to be affected by any positive scaling, it is important to mention that approximate notions of Nash equilibria are indeed affected. Therefore, the commonly used assumption in the literature when referring to approximate Nash equilibria is that the bimatrix game is positively normalized, and this assumption is adopted in the present entry.

Key Results

The work of Althöfer [1] shows that, for *any* probability vector p, there exists a probability vector $\hat{P}$ with logarithmic supports, so that for a fixed matrix C, $\max_j |p^T Ce_j - \hat{p}^T Ce_j| \leq \epsilon$, for any constant $\epsilon > 0$. Exploiting this fact, the work of Lipton, Markakis, and Mehta [14] shows that, for any bimatrix game and for any *constant* $\epsilon > 0$, there exists an ϵ-Nash equilibrium with only logarithmic support (in the number n of available pure strategies). Consider a bimatrix game $\Gamma = \langle A, B \rangle$, and let (x, y) be a Nash equilibrium for Γ. Fix a positive integer k and form a multiset S_1 by sampling k times from the set of pure strate-

gies of the row player, independently at random according to the distribution x. Similarly, form a multiset S_2 by sampling k times from set of pure strategies of the column player according to y. Let $\hat{x}$ be the mixed strategy for the row player that assigns probability $1/k$ to each member of S_1 and 0 to all other pure strategies, and let $\hat{y}$ be the mixed strategy for the column player that assigns probability $1/k$ to each member of S_2 and 0 to all other pure strategies. Then, $\hat{x}$ and $\hat{y}$ are called *k-uniform* [14], and the following holds:

Theorem 1 ([14]) *For any Nash equilibrium (x, y) of a $n \times n$ bimatrix game and for every $\epsilon > 0$, there exists, for every $k \geq (12 \ln n)/\epsilon^2$, a pair of k-uniform strategies $\hat{x}, \hat{y}$ such that $(\hat{x}, \hat{y})$ is an ϵ-Nash equilibrium.*

This result directly yields a quasi-polynomial $(n^{O(\ln n)})$ algorithm for computing such an approximate equilibrium. Moreover, as pointed out in [1], no algorithm that examines supports smaller than about $\ln n$ can achieve an approximation better than $1/4$.

Theorem 2 ([4]) *The problem of computing a $1/n^{\Theta(1)}$-Nash equilibrium of a $n \times n$ bimatrix game is PPAD-complete.*

Theorem 2 asserts that, unless $PPAD \subseteq P$, there exists no fully polynomial time approximation scheme for computing equilibria in bimatrix games. However, this does not rule out the existence of a polynomial approximation scheme for computing an ϵ-Nash equilibrium when ϵ is an absolute constant, or even when $\epsilon = \Theta(1 - /\text{poly}(\ln n))$. Furthermore, as observed in [4], if the problem of finding an ϵ-Nash equilibrium were *PPAD*-complete when ϵ is an absolute constant, then, due to Theorem 1, all *PPAD* problems would be solved in quasi-polynomial time, which is unlikely to be the case.

Two concurrent and independent works [6, 11] were the first to make progress in providing ϵ-Nash equilibria and ϵ-well-supported Nash equilibria for bimatrix games and some *constant* $0 < \epsilon < 1$. In particular, the work of Kontogiannis, Panagopoulou, and Spirakis [11] proposes a simple linear-time algorithm for computing a $3/4$-Nash equilibrium for any bimatrix game:

Theorem 3 ([11]) *Consider any $n \times m$ bimatrix game $\Gamma = \langle A, B \rangle$, and let $a_{i1,j1} = \max_{i,j} a_{ij}$ and $b_{i2,j2} = \max_{i,j} b_{ij}$. Then the pair of strategies $(\hat{x}, \hat{y})$ where $\hat{x}_{i_1} = \hat{x}_{i_2} = \hat{y}_{j_1} = \hat{y}_{j_2} = 1/2$ is a $3/4$-Nash equilibrium for Γ.*

The above technique can be extended so as to obtain a parametrized, stronger approximation:

Theorem 4 ([11]) *Consider a $n \times m$ bimatrix game $\Gamma = \langle A, B \rangle$. Let $\lambda_1{}^*(\lambda_2{}^*)$ be the minimum, among all Nash equilibria of Γ, expected payoff for the row (column) player, and let $\lambda = \max\{\lambda_1{}^*, \lambda_2{}^*\}$. Then, there exists a $(2 + \lambda)/4$-Nash equilibrium that can be computed in time polynomial in n and m.*

The work of Daskalakis, Mehta, and Papadimitriou [6] provides a simple algorithm for computing a $1/2$-Nash equilibrium: Pick an arbitrary row for the row player, say row i. Let $j = \arg\max_{'j} b'_{ij}$. Let $k = \arg\max_{'k} a'_{kj}$. Thus, j is a best-response column for the column player to the row i, and k is a best-response row for the row player to the column j. Let $\hat{x} = 1/2e_i + 1/2e_k$ and $\hat{y} = e_j$, i.e., the row player plays row i or row k with probability $1/2$ each, while the column player plays column j with probability 1. Then:

Theorem 5 ([6]) *The strategy profile $(\hat{x}, \hat{y})$ is a $1/2$-Nash equilibrium.*

A polynomial construction (based on linear programming) of a 0.38-Nash equilibrium is presented in [7].

For the more demanding notion of well-supported approximate Nash equilibrium, Daskalakis, Mehta, and Papadimitriou [6] propose an algorithm, which, under a quite interesting and plausible graph theoretic conjecture, constructs in polynomial time a $5/6$-well-supported Nash equilibrium. However, the status of this conjecture is still unknown. In [6], it is also shown how to transform a [0,1]-bimatrix game to a $\{0, 1\}$-bimatrix game of the same size, so that each ϵ-well-supported Nash equilibrium of the resulting game is $(1+\epsilon)/2$-well-supported Nash equilibrium of the original game.

An algorithm given by Kontogiannis and Spirakis computes a $2/3$-well-supported Nash equilibrium in polynomial time [12]. Their methodology for attacking the problem is based on the solvability of zero-sum bimatrix games (via its connection to linear programming) and provides a 0.5-well-supported Nash equilibrium for win-lose games and a $2/3$-well-supported Nash equilibrium for normalized games. In [9], a polynomial-time algorithm computes an ε-well-supported Nash equilibrium with $\varepsilon < 2/3$, by extending the $2/3$-algorithm of Kontogiannis and Spirakis. In particular, it is shown that either the strategies generated by their algorithm can be tweaked to improve the approximation or that we can find a sub-game that resembles matching pennies, which again leads to a better approximation. This allows to construct a (2/3-0.004735)-well-supported Nash equilibrium in polynomial time.

Two new results improved the approximation status of ϵ-Nash equilibria:

Theorem 6 ([2]) *There is a polynomial time algorithm, based on linear programming, that provides an 0.36392-Nash equilibrium.*

The second result below, due to Tsaknakis and Spirakis, is the best till now. Based on local search, it establishes that any local minimum of a very natural map in the space of pairs of mixed strategies or its dual point in a certain minimax problem used for finding the local minimum constitutes a 0.3393-Nash equilibrium.

Theorem 7 ([19]) *There exists a polynomial time algorithm, based on the stationary points of a natural optimization problem, that provides an 0.3393-Nash Equilibrium.*

In [20], it is shown that the problem of computing a Nash equilibrium for 2-person games can be polynomially reduced to an indefinite quadratic programming problem involving the spectrum of the adjacency matrix of a strongly connected directed graph on n vertices, where n is the total number of players' strategies. Based on that, a new method is presented for computing approximate equilibria, and it is shown that its complexity is a function of the average spectral energy of the underlying graph. The implications of the strong connectedness properties on the energy and on the complexity of the method are discussed, and certain classes of graphs are described for which the method is a polynomial time approximation scheme (PTAS). The worst- case complexity is bounded by a subexponential function in the total number of strategies n.

Kannan and Theobald [10] investigate a hierarchy of bimatrix games $\langle A, B\rangle$ which results from restricting the rank of the matrix $A + B$ to be of fixed rank at most k. They propose a new model of ϵ-approximation for games of rank k and, using results from quadratic optimization, show that approximate Nash equilibria of constant rank games can be computed deterministically in time polynomial in $1/\epsilon$. Moreover, [10] provides a randomized approximation algorithm for certain quadratic optimization problems, which yields a randomized approximation algorithm for the Nash equilibrium problem. This randomized algorithm has similar time complexity as the deterministic one, but it has the possibility of finding an exact solution in polynomial time if a conjecture is valid. Finally, they present a polynomial time algorithm for *relative approximation* (with respect to the payoffs in an equilibrium) provided that the matrix $A + B$ has a nonnegative decomposition.

Applications

Noncooperative game theory and its main solution concept, i.e., the Nash equilibrium, have been extensively used to understand the phenomena observed when decision-makers interact and have been applied in many diverse academic fields, such as biology, economics, sociology, and artificial intelligence. Since however the computation of a Nash equilibrium is in general *PPAD*-complete, it is important to provide efficient algorithms for approximating a Nash equilibrium; the algorithms discussed in this entry are a first step towards this direction.

Recommended Reading

1. Althöfer I (1994) On sparse approximations to randomized strategies and convex combinations. Linear Algebr Appl 199:339–355
2. Bosse H, Byrka J, Markakis E (2007) New algorithms for approximate Nash equilibria in bimatrix games. In: Proceedings of the 3rd international workshop on internet and network economics (WINE 2007), San Diego, 12–14 Dec 2007. Lecture notes in computer science
3. Chen X, Deng X (2005) Settling the complexity of 2-player Nash-equilibrium. In: Proceedings of the 47th annual IEEE symposium on foundations of computer science (FOCS'06), Berkeley, 21–24 Oct 2005
4. Chen X, Deng X, Teng S-H (2006) Computing Nash equilibria: approximation and smoothed complexity. In: Proceedings of the 47th annual IEEE symposium on foundations of computer science (FOCS'06), Berkeley, 21–24 Oct 2006
5. Daskalakis C, Goldberg P, Papadimitriou C (2006) The complexity of computing a Nash equilibrium. In: Proceedings of the 38th annual ACM symposium on theory of computing (STOC'06), Seattle, 21–23 May 2006, pp 71–78
6. Daskalakis C, Mehta A, Papadimitriou C (2006) A note on approximate Nash equilibria. In: Proceedings of the 2nd workshop on internet and network economics (WINE'06), Patras, 15–17 Dec 2006, pp 297–306
7. Daskalakis C, Mehta A, Papadimitriou C (2007) Progress in approximate Nash equilibrium. In: Proceedings of the 8th ACM conference on electronic commerce (EC07), San Diego, 11–15 June 2007
8. Daskalakis C, Papadimitriou C (2005) Three-player games are hard. In: Electronic colloquium on computational complexity (ECCC TR 05-139)
9. Fearnley J, Goldberg PW, Savani R, Bjerre Sørensen T (2012) Approximate well-supported Nash equilibria below two-thirds. In: SAGT 2012, Barcelona, pp 108–119
10. Kannan R, Theobald T (2007) Games of fixed rank: a hierarchy of bimatrix games. In: Proceedings of the ACM-SIAM symposium on discrete algorithms, New Orleans, 7–9 Jan 2007
11. Kontogiannis S, Panagopoulou PN, Spirakis PG (2006) Polynomial algorithms for approximating Nash equilibria of bimatrix games. In: Proceedings of the 2nd workshop on internet and network economics (WINE'06), Patras, 15–17 Dec 2006, pp 286–296
12. Kontogiannis S, Spirakis PG (2010) Well supported approximate equilibria in bimatrix games. Algorithmica 57(4):653–667
13. Lemke CE, Howson JT (1964) Equilibrium points of bimatrix games. J Soc Indust Appl Math 12:413–423
14. Lipton RJ, Markakis E, Mehta A (2003) Playing large games using simple startegies. In: Proceedings of the 4th ACM conference on electronic commerce (EC'03), San Diego, 9–13 June 2003, pp 36–41
15. Nash J (1951) Noncooperative games. Ann Math 54:289–295
16. von Neumann J, Morgenstern O (1944) Theory of games and economic behavior. Princeton University Press, Princeton
17. Papadimitriou CH (1991) On inefficient proofs of existence and complexity classes. In: Proceedings of the 4th Czechoslovakian symposium on combinatorics 1990, Prachatice
18. Savani R, von Stengel B (2004) Exponentially many steps for finding a Nash equilibrium in a bimatrix game. In: Proceedings of the 45th annual IEEE symposium on foundations of computer science (FOCS'04), Rome, 17–19 Oct 2004, pp 258–267
19. Tsaknakis H, Spirakis P (2007) An optimization approach for approximate Nash equilibria. In: Proceedings of the 3rd international workshop on internet and network economics (WINE 2007). Lecture notes in computer science. Also in J Internet Math 5(4):365–382 (2008)
20. Tsaknakis H, Spirakis PG (2010) Practical and efficient approximations of Nash equilibria for win-lose games based on graph spectra. In: WINE 2010, Stanford, pp 378–390

Arbitrage in Frictional Foreign Exchange Market

Mao-cheng Cai[1] and Xiaotie Deng[2,3]
[1]Chinese Academy of Sciences, Institute of Systems Science, Beijing, China
[2]AIMS Laboratory (Algorithms-Agents-Data on Internet, Market, and Social Networks), Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[3]Department of Computer Science, City University of Hong Kong, Hong Kong, China

Keywords

Arbitrage; Complexity; Foreign exchange; Market

Years and Authors of Summarized Original Work

2003; Cai, Deng

Problem Definition

The simultaneous purchase and sale of the same securities, commodities, or foreign exchange in order to profit from a differential in the price. This usually takes place on different exchanges or marketplaces and is also known as a "riskless profit."

Arbitrage is, arguably, the most fundamental concept in finance. It is a state of the variables of financial instruments such that a riskless profit can be made, which is generally believed not in existence. The economist's argument for its nonexistence is that active investment agents will exploit any arbitrage opportunity in a financial market and thus will deplete it as soon as it may arise. Naturally, the speed at which such an arbitrage opportunity can be located and be taken advantage of is important for the profit-seeking investigators, which falls in the realm of analysis of algorithms and computational complexity.

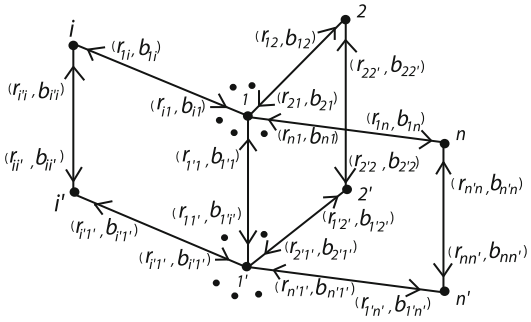
The identification of arbitrage states is, at frictionless foreign exchange market (a theoretical trading environment where all costs and restraints associated with transactions are nonexistent), not difficult at all and can be reduced to existence of arbitrage on three currencies (see [11]). In reality, friction does exist. Because of friction, it is possible that there exist arbitrage opportunities in the market but difficult to find it and to exploit it to eliminate it. Experimental results in foreign exchange markets showed that arbitrage does exist in reality. Examination of data from 10 markets over a 12-day period by Mavrides [11] revealed that a significant arbitrage opportunity exists. Some opportunities were observed to be persistent for a long time. The problem becomes worse at forward and future markets (in which future contracts in commodities are traded) coupled with covered interest rates, as observed by Abeysekera and Turtle [1] and Clinton [4]. An obvious interpretation is that the arbitrage opportunity was not immediately identified because of information asymmetry in the market. However, that is not the only factor. Both the time necessary to collect the market information (so that an arbitrage opportunity would be identified) and the time people (or computer programs) need to find the arbitrage transactions are important factors for eliminating arbitrage opportunities.

The computational complexity in identifying arbitrage, the level in difficulty measured by arithmetic operations, is different in different models of exchange systems. Therefore, to approximate an ideal exchange market, models with lower complexities should be preferred to those with higher complexities.

To model an exchange system, consider n foreign currencies: $N = \{1, 2, \ldots, n\}$. For each ordered pair (i, j), one may change one unit of currency i to r_{ij} units of currency j. Rate r_{ij} is the *exchange rate* from i to j. In an ideal market, the exchange rate holds for any amount that is exchanged. An *arbitrage opportunity* is a set of exchanges between pairs of currencies such that the net balance for each involved currency is nonnegative and there is at least one currency for which the net balance is positive. Under ideal market conditions, there is no arbitrage if and only if there is no arbitrage among any three currencies (see [11]).

Various types of *friction* can be easily modeled in such a system. Bid-offer spread may be expressed in the present mathematical format as $r_{ij}\, r_{ji} < 1$ for some $i, j \in N$. In addition, usually the traded amount is required to be in multiples of a fixed integer amount, hundreds, thousands, or millions. Moreover, different traders may bid or offer at different rates, and each for a limited amount. A more general model to describe these market *imperfections* will include, for pairs $i \neq j \in N$, l_{ij} different rates r_{ij}^k of exchanges from currency i to j up to b_{ij}^k units of currency i, $k = 1, \ldots, l_{ij}$, where l_{ij} is the number of different exchange rates from currency i to j.

A currency exchange market can be represented by a digraph $G = (V, E)$ with vertex set V and arc set E such that each vertex $i \in V$ represents currency i and each arc $a_{ij}^k \in E$ represents the currency exchange relation from i to j with rate r_{ij}^k and bound b_{ij}^k. Note that parallel arcs may occur for different exchange rates. Such a digraph is called an exchange digraph. Let $x = (x_{ij}^k)$ denote a currency exchange vector (Fig. 1).



Arbitrage in Frictional Foreign Exchange Market, Fig. 1 Digraph G_1

Problem 1 The existence of arbitrage in a frictional exchange market can be formulated as follows:

$$\sum_{j\neq i}\sum_{k=1}^{l_{ji}} \lfloor r_{ji}^k x_{ji}^k \rfloor - \sum_{j\neq i}\sum_{k=1}^{l_{ij}} x_{ij}^k \geq 0,\ i=1,\ldots,n, \tag{1}$$

at least one strict inequality holds

$$0 \leq x_{ij}^k \leq b_{ij}^k,\ 1 \leq k \leq l_{ij},\ 1 \leq i \neq j \leq n, \tag{2}$$

$$x_{ij}^k \text{ is integer},\ 1 \leq k \leq l_{ij},\ 1 \leq i \neq j \leq n. \tag{3}$$

Note that the first term in the right-hand side of (1) is the revenue at currency i by selling other currencies and the second term is the expense at currency i by buying other currencies.

The corresponding optimization problem is

Problem 2 The maximum arbitrage problem in a frictional foreign exchange market with bid-ask spreads, bound, and integrality constraints is the following integer linear programming (P):

$$\text{maximize} \sum_{i=1}^{n} w_i \sum_{j\neq i}\left(\sum_{k=1}^{l_{ji}} \lfloor r_{ji}^k x_{ji}^k \rfloor - \sum_{k=1}^{l_{ij}} x_{ij}^k\right)$$

subject to

$$\sum_{j\neq i}\left(\sum_{k=1}^{l_{ji}} \lfloor r_{ji}^k x_{ji}^k \rfloor - \sum_{k=1}^{l_{ij}} x_{ij}^k\right) \geq 0,\ i=1,\ldots,n, \tag{4}$$

$$0 \leq x_{ij}^k \leq b_{ij}^k,\ 1 \leq k \leq l_{ij},\ 1 \leq i \neq j \leq n, \tag{5}$$

$$x_{ij}^k \text{ is integer},\ 1 \leq k \leq l_{ij},\ 1 \leq i \neq j \leq n, \tag{6}$$

where $w_i \geq 0$ is a given weight for currency $i, i = 1, 2, \ldots, n$, with at least one $w_i > 0$.

Finally, consider another

Problem 3 In order to eliminate arbitrage, how many transactions and arcs in a exchange digraph have to be used for the currency exchange system?

Key Results

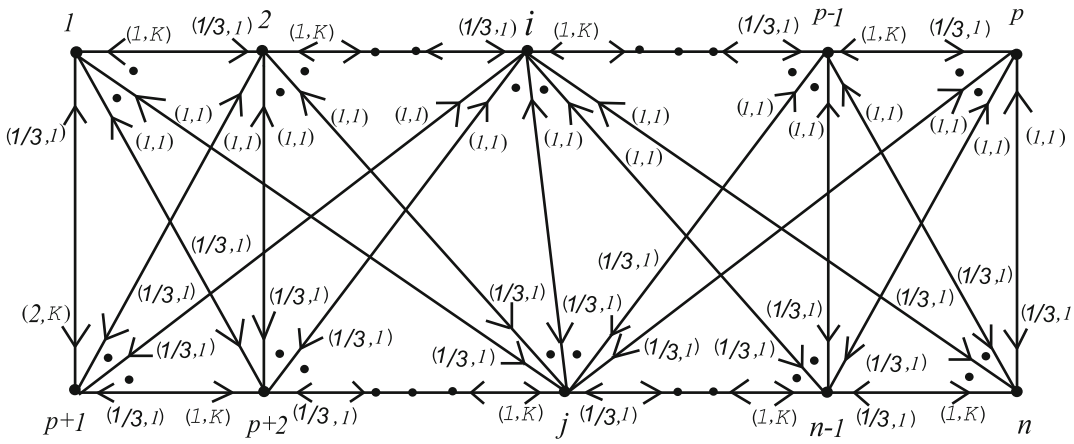
A decision problem is called nondeterministic polynomial (*NP* for short) if its solution (if one exists) can be guessed and verified in polynomial time; nondeterministic means that no particular rule is followed to make the guess. If a problem is *NP* and all other *NP* problems are polynomial-time reducible to it, the problem is *NP*-complete. And a problem is called *NP*-hard if every other problem in *NP* is polynomial-time reducible to it (Fig. 2).

Theorem 1 *It is NP-complete to determine whether there exists arbitrage in a frictional foreign exchange market with bid-ask spreads, bound, and integrality constraints even if all* $l_{ij} = 1$.

Then, a further inapproximability result is obtained.

Theorem 2 *There exists fixed* $\epsilon > 0$ *such that approximating* (P) *within a factor of* n^{ϵ} *is NP-hard even for any of the following two special cases:*

(P_1) *all* $l_{ij} = 1$ *and* $w_i = 1$.
(P_2) *all* $l_{ij} = 1$ *and all but one* $w_i = 0$.



Arbitrage in Frictional Foreign Exchange Market, Fig. 2 Digraph G_2

Now, consider two polynomially solvable special cases when the number of currencies is constant or the exchange digraph is star shaped (a digraph is star shaped if all arcs have a common vertex).

Theorem 3 *There are polynomial time algorithms for* (P) *when the number of currencies is constant.*

Theorem 4 *It is polynomially solvable to find the maximum revenue at the center currency of arbitrage in a frictional foreign exchange market with bid-ask spread, bound, and integrality constraints when the exchange digraph is star shaped.*

However, if the exchange digraph is the coalescence of a star-shaped exchange digraph and its copy, shown by Digraph G_1, then the problem becomes *NP*-complete.

Theorem 5 *It is NP-complete to decide whether there exists arbitrage in a frictional foreign exchange market with bid-ask spreads, bound, and integrality constraints even if its exchange digraph is coalescent.*

Finally, an answer to Problem 3 is as follows

Theorem 6 *There is an exchange digraph of order n such that at least* $\lfloor n/2 \rfloor \lceil n/2 \rceil - 1$ *transactions and at least* $n^2/4 + n - 3$ *arcs are in need to bring the system back to non-arbitrage states.*

For instance, consider the currency exchange market corresponding to digraph $G_2 = (V, E)$, where the number of currencies is $n = |V|$, $p = \lfloor n/ \rfloor$, and $K = n^2$.

Set

$$C = \{a_{ij} \in E | 1 \le i \le p, p+1 \le j \le n\}$$
$$\cup \{a_{1(p+1)}\} \backslash \{a_{(p+1)1}\} \cup \{a_{i(i-1)} | 2 \le i \le p\}$$
$$\cup \{a_{i(i+1)} | p+1 \le i \le n-1\}.$$

Then, $|C| = \lfloor n/2 \rfloor \lceil n/2 \rceil + n - 2 = |E| / > n^2/4 + n - 3$. It follows easily from the rates and bounds that each arc in C has to be used to eliminate arbitrage. And $\lfloor n/2 \rfloor \lceil n/2 \rceil - 1$ transactions corresponding to $\{a_{ij} \in E | 1 \le i \le p, p+1 \le j \le n\} \backslash \{a_{(p+1)1}\}$ are in need to bring the system back to non-arbitrage states.

Applications

The present results show that different foreign exchange systems exhibit quite different computational complexities. They may shed new light on how monetary system models are adopted and evolved in reality. In addition, it provides with a computational complexity point of view to the understanding of the now fast growing Internet electronic exchange markets.

Open Problems

The dynamic models involving in both spot markets (in which goods are sold for cash and delivered immediately) and futures markets are the most interesting ones. To develop good approximation algorithms for such general models would be important. In addition, it is also important to identify special market models for which polynomial time algorithms are possible even with future markets. Another interesting paradox in this line of study is why friction constraints that make arbitrage difficult are not always eliminated in reality.

Recommended Reading

1. Abeysekera SP, Turtle HJ (1995) Long-run relations in exchange markets: a test of covered interest parity. J Financ Res 18(4):431–447
2. Ausiello G, Crescenzi P, Gambosi G, Kann V, Marchetti-Spaccamela A, Protasi M (1999) Complexity and approximation: combinatorial optimization problems and their approximability properties. Springer, Berlin
3. Cai M, Deng X (2003) Approximation and computation of arbitrage in frictional foreign exchange market. Electron Notes Theor Comput Sci 78:1–10
4. Clinton K (1988) Transactions costs and covered interest arbitrage: theory and evidence. J Polit Econ 96(2):358–370
5. Deng X, Papadimitriou C (1994) On the complexity of cooperative game solution concepts. Math Oper Res 19(2):257–266
6. Deng X, Li ZF, Wang S (2002) Computational complexity of arbitrage in frictional security market. Int J Found Comput Sci 13(5):681–684
7. Deng X, Papadimitriou C, Safra S (2003) On the complexity of price equilibria. J Comput Syst Sci 67(2):311–324
8. Garey MR, Johnson DS (1979) Computers and intractability: a guide of the theory of *NP*-completeness. Freeman, San Francisco
9. Jones CK (2001) A network model for foreign exchange arbitrage, hedging and speculation. Int J Theor Appl Finance 4(6):837–852
10. Lenstra HW Jr (1983) Integer programming with a fixed number of variables. Math Oper Res 8(4):538–548
11. Mavrides M (1992) Triangular arbitrage in the foreign exchange market – inefficiencies, technology and investment opportunities. Quorum Books, London
12. Megiddo N (1978) Computational complexity and the game theory approach to cost allocation for a tree. Math Oper Res 3:189–196
13. Mundell RA (1981) Gold would serve into the 21st century. Wall Str J, Sep 30, p 33
14. Mundell RA (2000) Currency areas, exchange rate systems, and international monetary reform, paper delivered at Universidad del CEMA, Buenos Aires. http://www.robertmundell.net/pdf/Currency. Accessed 17 Apr 2000
15. Zhang S, Xu C, Deng X (2002) Dynamic arbitrage-free asset pricing with proportional transaction costs. Math Finance 12(1):89–97

Arithmetic Coding for Data Compression

Paul G. Howard[1] and Jeffrey Scott Vitter[2]
[1]Akamai Technologies, Cambridge, MA, USA
[2]University of Kansas, Lawrence, KS, USA

Keywords

Entropy coding; Statistical data compression

Years and Authors of Summarized Original Work

1994; Howard, Vitter

Problem Definition

Often it is desirable to encode a sequence of data efficiently to minimize the number of bits required to transmit or store the sequence. The sequence may be a file or message consisting of *symbols* (or *letters* or *characters*) taken from a fixed input alphabet, but more generally the sequence can be thought of as consisting of *events*, each taken from its own input set. Statistical data compression is concerned with encoding the data in a way that makes use of probability estimates of the events. Lossless compression has the property that the input sequence can be reconstructed exactly from the encoded sequence. Arithmetic

Arithmetic Coding for Data Compression, Table 1 Comparison of codes for Huffman coding, Hu-Tucker coding, and arithmetic coding for a sample 5-symbol alphabet

Symbol e_k	Prob.		Huffman		Hu-Tucker		Arithmetic
	p_k	$-\log_2 p_k$	Code	Length	Code	Length	Length
a	0.04	4.644	**1111**	4	**000**	3	4.644
b	0.18	2.474	**110**	3	**001**	3	2.474
c	0.43	1.218	**0**	1	**01**	2	1.218
d	0.15	2.737	**1110**	4	**10**	2	2.737
e	0.20	2.322	**10**	2	**11**	2	2.322

coding is a nearly optimal statistical coding technique that can produce a lossless encoding.

Problem (statistical data compression) INPUT: A sequence of m events $a_1, a_2, \ldots, a_m$. The ith event a_i is taken from a set of n distinct possible events $e_{i,1}, e_{i,2}, \ldots, e_{i,n}$, with an accurate assessment of the probability distribution P_i of the events. The distributions P_i need not be the same for each event a_i.

OUTPUT: A succinct encoding of the events that can be decoded to recover exactly the original sequence of events.

The goal is to achieve optimal or near-optimal encoding length. Shannon [10] proved that the smallest possible expected number of bits needed to encode the ith event is the *entropy* of P_i, denoted by

$$H(P_i) = \sum_{k=1}^{n} -p_{i,k} \log_2 p_{i,k}$$

where $p_{i,k}$ is the probability that e_k occurs as the ith event. An optimal code outputs $-\log_2 p$ bits to encode an event whose probability of occurrence is p.

The well-known Huffman codes [6] are optimal only among *prefix* (or *instantaneous*) codes, that is, those in which the encoding of one event can be decoded before encoding has begun for the next event. Hu-Tucker codes are prefix codes similar to Huffman codes and are derived using a similar algorithm, with the added constraint that coded messages preserve the ordering of original messages.

When an instantaneous code is not needed, as is often the case, arithmetic coding provides a number of benefits, primarily by relaxing the constraint that the code lengths must be integers: (1) The code length is optimal ($-\log_2 p$ bits for an event with probability p), even when probabilities are not integer powers of $\frac{1}{2}$. (2) There is no loss of coding efficiency even for events with probability close to 1. (3) It is trivial to handle probability distributions that change from event to event. (4) The input message to output message ordering correspondence of Hu-Tucker coding can be obtained with minimal extra effort.

As an example, consider a 5-symbol input alphabet. Symbol probabilities, codes, and code lengths are given in Table 1.

The average code length is 2.13 bits per input symbol for the Huffman code, 2.22 bits per symbol for the Hu-Tucker code, and 2.03 bits per symbol for arithmetic coding.

Key Results

In theory, arithmetic codes assign one "codeword" to each possible input sequence. The codewords consist of half-open subintervals of the half-open unit interval [0,1) and are expressed by specifying enough bits to distinguish the subinterval corresponding to the actual sequence from all other possible subintervals. Shorter codes correspond to larger subintervals and thus more probable input sequences. In practice, the subinterval is refined incrementally using the probabilities of the individual events, with bits being output as soon as they are known. Arithmetic codes almost always give better compression than prefix codes, but they lack the direct correspondence between

the events in the input sequence and bits or groups of bits in the coded output file.

The algorithm for encoding a file using arithmetic coding works conceptually as follows:

1. The "current interval" $[L, H)$ is initialized to [0,1).
2. For each event in the file, two steps are performed:
   (a) Subdivide the current interval into subintervals, one for each possible event. The size of an event's subinterval is proportional to the estimated probability that the event will be the next event in the file, according to the model of the input.
   (b) Select the subinterval corresponding to the event that actually occurs next and make it the new current interval.
3. Output enough bits to distinguish the final current interval from all other possible final intervals.

The length of the final subinterval is clearly equal to the product of the probabilities of the individual events, which is the probability p of the particular overall sequence of events. It can be shown that $\lfloor -\log_2 p \rfloor + 2$ bits are enough to distinguish the file from all other possible files.

For finite-length files, it is necessary to indicate the end of the file. In arithmetic coding, this can be done easily by introducing a special low-probability event that can be injected into the input stream at any point. This adds only $O(\log m)$ bits to the encoded length of an m-symbol file.

In step 2, one needs to compute only the subinterval corresponding to the event a_i that actually occurs. To do this, it is convenient to use two "cumulative" probabilities: the cumulative probability $P_C = \sum_{k=1}^{i-1} p_k$ and the next-cumulative probability $P_N = P_C + p_i = \sum_{k=1}^{i-1} p_k$. The new subinterval is $[L + P_C(H - L), L + P_N(H - L))$. The need to maintain and supply cumulative probabilities requires the model to have a sophisticated data structure, such as that of Moffat [7], especially when many more than two events are possible.

Modeling

The goal of modeling for statistical data compression is to provide probability information to the coder. The modeling process consists of structural and probability estimation components; each may be adaptive (starting from a neutral model, gradually build up the structure and probabilities based on the events encountered), semi-adaptive (specify an initial model that describes the events to be encountered in the data and then modify the model during coding so that it describes only the events yet to be coded), or static (specify an initial model and use it without modification during coding).

In addition there are two strategies for probability estimation. The first is to estimate each event's probability individually based on its frequency within the input sequence. The second is to estimate the probabilities collectively, assuming a probability distribution of a particular form and estimating the parameters of the distribution, either directly or indirectly. For direct estimation, the data can yield an estimate of the parameter (the variance, for instance). For indirect estimation [4], one can start with a small number of possible distributions and compute the code length that would be obtained with each; the one with the smallest code length is selected. This method is very general and can be used even for distributions from different families, without common parameters.

Arithmetic coding is often applied to text compression. The events are the symbols in the text file, and the model consists of the probabilities of the symbols considered in some context. The simplest model uses the overall frequencies of the symbols in the file as the probabilities; this is a zero-order Markov model, and its entropy is denoted H_0. The probabilities can be estimated adaptively starting with counts of 1 for all symbols and incrementing after each symbol is coded, or the symbol counts can be coded before coding the file itself and either modified during coding (a decrementing semi-adaptive code) or left unchanged (a static code). In all cases, the

code length is independent of the order of the symbols in the file.

Theorem 1 *For all input files, the code length L_A of an adaptive code with initial 1-weights is the same as the code length L_{SD} of the semi-adaptive decrementing code plus the code length L_M of the input model encoded assuming that all symbol distributions are equally likely. This code length is less than $L_S = mH_0 + L_M$, the code length of a static code with the same input model. In other words, $L_A = L_{SD} + L_M < mH_0 + L_M = L_S$.*

It is possible to obtain considerably better text compression by using higher-order Markov models. Cleary and Witten [2] were the first to do this with their PPM method. PPM requires adaptive modeling and coding of probabilities close to 1 and makes heavy use of arithmetic coding.

Implementation Issues

Incremental Output

The basic implementation of arithmetic coding described above has two major difficulties: the shrinking current interval requires the use of high-precision arithmetic, and no output is produced until the entire file has been read. The most straightforward solution to both of these problems is to output each leading bit as soon as it is known and then to double the length of the current interval so that it reflects only the unknown part of the final interval. Witten, Neal, and Cleary [11] add a clever mechanism for preventing the current interval from shrinking too much when the endpoints are close to $\frac{1}{2}$ but straddle $\frac{1}{2}$. In that case, one does not yet know the next output bit, but whatever it is, the *following* bit will have the opposite value; one can merely keep track of that fact and expand the current interval symmetrically about $\frac{1}{2}$. This follow-on procedure may be repeated any number of times, so the current interval size is always strictly longer than $\frac{1}{4}$.

Before [11] other mechanisms for incremental transmission and fixed precision arithmetic were developed through the years by a number of researchers beginning with Pasco [8]. The bit-stuffing idea of Langdon and others at IBM [9] that limits the propagation of carries in the additions serves a function similar to that of the follow-on procedure described above.

Use of Integer Arithmetic

In practice, the arithmetic can be done by storing the endpoints of the current interval as sufficiently large integers rather than in floating point or exact rational numbers. Instead of starting with the real interval [0,1), start with the integer interval $[0,N)$, N invariably being a power of 2. The subdivision process involves selecting non-overlapping integer intervals (of length at least 1) with lengths approximately proportional to the counts.

Limited-Precision Arithmetic Coding

Arithmetic coding as it is usually implemented is slow because of the multiplications (and in some implementations, divisions) required in subdividing the current interval according to the probability information. Since small errors in probability estimates cause very small increases in code length, introducing approximations into the arithmetic coding process in a controlled way can improve coding speed without significantly degrading compression performance. In the Q-Coder work at IBM [9], the time-consuming multiplications are replaced by additions and shifts, and low-order bits are ignored.

Howard and Vitter [3] describe a different approach to approximate arithmetic coding. The fractional bits characteristic of arithmetic coding are stored as state information in the coder. The idea, called *quasi-arithmetic coding*, is to reduce the number of possible states and replace arithmetic operations by table lookups; the lookup tables can be precomputed.

The number of possible states (after applying the interval expansion procedure) of an arithmetic coder using the integer interval $[0,N)$ is $3N^2/16$. The obvious way to reduce the number of states in order to make lookup tables practicable is to reduce N. Binary quasi-arithmetic coding causes an insignificant increase in the code length compared with pure arithmetic coding.

Theorem 2 *In a quasi-arithmetic coder based on full interval [0,N), using correct probability estimates, and excluding very large and very small probabilities, the number of bits per input event by which the average code length obtained by the quasi-arithmetic coder exceeds that of an exact arithmetic coder is at most*

$$\frac{4}{\ln 2}\left(\log_2 \frac{2}{e \ln 2}\right)\frac{1}{N} + O\left(\frac{1}{N^2}\right) \approx \frac{0.497}{N} + O\left(\frac{1}{N^2}\right),$$

and the fraction by which the average code length obtained by the quasi-arithmetic coder exceeds that of an exact arithmetic coder is at most

$$\left(\log_2 \frac{2}{e \ln 2}\right)\frac{1}{\log_2 N} + O\left(\frac{1}{(\log N)^2}\right) \approx \frac{0.0861}{\log_2 N} + O\left(\frac{1}{(\log N)^2}\right).$$

General-purpose algorithms for parallel encoding and decoding using both Huffman and quasi-arithmetic coding are given in [5].

Applications

Arithmetic coding can be used in most applications of data compression. Its main usefulness is in obtaining maximum compression in conjunction with an adaptive model or when the probability of one event is close to 1. Arithmetic coding has been used heavily in text compression. It has also been used in image compression in the JPEG international standards for image compression and is an essential part of the JBIG international standards for bilevel image compression. Many fast implementations of arithmetic coding, especially for a two-symbol alphabet, are covered by patents; considerable effort has been expended in adjusting the basic algorithm to avoid infringing those patents.

Open Problems

The technical problems with arithmetic coding itself have been completely solved. The remaining unresolved issues are concerned with modeling, in which the issue is how to decompose an input data set into a sequence of events, so that the set of events possible at each point in the data set can be described by a probability distribution suitable for input into the coder. The modeling issues are entirely application-specific.

Experimental Results

Some experimental results for the Calgary and Canterbury corpora are summarized in a report by Arnold and Bell [1].

Data Sets

Among the most widely used data sets suitable for research in arithmetic coding are the Calgary Corpus and Canterbury Corpus (corpus.canterbury.ac.nz) and the Pizza&Chili Corpus (pizzachili.dcc.uchile.cl or http://pizzachili.di.unipi.it).

URL to Code

A number of implementations of arithmetic coding are available on The Data Compression Resource on the Internet, www.data-compression.info/Algorithms/AC/.

Cross-References

▸ Burrows-Wheeler Transform
▸ Huffman Coding

Recommended Reading

1. Arnold R, Bell T (1997) A corpus for the evaluation of lossless compression algorithms. In: Proceedings of the IEEE data compression conference, Snowbird, Mar 1997, pp 201–210

2. Cleary JG, Witten IH (1984) Data compression using adaptive coding and partial string matching. IEEE Trans Commun COM-32:396–402
3. Howard PG, Vitter JS (1992) Practical implementations of arithmetic coding. In: Storer JA (ed) Images and text compression. Kluwer Academic, Norwell
4. Howard PG, Vitter JS (1993) Fast and efficient lossless image compression. In: Proceedings of the IEEE data compression conference, Snowbird, Mar 1993, pp 351–360
5. Howard PG, Vitter JS (1996) Parallel lossless image compression using Huffman and arithmetic coding. Inf Process Lett 59:65–73
6. Huffman DA (1952) A method for the construction of minimum redundancy codes. Proc Inst Radio Eng 40:1098–1101
7. Moffat A (1999) An improved data structure for cumulative probability tables. Softw Pract Exp 29:647–659
8. Pasco R (1976) Source coding algorithms for fast data compression. Ph.D. thesis, Stanford University
9. Pennebaker WB, Mitchell JL, Langdon GG, Arps RB (1988) An overview of the basic principles of the Q-coder adaptive binary arithmetic coder. IBM J Res Dev 32:717–726
10. Shannon CE (1948) A mathematical theory of communication. Bell Syst Tech J 27:398–403
11. Witten IH, Neal RM, Cleary JG (1987) Arithmetic coding for data compression. Commun ACM 30:520–540

Assignment Problem

Samir Khuller
Computer Science Department, University of Maryland, College Park, MD, USA

Keywords

Weighted bipartite matching

Years and Authors of Summarized Original Work

1955; Kuhn
1957; Munkres

Problem Definition

Assume that a complete bipartite graph, $G(X, Y, X \times Y)$, with weights $w(x, y)$ assigned to every edge (x, y) is given. A matching M is a subset of edges so that no two edges in M have a common vertex. A perfect matching is one in which all the nodes are matched. Assume that $|X| = |Y| = n$. The **weighted matching problem** is to find a matching with the greatest total weight, where $w(M) = \sum_{e \in M} w(e)$. Since G is a complete bipartite graph, it has a perfect matching. An algorithm that solves the weighted matching problem is due to Kuhn [4] and Munkres [6]. Assume that all edge weights are non-negative.

Key Results

Define a *feasible vertex labeling* ℓ as a mapping from the set of vertices in G to the reals, where

$$\ell(x) + \ell(y) \geq w(x, y).$$

Call $\ell(x)$ the label of vertex x. It is easy to compute a feasible vertex labeling as follows:

$$\forall y \in Y \ell(y) = 0$$

and

$$\forall x \in X \quad \ell(x) = \max_{y \in Y} w(x, y).$$

Define the **equality subgraph**, G_ℓ, to be the spanning subgraph of G, which includes all vertices of G but only those edges (x, y) that have weights such that

$$w(x, y) = \ell(x) + \ell(y).$$

The connection between equality subgraphs and maximum-weighted matchings is provided by the following theorem:

Theorem 1 *If the equality subgraph, G_ℓ, has a perfect matching, M^*, then M^* is a maximum-weighted matching in G.*

In fact, note that the sum of the labels is an upper bound on the weight of the maximum-weighted perfect matching. The algorithm eventually finds a matching and a feasible labeling

such that the weight of the matching is equal to the sum of all the labels.

High-Level Description

The above theorem is the basis of an algorithm for finding a maximum-weighted matching in a complete bipartite graph. Starting with a feasible labeling, compute the equality subgraph, and then find a maximum matching in this subgraph (here, one can ignore weights on edges). If the matching found is perfect, the process is done. If it is not perfect, more edges are added to the equality subgraph by revising the vertex labels. After adding edges to the equality subgraph, either the size of the matching goes up (an augmenting path is found) or the Hungarian tree continues to grow. (This is the structure of explored edges when one starts BFS simultaneously from all free nodes in S. When one reaches a matched node in T, one only explores the matched edge; however, all edges incident to nodes in S are explored.) In the former case, the phase terminates, and a new phase starts (since the matching size has gone up). In the latter case, the Hungarian tree grows by adding new nodes to it, and clearly, this cannot happen more than n times.

Let S be the set of free nodes in X. Grow Hungarian trees from each node in S. Let T be the nodes in Y encountered in the search for an augmenting path from nodes in S. Add all nodes from X that are encountered in the search to S.

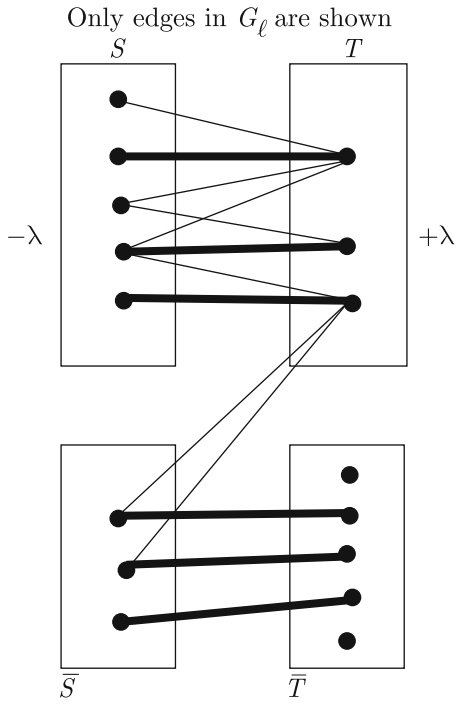
Note the following about this algorithm:

$$\overline{S} = X \backslash S.$$

$$\overline{T} = Y \backslash T.$$

$$|S| > |T|.$$

There are no edges from S to $\overline{T}$ since this would imply that one did not grow the Hungarian trees completely. As the Hungarian trees are grown in G_ℓ, alternate nodes in the search are placed into S and T. To revise the labels, take the labels in S, and start decreasing them uniformly (say, by λ), and at the same time, increase the labels in T by λ. This ensures that the edges from S to T do not leave the equality subgraph (Fig. 1).



Assignment Problem, Fig. 1 Sets S and T as maintained by the algorithm

As the labels in S are decreased, edges (in G) from S to $\overline{T}$ will potentially enter the equality subgraph, G_ℓ. As we increase λ, at some point in time, an edge enters the equality subgraph. This is when one stops and updates the Hungarian tree. If the node from $\overline{T}$ added to T is matched to a node in $\overline{S}$, both these nodes are moved to S and T, which yields a larger Hungarian tree. If the node from $\overline{T}$ is free, an augmenting path is found, and the phase is complete. One phase consists of those steps taken between increases in the size of the matching. There are at most n phases, where n is the number of vertices in G (since in each phase the size of the matching increases by 1). Within each phase, the size of the Hungarian tree is increased at most n times. It is clear that in $O(n^2)$ time, one can figure out which edge from S to $\overline{T}$ is the first to enter the equality subgraph (one simply scans all the edges). This yields an $O(n^4)$ bound on the total running time. How to implement it in $O(n^3)$ time is now shown.

More Efficient Implementation

Define the slack of an edge as follows:

$$slack\,(x, y) = \ell\,(x) + \ell\,(y) - w\,(x, y)\,.$$

Then,

$$\lambda = \min_{x \in S, y \in \overline{T}} slack\,(x, y)\,.$$

Naively, the calculation of λ requires $O(n^2)$ time. For every vertex $y \in \overline{T}$, keep track of the edge with the smallest slack, i.e.,

$$slack\,[y] = \min_{x \in S} slack\,(x, y)\,.$$

The computation of $slack[y]$ (for all $y \in \overline{T}$) requires $O(n^2)$ time at the start of a phase. As the phase progresses, it is easy to update all the *slack* values in $O(n)$ time since all of them change by the same amount (the labels of the vertices in S are going down uniformly). Whenever a node u is moved from $\overline{S}$ to S, one must recompute the slacks of the nodes in $\overline{T}$, requiring $O(n)$ time. But a node can be moved from $\overline{S}$ to S at most n times.

Thus, each phase can be implemented in $O(n^2)$ time. Since there are n phases, this gives a running time of $O(n^3)$. For sparse graphs, there is a way to implement the algorithm in $O(n(m + n \log n))$ time using min cost flows [1], where m is the number of edges.

Applications

There are numerous applications of biparitite matching, for example, scheduling unit-length jobs with integer release times and deadlines even with time-dependent penalties.

Open Problems

Obtaining a linear, or close to linear, time algorithm.

Recommended Reading

Several books on combinatorial optimization describe algorithms for weighted bipartite matching (see [2,5]). See also Gabow's paper [3].

1. Ahuja R, Magnanti T, Orlin J (1993) Network flows: theory, algorithms and applications. Prentice Hall, Englewood Cliffs
2. Cook W, Cunningham W, Pulleyblank W, Schrijver A (1998) Combinatorial Optimization. Wiley, New York
3. Gabow H (1990) Data structures for weighted matching and nearest common ancestors with linking. In: Symposium on discrete algorithms, San Francisco, pp 434–443
4. Kuhn H (1955) The Hungarian method for the assignment problem. Naval Res Logist Q 2:83–97
5. Lawler E (1976) Combinatorial optimization: networks and matroids. Holt, Rinehart and Winston, New York
6. Munkres J (1957) Algorithms for the assignment and transportation problems. J Soc Ind Appl Math 5:32–38

Asynchronous Consensus Impossibility

Maurice Herlihy
Department of Computer Science,
Brown University, Providence, RI, USA

Keywords

Agreement; Wait-free consensus

Years and Authors of Summarized Original Work

1985; Fischer, Lynch, Paterson

Problem Definition

Consider a distributed system consisting of a set of *processes* that communicate by sending and receiving messages. The network is a multiset of messages, where each message is addressed to some process. A process is a state machine that can take three kinds of *steps*.

- In a *send* step, a process places a message in the network.
- In a *receive* step, a process A either reads and removes from the network a message addressed to A, or it reads a distinguished *null* value, leaving the network unchanged. If a message addressed to A is placed in the network, and if A subsequently performs an infinite number of receive steps, then A will eventually receive that message.
- In a *computation* state, a process changes state without communicating with any other process.

Processes are *asynchronous*: there is no bound on their relative speeds. Processes can *crash*: they can simply halt and take no more steps. This article considers executions in which at most one process crashes.

In the *consensus* problem, each process starts with a private *input* value, communicates with the others, and then halts with a *decision* value. These values must satisfy the following properties:

- *Agreement:* all processes' decision values must agree.
- *Validity:* every decision value must be some process' input.
- *Termination:* every non-fault process must decide in a finite number of steps.

Fischer, Lynch, and Paterson showed that there is no protocol that solves consensus in any asynchronous message-passing system where even a single process can fail. This result is one of the most influential results in Distributed Computing, laying the foundations for a number of subsequent research efforts.

Terminology

Without loss of generality, one can restrict attention to *binary* consensus, where the inputs are 0 or 1. A *protocol state* consists of the states of the processes and the multiset of messages in transit in the network. An *initial state* is a protocol state before any process has moved, and a *final state* is a protocol state after all processes have finished. The *decision value* of any final state is the value decided by all processes in that state.

Any terminating protocol's set of possible states forms a tree, where each node represents a possible protocol state, and each edge represents a possible step by some process. Because the protocol must terminate, the tree is finite. Each leaf node represents a final protocol state with decision value either 0 or 1.

A *bivalent* protocol state is one in which the eventual decision value is not yet fixed. From any bivalent state, there is an execution in which the eventual decision value is 0, and another in which it is 1. A *univalent* protocol state is one in which the outcome is fixed. Every execution starting from a univalent state decides the same value. A *1-valent* protocol state is univalent with eventual decision value 1, and similarly for a *0-valent* state.

A protocol state is *critical* if

- it is bivalent, and
- if any process takes a step, the protocol state becomes univalent.

Key Results

Lemma 1 *Every consensus protocol has a bivalent initial state.*

Proof Assume, by way of contradiction, that there exists a consensus protocol for $(n+1)$ threads $A_0, \cdots, A_n$ in which every initial state is univalent. Let s_i be the initial state where processes $A_i, \cdots, A_n$ have input 0 and $A_0, \ldots, A_{i-1}$ have input 1. Clearly, s_0 is 0-valent: all processes have input 0, so all must decide 0 by the validity condition. If s_i is 0-valent, so is s_{i+1}. These states differ only in the input to process $A_i : 0$ in s_i, and 1 in s_{i+1}. Any execution starting from s_i in which A_i halts before taking any steps is indistinguishable from an execution starting from s_{i+1} in which A_i halts before taking any steps. Since processes must decide 0 in the first execution, they must decide 1 in the second. Since there

is one execution starting from s_{i+1} that decides 0, and since s_{i+1} is univalent by hypothesis, s_{i+1} is 0-valent. It follows that the state s_{n+1}, in which all processes start with input 1, is 0-valent, a contradiction. □

Lemma 2 *Every consensus protocol has a critical state.*

Proof by contradiction. By Lemma 1, the protocol has a bivalent initial state. Start the protocol in this state. Repeatedly choose a process whose next step leaves the protocol in a bivalent state, and let that process take a step. Either the protocol runs forever, violating the termination condition, or the protocol eventually enters a critical state.□

Theorem 1 *There is no consensus protocol for an asynchronous message-passing system where a single process can crash.*

Proof Assume by way of contradiction that such a protocol exists. Run the protocol until it reaches a critical state s. There must be two processes A and B such that A's next step carries the protocol to a 0-valent state, and B's next step carries the protocol to a 1-valent state.

Starting from s, let s_A be the state reached if A takes the first step, s_B if B takes the first step, s_{AB} if A takes a step followed by B, and so on. States s_A and s_{AB} are 0-valent, while s_B and s_{BA} are 1-valent. The rest is a case analysis.

Of all the possible pairs of steps A and B could be about to execute, most of them *commute*: states s_{AB} and s_{BA} are identical, which is a contradiction because they have different valences.

The only pair of steps that do not commute occurs when A is about to send a message to B (or vice versa). Let s_{AB} be the state resulting if A sends a message to B and B then receives it, and let s_{BA} be the state resulting if B receives a different message (or *null*) and then A sends its message to B. Note that every process other than B has the same local state in s_{AB} and s_{BA}. Consider an execution starting from s_{AB} in which every process other than B takes steps in round-robin order. Because s_{AB} is 0-valent, they will eventually decide 0. Next, consider an execution starting from s_{BA} in which every process other than B takes steps in round-robin order. Because s_{BA} is 1-valent, they will eventually decide 1. But all processes other than B have the same local states at the end of each execution, so they cannot decide different values, a contradiction. □

In the proof of this theorem, and in the proofs of the preceding lemmas, we construct scenarios where at most a single process is delayed. As a result, this impossibility result holds for any system where a single process can fail undetectably.

Applications

The consensus problem is a key tool for understanding the power of various asynchronous models of computation.

Open Problems

There are many open problems concerning the solvability of consensus in other models, or with restrictions on inputs.

Related Work

The original paper by Fischer, Lynch, and Paterson [8] is still a model of clarity.

Many researchers have examined alternative models of computation in which consensus can be solved. Dolev, Dwork, and Stockmeyer [5] examine a variety of alternative message-passing models, identifying the precise assumptions needed to make consensus possible. Dwork, Lynch, and Stockmeyer [6] derive upper and lower bounds for a semi-synchronous model where there is an upper and lower bound on message delivery time. Ben-Or [1] showed that introducing randomization makes consensus possible in an asynchronous message-passing system. Chandra and Toueg [3] showed that consensus becomes possible if in the presence of an oracle that can (unreliably) detect when a process has crashed. Each of the papers cited

here has inspired many follow-up papers. A good place to start is the excellent survey by Fich and Ruppert [7].

A protocol is *wait-free* if it tolerates failures by all but one of the participants. A concurrent object implementation is *linearizable* if each method call seems to take effect instantaneously at some point between the method's invocation and response. Herlihy [9] showed that shared-memory objects can each be assigned a *consensus number*, which is the maximum number of processes for which there exists a wait-free consensus protocol using a combination of read-write memory and the objects in question. Consensus numbers induce an infinite hierarchy on objects, where (simplifying somewhat) higher objects are more powerful than lower objects. In a system of n or more concurrent processes, it is impossible to construct a lock-free implementation of an object with consensus number n from an object with a lower consensus number. On the other hand, any object with consensus number n is *universal* in a system of n or fewer processes: it can be used to construct a wait-free linearizable implementation of any object.

In 1990, Chaudhuri [4] introduced the *k-set agreement* problem (sometimes called *k-set consensus*, which generalizes consensus by allowing k or fewer distinct decision values to be chosen. In particular, 1-set agreement is consensus. The question whether k-set agreement can be solved in asynchronous message-passing models was open for several years, until three independent groups [2, 10, 11] showed that no protocol exists.

Cross-References

- ▸ Linearizability
- ▸ Topology Approach in Distributed Computing

Recommended Reading

1. Ben-Or M (1983) Another advantage of free choice (extended abstract): completely asynchronous agreement protocols. In: PODC '83: proceedings of the second annual ACM symposium on principles of distributed computing. ACM Press, New York, pp 27–30
2. Borowsky E, Gafni E (1993) Generalized FLP impossibility result for t-resilient asynchronous computations. In: Proceedings of the 1993 ACM symposium on theory of computing, May 1993, pp 206–215
3. Chandra TD, Toueg S (1996) Unreliable failure detectors for reliable distributed systems. J ACM 43(2):225–267
4. Chaudhuri S (1990) Agreement is harder than consensus: set consensus problems in totally asynchronous systems. In: Proceedings of the ninth annual ACM symposium on principles of distributed computing, Aug 1990, pp 311–324
5. Chandhuri S (1993) More choices allow more faults: set consensus problems in totally asynchronous systems. Inf Comput 105(1):132–158
6. Dwork C, Lynch N, Stockmeyer L (1988) Consensus in the presence of partial synchrony. J ACM 35(2):288–323
7. Fich F, Ruppert E (2003) Hundreds of impossibility results for distributed computing. Distrib Comput 16(2–3):121–163
8. Fischer M, Lynch N, Paterson M (1985) Impossibility of distributed consensus with one faulty process. J ACM 32(2):374–382
9. Herlihy M (1991) Wait-free synchronization. ACM Trans Program Lang Syst (TOPLAS) 13(1):124–149
10. Herlihy M, Shavit N (1999) The topological structure of asynchronous computability. J ACM 46(6):858–923
11. Saks ME, Zaharoglou F (2000) Wait-free k-set agreement is impossible: the topology of public knowledge. SIAM J Comput 29(5):1449–1483

Atomic Broadcast

Xavier Défago
School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, Japan

Keywords

Atomic multicast; Total order broadcast; Total order multicast

Years and Authors of Summarized Original Work

1995; Cristian, Aghili, Strong, Dolev

Problem Definition

The problem is concerned with allowing a set of processes to concurrently broadcast messages while ensuring that all destinations consistently deliver them in the exact same sequence, in spite of the possible presence of a number of faulty processes.

The work of Cristian, Aghili, Strong, and Dolev [7] considers the problem of atomic broadcast in a system with approximately synchronized clocks and bounded transmission and processing delays. They present successive extensions of an algorithm to tolerate a bounded number of omission, timing, or Byzantine failures, respectively.

Related Work

The work presented in this entry originally appeared as a widely distributed conference contribution [6], over a decade before being published in a journal [7], at which time the work was well-known in the research community. Since there was no significant change in the algorithms, the historical context considered here is hence with respect to the earlier version.

Lamport [11] proposed one of the first published algorithms to solve the problem of ordering broadcast messages in a distributed systems. That algorithm, presented as the core of a mutual exclusion algorithm, operates in a fully asynchronous system (i.e., a system in which there are no bounds on processor speed or communication delays), but does not tolerate failures. Although the algorithms presented here rely on physical clocks rather than Lamport's logical clocks, the principle used for ordering messages is essentially the same: message carry a timestamp of their sending time; messages are delivered in increasing order of the timestamp, using the sending processor name for messages with equal timestamps.

At roughly the same period as the initial publication of the work of Cristian et al. [6], Chang and Maxemchuck [3] proposed an atomic broadcast protocol based on a token passing protocol, and tolerant to crash failures of processors. Also, Carr [1] proposed the Tandem global update protocol, tolerant to crash failures of processors.

Cristian [5] later proposed an extension to the omission-tolerant algorithm presented here, under the assumption that the communication system consists of $f + 1$ independent broadcast channels (where f is the maximal number of faulty processors). Compared with the more general protocol presented here, its extension generates considerably fewer messages.

Since the work of Cristian, Aghili, Strong, and Dolev [7], much has been published on the problem of atomic broadcast (and its numerous variants). For further reading, Défago, Schiper, and Urbán [8] surveyed more than sixty different algorithms to solve the problem, classifying them into five different classes and twelve variants. That survey also reviews many alternative definitions and references about two hundred articles related to this subject. This is still a very active research area, with many new results being published each year.

Hadzilacos and Toueg [10] provide a systematic classification of specifications for variants of atomic broadcast as well as other broadcast problems, such as reliable broadcast, FIFO broadcast, or causal broadcast.

Chandra and Toueg [2] proved the equivalence between atomic broadcast and the *consensus* problem. Thus, any application solved by a consensus can also be solved by atomic broadcast and vice-versa. Similarly, impossibility results apply equally to both problems. For instance, it is well-known that consensus, thus atomic broadcast, cannot be solved deterministically in an asynchronous system with the presence of a faulty process [9].

Notations and Assumptions

The system G consists of n distributed processors and m point-to-point communication links. A link does not necessarily exists between every pair of processors, but it is assumed that the communication network remains connected even in the face of faults (whether processors or links). All processors have distinct names and there exists a total order on them (e.g., lexicographic order).

A component (link or processor) is said to be *correct* if its behavior is consistent with its specification, and *faulty* otherwise. The paper considers three classes of component failures, namely, omission, timing, and Byzantine failures.

- An *omission* failure occurs when the faulty component fails to provide the specified output (e.g., loss of a message).
- A *timing* failure occurs when the faulty component omits a specified output, or provides it either too early or too late.
- A *Byzantine* failure [12] occurs when the component does not behave according to its specification, for instance, by providing output different from the one specified. In particular, the paper considers authentication-detectable Byzantine failures, that is, ones that are detectable using a message authentication protocol, such as error correction codes or digital signatures.

Each processor p has access to a local clock C_p with the properties that (1) two separate clock readings yield different values, and (2) clocks are ε-synchronized, meaning that, at any real time t, the deviation in readings of the clocks of any two processors p and q is at most ε.

In addition, transmission and processing delays, as measured on the clock of a correct processor, are bounded by a known constant δ. This bound accounts not only for delays in transmission and processing, but also for delays due to scheduling, overload, clock drift or adjustments. This is called a synchronous system model.

The diffusion time $d\delta$ is the time necessary to propagate information to all correct processes, in a surviving network of diameter d with the presence of a most π processor failures and λ link failures.

Problem Definition

The problem of atomic broadcast is defined in a synchronous system model as a broadcast primitive which satisfies the following three properties: atomicity, order, and termination.

Problem 1 (Atomic broadcast)
INPUT: A stream of messages broadcast by n concurrent processors, some of which may be faulty.
OUTPUT: The messages delivered in sequence, with the following properties:

1. *Atomicity*: if any correct processor delivers an update at time U on its clock, then that update was initiated by some processor and is delivered by each correct processor at time U on its clock.
2. *Order*: all updates delivered by correct processors are delivered in the same order by each correct processor.
3. *Termination*: every update whose broadcast is initiated by a correct processor at time T on its clock is delivered at all correct processors at time $T + \Delta$ on their clock.

Nowadays, problem definitions for atomic broadcast that do not explicitly refer to physical time are often preferred. Many variants of time-free definitions are reviewed by Hadzilacos and Toueg [10] and Défago et al. [8]. One such alternate definition is presented below, with the terminology adapted to the context of this entry.

Problem 2 (Total order broadcast)
INPUT: A stream of messages broadcast by n concurrent processors, some of which may be faulty.
OUTPUT: The messages delivered in sequence, with the following properties:

1. *Validity*: if a correct processor broadcasts a message m, then it eventually delivers m.
2. *Uniform agreement*: if a processor delivers a message m, then all correct processors eventually deliver m.
3. *Uniform integrity*: for any message m, every processor delivers m at most once, and only if m was previously broadcast by its sending processor.
4. *Gap-free uniform total order*: if some processor delivers message m' after message m, then a processor delivers m' only after it has delivered m.

Key Results

The paper presents three algorithms for solving the problem of atomic broadcast, each under an increasingly demanding failure model, namely, omission, timing, and Byzantine failures. Each protocol is actually an extension of the previous one.

All three protocols are based on a classical flooding, or information diffusion, algorithm [14]. Every message carries its initiation timestamp T, the name of the initiating processor s, and an update σ. A message is then uniquely identified by (s, T). Then, the basic protocol is simple. Each processor logs every message it receives until it is delivered. When it receives a message that was never seen before, it forwards that message to all other neighbor processors.

Atomic Broadcast for Omission Failures

The first atomic broadcast protocol, supporting omission failures, considers a termination time Δ_o as follows.

$$\Delta_o = \pi\delta + d\delta + \varepsilon\,. \tag{1}$$

The delivery deadline $T + \Delta_o$ is the time by which a processor can be sure that it has received copies of every message with timestamp T (or earlier) that could have been received by some correct process.

The protocol then works as follows. When a processor initiates an atomic broadcast, it propagates that message, similar to the diffusion algorithm described above. The main exception is that every message received after the local clock exceeds the delivery deadline of that message, is discarded. Then, at local time $T + \Delta_o$, a processor delivers all messages timestamped with T, in order of the name of the sending processor. Finally, it discards all copies of the messages from its logs.

Atomic Broadcast for Timing Failures

The second protocol extends the first one by introducing a hop count (i.e., a counter incremented each time a message is relayed) to the messages. With this information, each relaying processor can determine when a message is timely, that is, if a message timestamped T with hop count h is received at time U then the following condition must hold.

$$T - h\varepsilon < U < T + h(\delta + \varepsilon)\,. \tag{2}$$

Before relaying a message, each processor checks the acceptance test above and discard the message if it does not satisfy it. The termination time Δ_t of the protocol for timing failures is as follows.

$$\Delta_t = \pi(\delta + \varepsilon) + d\delta + \varepsilon\,. \tag{3}$$

The authors point out that discarding early messages is not necessary for correctness, but ensures that correct processors keep messages in their log for a bounded amount of time.

Atomic Broadcast for Byzantine Failures

Given some text, every processor is assumed to be able to generate a signature for it, that cannot be faked by other processors. Furthermore, every processor knows the name of every other processors in the network, and has the ability to verify the authenticity of their signature.

Under the above assumptions, the third protocol extends the second one by adding signatures to the messages. To prevent a Byzantine processor (or link) from tampering with the hop count, a message is co-signed by every processor that relays it. For instance, a message signed by k processors $p_1, \ldots, p_k$ is as follows.

$$(relayed, \ldots (relayed, (first, T, \sigma, p_1, s_1), p_2, s_2)\,, \ldots p_k, s_k)$$

Where σ is the update, T the timestamp, p_1 the message source, and s_i the signature generated by processor p_i. Any message for which one of the signature cannot be authenticated is simply discarded. Also, if several updates initiated by the same processor p carry the same timestamp, this indicates that p is faulty and the corresponding updates are discarded. The remainder of the protocol is the same as the second one, where

the number of hops is given by the number of signatures. The termination time Δ_b is also as follows.

$$\Delta_b = \pi(\delta + \varepsilon) + d\delta + \varepsilon\,. \tag{4}$$

The authors insist however that, in this case, the transmission time δ must be considerably larger than in the previous case, since it must account for the time spent in generating and verifying the digital signatures; usually a costly operation.

Bounds

In addition to the three protocols presented above and their correctness, Cristian et al. [7] prove the following two lower bounds on the termination time of atomic broadcast protocols.

Theorem 1 *If the communication network G requires x steps, then any atomic broadcast protocol tolerant of up to π processor and λ link omission failures has a termination time of at least $x\delta + \varepsilon$.*

Theorem 2 *Any atomic broadcast protocol for a Hamiltonian network with n processors that tolerate $n-2$ authentication-detectable Byzantine processor failures cannot have a termination time smaller than $(n-1)(\delta + \varepsilon)$.*

Applications

The main motivation for considering this problem is its use as the cornerstone for ensuring fault-tolerance through process replication. In particular, the authors consider a *synchronous replicated storage*, which they define as a distributed and resilient storage system that displays the same content at every correct physical processor at any clock time. Using atomic broadcast to deliver updates ensures that all updates are applied at all correct processors in the same order. Thus, provided that the replicas are initially consistent, they will remain consistent. This technique, called *state-machine replication* [11, 13] or also *active replication*, is widely used in practice as a means of supporting fault-tolerance in distributed systems.

In contrast, Cristian et al. [7] consider atomic broadcast in a *synchronous* system with bounded transmission and processing delays. Their work was motivated by the implementation of a highly-available replicated storage system, with tightly coupled processors running a real-time operating system.

Atomic broadcast has been used as a support for the replication of running processes in real-time systems or, with the problem reformulated to isolate explicit timing requirements, has also been used as a support for fault-tolerance and replication in many group communication toolkits (see survey of Chockler et al. [4]).

In addition, atomic broadcast has been used for the replication of database systems, as a means to reduce the synchronization between the replicas. Wiesmann and Schiper [15] have compared different database replication and transaction processing approaches based on atomic broadcast, showing interesting performance gains.

Cross-References

- Asynchronous Consensus Impossibility
- Causal Order, Logical Clocks, State Machine Replication
- Clock Synchronization
- Failure Detectors

Recommended Reading

1. Carr R (1985) The Tandem global update protocol. Tandem Syst Rev 1:74–85
2. Chandra TD, Toueg S (1996) Unreliable failure detectors for reliable distributed systems. J ACM 43:225–267
3. Chang J-M, Maxemchuk NF (1984) Reliable broadcast protocols. ACM Trans Comput Syst 2:251–273
4. Chockler G, Keidar I, Vitenberg R (2001) Group communication specifications: a comprehensive study. ACM Comput Surv 33:427–469
5. Cristian F (1990) Synchronous atomic broadcast for redundant broadcast channels. Real-Time Syst 2:195–212

6. Cristian F, Aghili H, Strong R, Dolev D (1985) Atomic broadcast: from simple message diffusion to Byzantine agreement. In: Proceedings of the 15th international symposium on fault-tolerant computing (FTCS-15), Ann Arbor, June 1985. IEEE Computer Society Press, pp 200—206
7. Cristian F, Aghili H, Strong R, Dolev D (1995) Atomic broadcast: from simple message diffusion to Byzantine agreement. Inform Comput 118:158–179
8. Défago X, Schiper A, Urbán P (2004) Total order broadcast and multicast algorithms: taxonomy and survey. ACM Comput Surv 36:372–421
9. Fischer MJ, Lynch NA, Paterson MS (1985) Impossibility of distributed consensus with one faulty process. J ACM 32:374–382
10. Hadzilacos V, Toueg S (1993) Fault-tolerant broadcasts and related problems. In: Mullender S (ed) Distributed systems, 2nd edn. ACM Press Books/Addison-Wesley, pp 97–146, Extended version appeared as Cornell Univ. TR 94-1425
11. Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21:558–565
12. Lamport L, Shostak R, Pease M (1982) The Byzantine generals problem. ACM Trans Prog Lang Syst 4:382–401
13. Schneider FB (1990) Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput Surv 22:299–319
14. Segall A (1983) Distributed network protocols. IEEE Trans Inf Theory 29:23–35
15. Wiesmann M, Schiper A (2005) Comparison of database replication techniques based on total order broadcast. IEEE Trans Knowl Data Eng 17:551–566

Attribute-Efficient Learning

Jyrki Kivinen
Department of Computer Science, University of Helsinki, Helsinki, Finland

Keywords

Learning with irrelevant attributes

Years and Authors of Summarized Original Work

1987; Littlestone

Problem Definition

Given here is a basic formulation using the *online mistake-bound* model, which was used by Littlestone [9] in his seminal work.

Fix a class C of Boolean functions over n variables. To start a learning scenario, a *target function* $f_* \in C$ is chosen but not revealed to the *learning algorithm*. Learning then proceeds in a sequence of *trials*. At trial t, an input $\boldsymbol{x}_t \in \{0,1\}^n$ is first given to the learning algorithm. The learning algorithm then produces its *prediction* $\hat{y}_t$, which is its guess as to the unknown value $f_*(x_t)$. The correct value $y_t = f_*(x_t)$ is then revealed to the learner. If $y_t \neq \hat{y}_t$, the learning algorithm made a *mistake*. The learning algorithm learns C with mistake-bound m, if the number of mistakes never exceeds m, no matter how many trials are made and how f_* and $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ are chosen.

Variable (or attribute) X_i is *relevant* for function $f : \{0,1\}^n \to \{0,1\}$ if $f(x_1, \ldots, x_i, \ldots, x_n) \neq f(x_1, \ldots, 1 - x_i, \ldots, x_n)$ holds for some $\vec{x} \in \{0,1\}^n$. Suppose now that for some $k \leq n$, every function $f \in C$ has at most k relevant variables. It is said that a learning algorithm learns class C *attribute-efficiently*, if it learns C with a mistake-bound polynomial in k and $\log n$. Additionally, the computation time for each trial is usually required to be polynomial in n.

Key Results

The main part of current research of attribute-efficient learning stems from Littlestone's Winnow algorithm [9]. The basic version of Winnow maintains a weight vector $\boldsymbol{w}_t = (w_{t,1}, \ldots, w_{t,n}) \in \mathbb{R}^n$. The prediction for input $\boldsymbol{x}_t \in \{0,1\}^n$ is given by

$$\hat{y}_t = \operatorname{sign}\left(\sum_{i=1}^{n} w_{t,i} x_{t,i} - \theta\right)$$

where θ is a parameter of the algorithm. Initially $\boldsymbol{w}_1 = (1, \ldots, 1)$, and after trial t, each component $w_{t,i}$ is updated according to

$$w_{t+1,i} = \begin{cases} \alpha w_{t,i} & \text{if } y_t = 1,\ \hat{y}_t = 0 \text{ and } x_{t,i} = 1 \\ w_{t,i}/\alpha & \text{if } y_t = 0,\ \hat{y}_t = 1 \text{ and } x_{t,i} = 1 \\ w_{t,i} & \text{otherwise} \end{cases} \tag{1}$$

where $\alpha > 1$ is a learning rate parameter.

Littlestone's basic result is that with a suitable choice of θ and α, Winnow learns the class of monotone k-literal disjunctions with mistake-bound $O(k \log n)$. Since the algorithm changes its weights only when a mistake occurs, this bound also guarantees that the weights remain small enough for computation times to remain polynomial in n. With simple transformations, Winnow also yields attribute-efficient learning algorithms for general disjunctions and conjunctions. Various subclasses of DNF formulas and decision lists [8] can be learned, too.

Winnow is quite robust against noise, i.e., errors in input data. This is extremely important for practical applications. Remove now the assumption about a target function $f_* \in C$ satisfying $y_t = f_*(x_t)$ for all t. Define *attribute error* of a pair $(\boldsymbol{x}, y)$ with respect to a function f as the minimum Hamming distance between $\boldsymbol{x}$ and $\boldsymbol{x}'$ such that $f(\boldsymbol{x}') = y$. The attribute error of a sequence of trials with respect to f is the sum of attribute errors of the individual pairs $(\boldsymbol{x}_t, y_t)$. Assuming the sequence of trials has attribute error at most A with respect to some k-literal disjunction, Auer and Warmuth [1] show that Winnow makes $O(A+k \log n)$ mistakes. The noisy scenario can also be analyzed in terms of *hinge loss* [5].

The update rule (1) has served as a model for a whole family of *multiplicative update algorithms*. For example, Kivinen and Warmuth [7] introduce the exponentiated gradient algorithm, which is essentially Winnow modified for continuous-valued prediction, and show how it can be motivated by a relative entropy minimization principle.

Consider a function class C where each function can be encoded using $O(p(k) \log n)$ bits for some polynomial p. An example would be Boolean formulas with k relevant variables, when the size of the formula is restricted to $p(k)$ ignoring the size taken by the variables. The cardinality of C is then $|C| = 2^{O(p(k) \log n)}$. The classical halving algorithm (see [9] for discussion and references) learns any class consisting of m Boolean functions with mistake-bound $\log_2 m$ and would thus provide an attribute-efficient algorithm for such a class C. However, the running time would not be polynomial. Another serious drawback would be that the halving algorithm does not tolerate any noise. Interestingly, a multiplicative update similar to (1) has been used in Littlestone and Warmuth's weighted majority algorithm [10], and also Vovk's aggregating algorithm [14], to produce a noise-tolerant generalization of the halving algorithm.

Attribute-efficient learning has also been studied in other learning models than the mistake-bound model, such as Probably Approximately Correct learning [4], learning with uniform distribution [12], and learning with membership queries [3]. The idea has been further developed into learning with a potentially infinite number of attributes [2].

Applications

Attribute-efficient algorithms for simple function classes have a potentially interesting application as a component in learning more complex function classes. For example, any monotone k-term DNF formula over variables $x_1, \ldots, x_n$ can be represented as a monotone k-literal disjunction over 2^n variables z_A, where z_A is defined as $z_A = \prod_{i \in A} x_i$ for $A \subseteq \{1, \ldots, n\}$. Running Winnow with the transformed inputs $\boldsymbol{z} \in \{0,1\}^{2^n}$ would give a mistake-bound $O(k \log 2^n) = O(kn)$. Unfortunately the running time would be linear in 2^n, at least for a naive implementation. Khardon et al. [6] provide discouraging computational hardness results for this potential application.

Online learning algorithms have a natural application domain in signal processing. In this setting, the sender emits a true signal y_t at time t, for $t = 1, 2, 3, \ldots$. At some later time $(t + d)$, a receiver receives a signal z_t, which is a sum

of the original signal y_t and various echoes of earlier signals $y_{t'}$, $t' < t$, all distorted by random noise. The task is to recover the true signal y_t based on received signals $z_t, z_{t-1}, \ldots, z_{t-l}$ over some time window l. Currently attribute-efficient algorithms are not used for such tasks, but see [11] for preliminary results.

Attribute-efficient learning algorithms are similar in spirit to statistical methods that find sparse models. In particular, statistical algorithms that use L_1 regularization are closely related to multiplicative algorithms such as winnow and exponentiated gradient. In contrast, more classical L_2 regularization leads to algorithms that are not attribute-efficient [13].

Cross-References

▸ Learning DNF Formulas

Recommended Reading

1. Auer P, Warmuth MK (1998) Tracking the best disjunction. Mach Learn 32(2):127–150
2. Blum A, Hellerstein L, Littlestone N (1995) Learning in the presence of finitely or infinitely many irrelevant attributes. J Comput Syst Sci 50(1):32–40
3. Bshouty N, Hellerstein L (1998) Attribute-efficient learning in query and mistake-bound models. J Comput Syst Sci 56(3):310–319
4. Dhagat A, Hellerstein L (1994) PAC learning with irrelevant attributes. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe. IEEE Computer Society, Los Alamitos, pp 64–74
5. Gentile C, Warmuth MK (1999) Linear hinge loss and average margin. In: Kearns MJ, Solla SA, Cohn DA (eds) Advances in Neural Information Processing Systems, vol 11. MIT, Cambridge, pp 225–231
6. Khardon R, Roth D, Servedio RA (2005) Efficiency versus convergence of boolean kernels for on-line learning algorithms. J Artif Intell Res 24:341–356
7. Kivinen J, Warmuth MK (1997) Exponentiated gradient versus gradient descent for linear predictors. Inf Comput 132(1):1–64
8. Klivans AR, Servedio RA (2006) Toward attribute efficient learning of decision lists and parities. J Mach Learn Res 7:587–602
9. Littlestone N (1988) Learning quickly when irrelevant attributes abound: a new linear threshold algorithm. Mach Learn 2(4):285–318
10. Littlestone N, Warmuth MK (1994) The weighted majority algorithm. Inf Comput 108(2): 212–261
11. Martin RK, Sethares WA, Williamson RC, Johnson CR Jr (2002) Exploiting sparsity in adaptive filters. IEEE Trans Signal Process 50(8):1883–1894
12. Mossel E, O'Donnell R, Servedio RA (2004) Learning functions of k relevant variables. J Comput Syst Sci 69(3):421–434
13. Ng AY (2004) Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In: Greiner R, Schuurmans D (eds) Proceedings of the 21st International Conference on Machine Learning, Banff. The International Machine Learning Society, Princeton, pp 615–622
14. Vovk V (1990) Aggregating strategies. In: Fulk M, Case J (eds) Proceedings of the 3rd Annual Workshop on Computational Learning Theory, Rochester. Morgan Kaufmann, San Mateo, pp 371–383

Automated Search Tree Generation

Falk Hüffner
Department of Math and Computer Science, University of Jena, Jena, Germany

Keywords

Automated proofs of upper bounds on the running time of splitting algorithms

Years and Authors of Summarized Original Work

2004; Gramm, Guo, Hüffner, Niedermeier

Problem Definition

This problem is concerned with the automated development and analysis of search tree algorithms. Search tree algorithms are a popular way to find optimal solutions to NP-complete problems. (For ease of presentation, only decision problems are considered; adaption to optimization problems is straightforward.) The idea is to recursively solve several smaller instances in such a way that at least one branch is a yes-

instance if and only if the original instance is. Typically, this is done by trying all possibilities to contribute to a solution certificate for a small part of the input, yielding a small local modification of the instance in each branch.

For example, consider the NP-complete CLUSTER EDITING problem: can a given graph be modified by adding or deleting up to k edges such that the resulting graph is a *cluster graph*, that is, a graph that is a disjoint union of cliques? To give a search tree algorithm for CLUSTER EDITING, one can use the fact that cluster graphs are exactly the graphs that do not contain a P_3 (a path of 3 vertices) as an induced subgraph. One can thus solve CLUSTER EDITING by finding a P_3 and splitting it into 3 branches: delete the first edge, delete the second edge, or add the missing edge. By this characterization, whenever there is no P_3 found, one already has a cluster graph. The original instance has a solution with k modifications if and only if at least one of the branches has a solution with $k-1$ modifications.

Analysis

For NP-complete problems, the running time of a search tree algorithm only depends on the size of the search tree up to a polynomial factor , which depends on the number of branches and the reduction in size of each branch. If the algorithm solves a problem of size s and calls itself recursively for problems of sizes $s-d_1,\ldots,s-d_i$, then $(d_1,\ldots,d_i)$ is called the *branching vector* of this recursion. It is known that the size of the search tree is then $O(\alpha^s)$, where the *branching number* α is the only positive real root of the *characteristic polynomial*

$$z^d - z^{d-d_1} - \cdots - z^{d-d_i}\ , \qquad (1)$$

where $d = \max\{d_1,\ldots,d_i\}$. For the simple CLUSTER EDITING search tree algorithm and the size measure k, the branching vector is (1, 1, 1) and the branching number is 3, meaning that the running time is up to a polynomial factor $O(3^k)$.

Case Distinction

Often, one can obtain better running times by distinguishing a number of cases of instances, and giving a specialized branching for each case. The overall running time is then determined by the branching number of the worst case. Several publications obtain such algorithms by hand (e.g., a search tree of size $O(2.27^k)$ for CLUSTER EDITING [4]); the topic of this work is how to automate this. That is, the problem is the following:

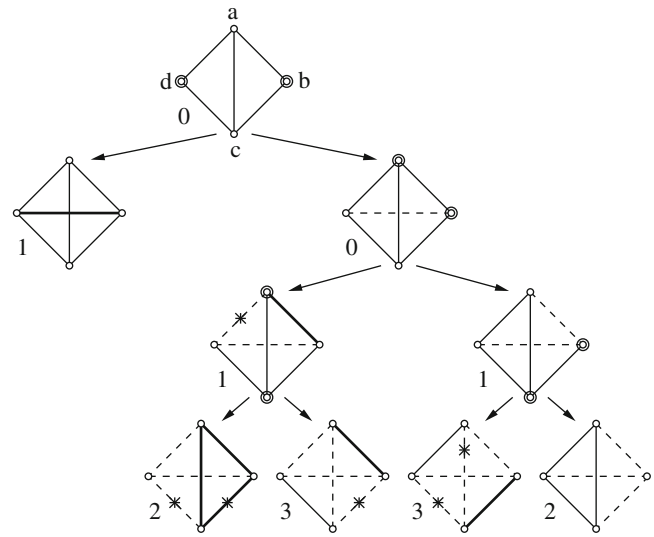
Problem 1 (Fast Search Tree Algorithm)
INPUT: An NP-hard problem $\mathcal{P}$ and a size measure $s(I)$ of an instance I of $\mathcal{P}$ where instances I with $s(I)=0$ can be solved in polynomial time.
OUTPUT: A partition of the instance set of $\mathcal{P}$ into *cases*, and for each case a branching such that the maximum branching number over all branchings is as small as possible.

Note that this problem definition is somewhat vague; in particular, to be useful, the case an instance belongs to must be recognizable quickly. It is also not clear whether an optimal search tree algorithm exists; conceivably, the branching number can be continuously reduced by increasingly complicated case distinctions.

Key Results

Gramm et al. [3] describe a method to obtain fast search tree algorithms for CLUSTER EDITING and related problems, where the size measure is the number of editing operations k. To get a case distinction, a number of subgraphs are enumerated such that each instance is known to contain at least one of these subgraphs. It is next described how to obtain a branching for a particular case.

A standard way of systematically obtaining specialized branchings for instance cases is to use a combination of *basic branching* and *data reduction rules*. Basic branching is typically a very simple branching technique, and data reduction rules replace an instance with a smaller, solution-equivalent instance in polynomial time. Applying this to CLUSTER EDITING first requires a small modification of the problem: one considers an *annotated* version, where an edge can be marked as *permanent* and a non-edge can be marked as *forbidden*. Any such annotated vertex pair cannot



Automated Search Tree Generation, Fig. 1 Branching for a CLUSTER EDITING case using only basic branching on vertex pairs (*double circles*), and applications of the reduction rules (*asterisks*). Permanent edges are marked *bold*, forbidden edges *dashed*. The *numbers* next to the subgraphs state the change of the problem size k. The branching vector is (1, 2, 3, 3, 2), corresponding to a search tree size of $O(2.27^k)$

be edited anymore. For a pair of vertices, the basic branching then branches into two cases: permanent or forbidden (one of these options will require an editing operation). The reduction rules are: if two permanent edges are adjacent, the third edge of the triangle they induce must also be permanent; and if a permanent and a forbidden edge are adjacent, the third edge of the triangle they induce must be forbidden.

Figure 1 shows an example branching derived in this way.

Using a refined method of searching the space for all possible cases and to distinguish all branchings for a case, Gramm et al. [3] derive a number of search tree algorithms for graph modification problems.

Applications

Gramm et al. [3] apply the automated generation of search tree algorithms to several graph modification problems (see also Table 1). Further, Hüffner [5] demonstrates an application of DOMINATING SET on graphs with maximum degree 4, where the size measure is the size of the dominating set.

Fedin and Kulikov [2] examine variants of SAT; however, their framework is limited in that it only proves upper bounds for a fixed algorithm instead of generating algorithms.

Skjernaa [6] also presents results on variants of SAT. His framework does not require user-provided data reduction rules, but determines reductions automatically.

Open Problems

The analysis of search tree algorithms can be much improved by describing the "size" of an instance by more than one variable, resulting in multivariate recurrences [1]. It is open to introduce this technique into an automation framework.

It has frequently been reported that better running time bounds obtained by distinguishing a large number of cases do not necessarily speed up, but in fact can slow down, a program. A careful investigation of the tradeoffs involved and a corresponding adaption of the automation frameworks is an open task.

Experimental Results

Gramm et al. [3] and Hüffner [5] report search tree sizes for several NP-complete problems. Further, Fedin and Kulikov [2] and Skjernaa [6]

Automated Search Tree Generation, Table 1 Summary of search tree sizes where automation gave improvements. "Known" is the size of the best previously published "hand-made" search tree. For the satisfiability problems, m is the number of clauses and l is the length of the formula

Problem	Trivial	Known	New
CLUSTER EDITING	3	2.27	1.92 [3]
CLUSTER DELETION	2	1.77	1.53 [3]
CLUSTER VERTEX DELETION	3	2.27	2.26 [3]
BOUNDED DEGREE DOMINATING SET	4		3.71 [5]
X3SAT, size measure m	3	1.1939	1.1586 [6]
$(n, 3)$-MAXSAT, size measure m	2	1.341	1.2366 [2]
$(n, 3)$-MAXSAT, size measure l	2	1.1058	1.0983 [2]

report on variants of satisfiability. Table 1 summarizes the results.

Cross-References

▸ Vertex Cover Search Trees

Acknowledgments Partially supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

Recommended Reading

1. Eppstein D (2004) Quasiconvex analysis of backtracking algorithms. In: Proceedings of the 15th SODA. ACM/SIAM, pp 788–797
2. Fedin SS, Kulikov AS (2006) Automated proofs of upper bounds on the running time of splitting algorithms. J Math Sci 134:2383–2391. Improved results at http://logic.pdmi.ras.ru/~kulikov/autoproofs.html
3. Gramm J, Guo J, Hüffner F, Niedermeier R (2004) Automated generation of search tree algorithms for hard graph modification problems. Algorithmica 39:321–347
4. Gramm J, Guo J, Hüffner F, Niedermeier R (2005) Graph-modeled data clustering: exact algorithms for clique generation. Theor Comput Syst 38:373–392
5. Hüffner F (2003) Graph modification problems and automated search tree generation. Diplomarbeit, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
6. Skjernaa B (2004) Exact algorithms for variants of satisfiability and colouring problems. PhD thesis, Department of Computer Science, University of Aarhus

B

Backdoors to SAT

Serge Gaspers
Optimisation Research Group, National ICT Australia (NICTA), Sydney, NSW, Australia
School of Computer Science and Engineering, University of New SouthWales (UNSW), Sydney, NSW, Australia

Keywords

Islands of tractability; Parameterized complexity; Satisfiability

Years and Authors of Summarized Original Work

2013; Gaspers, Szeider

Problem Definition

In the satisfiability problem (SAT), the input is a Boolean formula in conjunctive normal form (CNF), and the question is whether the formula is satisfiable, that is, whether there exists an assignment of truth values to the variables such that the formula evaluates to true. For example, the formula

$$(x \vee \neg y) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg z) \wedge (\neg x \vee y \vee z)$$

is satisfiable since it evaluates to true if we set x, y, and z to true.

Several classes of CNF formulas are known for which SAT can be solved in polynomial time – so-called islands of tractability. For a given island of tractability C, a C-backdoor is a set of variables of a formula such that assigning truth values to these variables gives a formula in C.

Formally, let C be a class of formulas for which the recognition problem and the satisfiability problem can be solved in polynomial time. For a subset of variables $X \subseteq \mathrm{var}(F)$ of a CNF formula F, and an assignment $\alpha : X \to \{\text{true}, \text{false}\}$ of truth values to these variables, the reduced formula $F[\alpha]$ is obtained from F by removing all the clauses containing a true literal under α and removing all false literals from the remaining clauses. The notion of backdoors was introduced by Williams et al. [15], and they come in two variants:

Definition 1 ([15]) A *weak C-backdoor* of a CNF formula F is a set of variables $X \subseteq \mathrm{var}(F)$ such that there exists an assignment α to X such that $F[\alpha] \in C$ and $F[\alpha]$ is satisfiable.

Definition 2 ([15]) A *strong C-backdoor* of a CNF formula F is a set of variables $X \subseteq \mathrm{var}(F)$ such that for each assignment α to X, we have that $F[\alpha] \in C$.

There are two main computational problems associated with backdoors. In the *detection* problem, the input is a CNF formula F and an integer k, and the question is whether F has a

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

weak/strong C-backdoor of size k. In the *evaluation* problem, the input is a CNF formula F and a weak/strong C-backdoor X, and the question is whether F is satisfiable. (In the case of weak C-backdoors, one usually requires to find a satisfying assignment since every formula that has a weak C-backdoor is satisfiable.)

The size of a smallest weak/strong C-backdoor of a CNF formula F naturally defines the distance of F to C. The size of the backdoor then becomes a very relevant parameter in studying the parameterized complexity [1] of backdoor detection and backdoor evaluation.

For a base class C where #SAT (determine the number of satisfying assignments) or Max-SAT (find an assignment that maximizes the number or weight of satisfied clauses) can be solved in polynomial time, strong C-backdoors can also be used to solve these generalizations of SAT.

Key Results

While backdoor evaluation problems are fixed-parameter tractable for SAT, the parameterized complexity of backdoor detection depends on the particular island of tractability C that is considered. If weak (resp., strong) C-backdoor detection is fixed-parameter tractable parameterized by backdoor size, SAT is fixed-parameter tractable parameterized by the size of the smallest weak (resp., strong) C-backdoor. A sample of results for the parameterized complexity of backdoor detection is presented in Table 1. The considered islands of tractability are defined in Table 2.

It can be observed that restricting the input formulas to have bounded clause length can make backdoor detection more tractable. Also, weak backdoor detection is often no more tractable than strong backdoor detection; the outlier here is FOREST-backdoor detection for general CNF formulas, where the weak version is known to be W[2]-hard but the parameterized complexity of the strong variant is still open. A CNF formula belongs to the island of tractability Forest if its incidence graph is acyclic. Here, the *incidence graph* of a CNF formula F is the bipartite graph on the variables and clauses of F where a clause is incident to the variables it contains.

Backdoors to SAT, Table 1 The parameterized complexity of finding weak and strong backdoors of CNF formulas and r-CNF formulas, where $r \geq 3$ is a fixed integer

	Weak		Strong	
Island	CNF	r-CNF	CNF	r-CNF
HORN	W[2]-h [10]	FPT [7]	FPT [10]	FPT [10]
2CNF	W[2]-h [10]	FPT [7]	FPT [10]	FPT [10]
UP	W[P]-c [14]	W[P]-c [14]	W[P]-c [14]	W[P]-c [14]
FOREST	W[2]-h [6]	FPT [6]	Open	Open
RHORN	W[2]-h [7]	W[2]-h [7]	W[2]-h [7]	Open
CLU	W[2]-h [11]	FPT [7]	W[2]-h [11]	FPT [11]

Backdoors to SAT, Table 2 Some islands of tractability

Island	Description
HORN	Horn formulas, i.e., CNF formulas where each clause contains at most one positive literal
2CNF	Krom formulas, i.e., CNF formulas where each clause contains at most two literals
UP	CNF formulas from which the empty formula or an empty clause can be derived by unit propagation (setting the literals in unit length clauses to true)
FOREST	Acyclic formulas, i.e., CNF formulas whose incidence graphs are forests
RHORN	Renamable Horn formulas, i.e., CNF formulas that can be made Horn by flipping literals
CLU	Cluster formulas, i.e., CNF formulas that are variable disjoint unions of hitting formulas. A formula is *hitting* if every two of its clauses have at least one variable occurring positively in one clause and negatively in the other

The width of graph decompositions constitutes another measure for the tractability of CNF formulas that is orthogonal to backdoors. For example, Fischer et al. [2] and Samer and Szeider [12] give linear-time algorithms solving #SAT for CNF formulas in $\mathcal{W}_{\leq t}$.

Definition 3 For every integer $t \geq 0$, $\mathcal{W}_{\leq t}$ is the class of CNF formulas whose incidence graph has treewidth at most t.

Combining backdoor and graph decomposition methods, let us now consider backdoors to $\mathcal{W}_{\leq t}$. Since FOREST $= \mathcal{W}_{\leq 1}$, weak $\mathcal{W}_{\leq t}$-backdoor detection is already W[2]-hard for $t = 1$. Fomin et al. [3] give parameterized algorithms for weak $\mathcal{W}_{\leq t}$-backdoor detection when the input formula has bounded clause length. Concerning strong $\mathcal{W}_{\leq t}$-backdoor detection for formulas with bounded clause length, Fomin et al. [3] sidestep the issue of computing a backdoor by giving a fixed-parameter algorithm, where the parameter is the size of the smallest $\mathcal{W}_{\leq t}$-backdoor, that directly solves r-SAT. The parameterized complexity of strong $\mathcal{W}_{\leq t}$-backdoor detection remains open, even for $t = 1$. However, a fixed-parameter approximation algorithm was designed by Gaspers and Szeider.

Theorem 1 ([8]) *There is a cubic-time algorithm that, given a CNF formula F and two constants $k, t \geq 0$, either finds a strong $\mathcal{W}_{\leq t}$-backdoor of size at most 2^k or concludes that F has no strong $\mathcal{W}_{\leq t}$-backdoor set of size at most k.*

Using one of the #SAT algorithms for $\mathcal{W}_{\leq t}$ [2, 12], one can use Theorem 1 to obtain a fixed-parameter algorithm for #SAT parameterized by the size of the smallest strong backdoor to $\mathcal{W}_{\leq t}$.

Corollary 1 ([8]) *There is a cubic-time algorithm that, given a CNF formula F, computes the number of satisfying assignments of F or concludes that F has no strong $\mathcal{W}_{\leq t}$-backdoor of size k for any pair of constants $k, t \geq 0$.*

In general, a fixed-parameter approximation algorithm for weak/strong C-backdoor detection is sufficient to make SAT fixed-parameter tractable parameterized by the size of a smallest weak/strong C-backdoor.

Backdoors for SAT have been considered for combinations of base classes [4, 9], and the notion of backdoors has been extended to other computational reasoning problems such as constraint satisfaction, quantified Boolean formulas, planning, abstract argumentation, and nonmonotonic reasoning; see [7]. Other variants of the notion of backdoors include deletion backdoors where variables are deleted instead of instantiated, backdoors that are sensitive to clause-learning, pseudo-backdoors that relax the requirement that the satisfiability problem for an island of tractability be solved in polynomial time, and backdoor trees; see [5].

Applications

SAT is an NP-complete problem, but modern SAT solvers perform extremely well, especially on structured and industrial instances [13].

The study of backdoors, and especially the parameterized complexity of backdoor detection problems, is one nascent approach to try and explain the empirically observed running times of SAT solvers.

Open Problems

Major open problems in the area include to determine whether

- strong FOREST-backdoor detection is fixed-parameter tractable, and whether
- strong RHORN-backdoor detection is fixed-parameter tractable for 3-CNF formulas.

Experimental Results

Experimental results evaluate running times of algorithms to find backdoors in benchmark instances, evaluate the size of backdoors of known SAT benchmark instances, compare backdoor sizes for various islands of tractability, compare backdoor sizes for various notions of backdoors, evaluate what effect preprocessing has on backdoor size, compare how backdoor sizes of random instances compare to backdoor

sizes of real-world industrial instances, and evaluate how SAT solver running times change if we force the solver to branch only on the variables of a given backdoor. The main messages are that the islands of tractability with the smallest backdoors are also those for which the backdoor detection problems are the most intractable and that existing SAT solvers can be significantly sped up on many real-world SAT instances if we feed them small backdoors. The issue is, of course, to compute these backdoors, and knowledge of the application domain, or specific SAT translations might help significantly with this task in practice. See [5] for a survey.

Cross-References

- Backtracking Based *k*-SAT Algorithms
- Exact Algorithms for General CNF SAT
- Parameterized SAT
- Treewidth of Graphs

Recommended Reading

1. Downey RG, Fellows MR (2013) Fundamentals of parameterized complexity. Springer, London
2. Fischer E, Makowsky JA, Ravve ER (2008) Counting truth assignments of formulas of bounded tree-width or clique-width. Discret Appl Math 156(4):511–529
3. Fomin FV, Lokshtanov D, Misra N, Ramanujan MS, Saurabh S (2015) Solving d-SAT via backdoors to small treewidth. In: Proceedings of the 26th annual ACM-SIAM symposium on discrete algorithms, SODA 2015, San Diego, 4–6 Jan 2015. SIAM, pp 630–641
4. Ganian R, Ramanujan M, Szeider S (2014) Discovering archipelagos of tractability: split-backdoors to constraint satisfaction. Presented at PCCR 2014 – the 2nd workshop on the parameterized complexity of computational reasoning, Vienna
5. Gario M (2011) Backdoors for SAT. Master's thesis, Dresden University of Technology. http://marco.gario.org/work/master/
6. Gaspers S, Szeider S (2012) Backdoors to acyclic SAT. In: Proceedings of the 39th international colloquium on automata, languages, and programming (ICALP 2012), Warwick. Lecture notes in computer science, vol 7391. Springer, pp 363–374
7. Gaspers S, Szeider S (2012) Backdoors to satisfaction. In: Bodlaender HL, Downey R, Fomin FV, Marx D (eds) The multivariate algorithmic revolution and beyond – essays dedicated to Michael R. Fellows on the occasion of his 60th birthday. Lecture notes in computer science, vol 7370. Springer, New York, pp 287–317
8. Gaspers S, Szeider S (2013) Strong backdoors to bounded treewidth SAT. In: 54th annual IEEE symposium on foundations of computer science, FOCS 2013, Berkeley, 26–29 Oct 2013. IEEE Computer Society, pp 489–498
9. Gaspers S, Misra N, Ordyniak S, Szeider S, Zivny S (2014) Backdoors into heterogeneous classes of SAT and CSP. In: Proceedings of the 28th AAAI conference on artificial intelligence (AAAI 2014), Québec City. AAAI Press, pp 2652–2658
10. Nishimura N, Ragde P, Szeider S (2004) Detecting backdoor sets with respect to Horn and binary clauses. In: Proceedings of the 7th international conference on theory and applications of satisfiability testing (SAT 2004), Vancouver, pp 96–103
11. Nishimura N, Ragde P, Szeider S (2007) Solving #SAT using vertex covers. Acta Informatica 44(7–8):509–523
12. Samer M, Szeider S (2010) Algorithms for propositional model counting. J Discret Algorithms 8(1):50–64
13. SAT competition (2002–) The international SAT competitions web page. http://www.satcompetition.org
14. Szeider S (2005) Backdoor sets for DLL subsolvers. J Autom Reason 35(1–3):73–88
15. Williams R, Gomes C, Selman B (2003) Backdoors to typical case complexity. In: Proceedings of the eighteenth international joint conference on artificial intelligence, IJCAI 2003. Morgan Kaufmann, San Francisco, pp 1173–1178

Backtracking Based *k*-SAT Algorithms

Ramamohan Paturi[1], Pavel Pudlák[2], Michael Saks[3], and Francis Zane[4]
[1]Department of Computer Science and Engineering, University of California at San Diego, San Diego, CA, USA
[2]Academy of Science of the Czech Republic, Mathematical Institute, Prague, Czech Republic
[3]Department of Mathematics, Rutgers, State University of New Jersey, Piscataway, NJ, USA
[4]Lucent Technologies, Bell Laboraties, Murray Hill, NJ, USA

Keywords

Boolean formulas; Conjunctive normal form satisfiability; Exponential time algorithms; Resolution

Years and Authors of Summarized Original Work

2005; Paturi, Pudlák, Saks, Zane

Problem Definition

Determination of the complexity of k-CNF satisfiability is a celebrated open problem: given a Boolean formula in conjunctive normal form with at most k literals per clause, find an assignment to the variables that satisfies each of the clauses or declare none exists. It is well known that the decision problem of k-CNF satisfiability is NP-complete for $k \geq 3$. This entry is concerned with algorithms that significantly improve the worst-case running time of the naive exhaustive search algorithm, which is $\text{poly}(n)2^n$ for a formula on n variables. Monien and Speckenmeyer [8] gave the first real improvement by giving a simple algorithm whose running time is $O(2_k^{(1-\varepsilon)n})$, with $\varepsilon_k > 0$ for all k. In a sequence of results [1, 3, 5–7, 9–12], algorithms with increasingly better running times (larger values of ε_k) have been proposed and analyzed.

These algorithms usually follow one of two lines of attack to find a satisfying solution. Backtrack search algorithms make up one class of algorithms. These algorithms were originally proposed by Davis, Logemann, and Loveland [4] and are sometimes called Davis-Putnam procedures. Such algorithms search for a satisfying assignment by assigning values to variables one by one (in some order), backtracking if a clause is made false. The other class of algorithms is based on local searches (the first guaranteed performance results were obtained by Schöning [12]). One starts with a randomly (or strategically) selected assignment and searches locally for a satisfying assignment guided by the unsatisfied clauses.

This entry presents **ResolveSat**, a randomized algorithm for k-CNF satisfiability which achieves some of the best known upper bounds. **ResolveSat** is based on an earlier algorithm of Paturi, Pudlák, and Zane [10], which is essentially a backtrack search algorithm where the variables are examined in a randomly chosen order. An analysis of the algorithm is based on the observation that as long as the formula has a satisfying assignment which is isolated from other satisfying assignments, a third of the variables are expected to occur as unit clauses as the variables are assigned in a random order. Thus, the algorithm needs to correctly guess the values of at most 2/3 of the variables. This analysis is extended to the general case by observing that either there exists an isolated satisfying assignment or there are many solutions, so the probability of guessing one correctly is sufficiently high.

ResolveSat combines these ideas with resolution to obtain significantly improved bounds [9]. In fact, **ResolveSat** obtains the best known upper bounds for k-CNF satisfiability for all $k \geq 5$. For $k = 3$ and 4, Iwama and Takami [6] obtained the best known upper bound with their randomized algorithm which combines the ideas from Schöning's local search algorithm and **ResolveSat**. Furthermore, for the promise problem of unique k-CNF satisfiability whose instances are conjectured to be among the hardest instances of k-CNF satisfiability [2], **ResolveSat** holds the best record for all $k \geq 3$. Bounds obtained by **ResolveSat** for unique k-SAT and k-SAT for $k = 3, 4, 5, 6$ are shown in Table 1. Here, these bounds are compared with those of Schöning [12], subsequently improved results based on local search [1, 5, 11], and the most recent improvements due to Iwama and Takami [6]. The upper bounds obtained by these algorithms are expressed in the form $2^{cn-o(n)}$ and the numbers in the table represent the exponent c. This comparison focuses only on the best bounds irrespective of the type of the algorithm (randomized versus deterministic).

Backtracking Based k-SAT Algorithms, Table 1 This table shows the exponent c in the bound $2^{cn-o(n)}$ for the unique k-SAT and k-SAT from the **ResolveSat** algorithm, the bounds for k-SAT from Schöning's algorithm [12], its improved versions for 3-SAT [1, 5, 11], and the hybrid version of [6]

k	unique k-SAT[9]	k-SAT[9]	k-SAT[12]	k-SAT [1,5,11]	k-SAT[6]
3	0.386 …	0.521 …	0.415 …	0.409 …	0.404 …
4	0.554 …	0.562 …	0.584 …		0.559 …
5	0.650 …		0.678 …		
6	0.711 …		0.736 …		

Notation

In this entry, a CNF Boolean formula $F(x_1, x_2, \ldots, x_n)$ is viewed as both a Boolean function and a set of clauses. A Boolean formula F is a k-CNF if all the clauses have size at most k. For a clause C, write var(C) for the set of variables appearing in C. If $v \in \text{var}(C)$, the *orientation* of v is positive if the literal v is in C and is negative if $\bar{v}$ is in C. Recall that if F is a CNF Boolean formula on variables $(x_1, x_2, \ldots, x_n)$ and a is a partial assignment of the variables, the *restriction* of F by a is defined to be the formula $F' = F\lceil_a$ on the set of variables that are not set by a, obtained by treating each clause C of F as follows: if C is set to 1 by a, then delete C and otherwise replace C by the clause C' obtained by deleting any literals of C that are set to 0 by a. Finally, a *unit clause* is a clause that contains exactly one literal.

Key Results

ResolveSat Algorithm

The **ResolveSat** algorithm is very simple. Given a k-CNF formula, it first generates clauses that can be obtained by resolution without exceeding a certain clause length. Then it takes a random order of variables and gradually assigns values to them in this order. If the currently considered variable occurs in a unit clause, it is assigned as the only value that satisfies the clause. If it occurs in contradictory unit clauses, the algorithm starts over. At each step, the algorithm also checks if the formula is satisfied. If the formula is satisfied, then the input is accepted. This subroutine is repeated until either a satisfying assignment is found or a given time limit is exceeded.

The **ResolveSat** algorithm uses the following subroutine, which takes an arbitrary assignment y, a CNF formula F, and a permutation π as input, and produces an assignment u. The assignment u is obtained by considering the variables of y in the order given by π and modifying their values in an attempt to satisfy F.

Function **Modify**(CNF formula $G(x_1, x_2, \ldots, x_n)$, permutation π of $\{1, 2, \ldots, n\}$, assignment $y) \rightarrow$ (assignment u) $G_0 = G$.**for** $i = 1$ **to** n **if** G_{i-1} contains the unit clause $x_{\pi(i)}$ **then** $u_{\pi(i)} = 1$ **else if** G_{i-1} contains the unit clause $\bar{x}_{\pi(i)}$ **then** $u_{\pi(i)} = 0$ **else** $u_{\pi(i)} = y_{\pi(i)}$ $G_i = G_{i-1}\lceil_{x\pi(i)=u\pi(i)}$ **end** /* end for loop */**return** u;

The algorithm **Search** is obtained by running **Modify** (G, π, y) on many pairs (π, y), where π is a random permutation and y is a random assignment.

Search(CNF-formula F, integer I)**repeat** I times π = uniformly random permutation of $1, \ldots, n$ y = uniformly random vector $\in \{0,1\}^n$ u = **Modify** (F, π, y); **if** u satisfies F **then** output(u); **exit**;**end**/* end repeat loop */output('Unsatisfiable');

The **ResolveSat** algorithm is obtained by combining **Search** with a preprocessing step consisting of *bounded resolution*. For the clauses C_1 and C_2, C_1 and C_2 *conflict* on variable v if one of them contains v and the other contains $\bar{v}$. C_1 and C_2 is a *resolvable pair* if they conflict on exactly one variable v. For such a pair, their *resolvent*, denoted $R(C_1, C_2)$, is the clause $C = D_1 \vee D_2$ where D_1 and D_2 are obtained by deleting v and $\bar{v}$ from C_1 and C_2. It is easy to see that any assignment satisfying C_1 and C_2 also satisfies C. Hence, if F is a satisfiable CNF formula containing the resolvable pair C_1, C_2 then the formula $F' = F \wedge R(C_1, C_2)$ has the same satisfying assignments as F. The resolvable pair C_1, C_2 is *s-bounded* if $|C_1|, |C_2| \leq s$ and $|R(C_1, C_2)| \leq s$. The following subroutine extends a formula F to a formula F_s by applying as many steps of s-bounded resolution as possible.

Resolve(CNF Formula F, integer s)$F_s = F$.**while** F_s has an s-bounded resolvable pair C_1, C_2 with $R(C_1, C_2) \notin F_s$ $F_s = F_s \wedge R(C_1, C_2)$.**return** (F_s).

The algorithm for k-SAT is the following simple combination of **Resolve** and **Search**:

ResolveSat(CNF-formula F, integer s, positive integer I)$F_s =$ **Resolve** (F, s).**Search** (F_s, I).

Analysis of ResolveSat

The running time of **ResolveSat** (F, s, I) can be bounded as follows. **Resolve** (F, s) adds at most $O(n^s)$ clauses to F by comparing pairs

of clauses, so a naive implementation runs in time $n^{3s}\text{poly}(n)$ (this time bound can be improved, but this will not affect the asymptotics of the main results). **Search** (F_s, I) runs in time $I(|F| + n^s)\text{poly}(n)$. Hence, the overall running time of **ResolveSat** (F, s, I) is crudely bounded from above by $(n^{3s} + I(|F| + n^s))\text{poly}(n)$. If $s = O(n/\log n)$, the overall running time can be bounded by $I|F|2^{O(n)}$ since $n^s = 2^{O(n)}$. It will be sufficient to choose s either to be some large constant or to be a *slowly growing* function of n. That is, $s(n)$ tends to infinity with n but is $O(\log n)$.

The algorithm **Search** (F, I) always answers "unsatisfiable" if F is unsatisfiable. Thus, the only problem is to place an upper bound on the error probability in the case that F is satisfiable. Define $\tau(F)$ to be the probability that **Modify** (F, π, y) finds some satisfying assignment. Then for a satisfiable F, the error probability of **Search** (F, I) is equal to $(1 - \tau(F))^I \le \mathrm{e}^{-I\tau(F)}$, which is at most e^{-n} provided that $I \ge n/\tau(F)$. Hence, it suffices to give good upper bounds on $\tau(F)$.

Complexity analysis of **ResolveSat** requires certain constants μ_k for $k \ge 2$:

$$\mu_k = \sum_{j=1}^{\infty} \frac{1}{j\left(j + \frac{1}{k-1}\right)}.$$

It is straightforward to show that $\mu_3 = 4 - 4\ln 2 > 1.226$ using Taylor's series expansion of $\ln 2$. Using standard facts, it is easy to show that μ_k is an increasing function of k with the limit

$$\sum_{j=1}^{\infty}(1/j^2) = (\pi^2/6) = 1.644\ldots$$

The results on the algorithm **ResolveSat** are summarized in the following three theorems.

Theorem 1

(i) *Let $k \ge 5$, and let $s(n)$ be a function going to infinity. Then for any satisfiable k-CNF formula F on n variables,*

$$\tau(F_s) \le 2^{-\left(1-\frac{\mu_k}{k-1}\right)n-o(n)}.$$

*Hence, **ResolveSat** (F, s, I) with $I = 2^{(1-\mu_k/(k-1))n+O(n)}$ has error probability $O(1)$ and running time $2^{(1-\mu_k/(k-1))n+O(n)}$ on any satisfiable k-CNF formula, provided that $s(n)$ goes to infinity sufficiently slowly.*

(ii) *For $k \ge 3$, the same bounds are obtained provided that F is uniquely satisfiable.*

Theorem 1 is proved by first considering the uniquely satisfiable case and then relating the general case to the uniquely satisfiable case. When $k \ge 5$, the analysis reveals that the asymptotics of the general case is no worse than that of the uniquely satisfiable case. When $k = 3$ or $k = 4$, it gives somewhat worse bounds for the general case than for the uniquely satisfiable case.

Theorem 2 *Let $s = s(n)$ be a slowly growing function. For any satisfiable n-variable 3-CNF formula, $\tau(F_s) \ge 2^{-0.521n}$, and so **ResolveSat** (F, s, I) with $I = n2^{0.521n}$ has error probability $O(1)$ and running time $2^{0.521n+O(n)}$.*

Theorem 3 *Let $s = s(n)$ be a slowly growing function. For any satisfiable n-variable 4-CNF formula, $\tau(F_s) \ge 2^{-0.5625n}$, and so **ResolveSat** (F, s, I) with $I = n2^{0.5625n}$ has error probability $O(1)$ and running time $2^{0.5625n+O(n)}$.*

Applications

Various heuristics have been employed to produce implementations of 3-CNF satisfiability algorithms which are considerably more efficient than exhaustive search algorithms. The **ResolveSat** algorithm and its analysis provide a rigorous explanation for this efficiency and identify the structural parameters (e.g., the width of clauses and the number of solutions), influencing the complexity.

Open Problems

The gap between the bounds for the general case and the uniquely satisfiable case when $k \in \{3, 4\}$ is due to a weakness in analysis, and it

is conjectured that the asymptotic bounds for the uniquely satisfiable case hold in general for all k. If true, the conjecture would imply that **ResolveSat** is also faster than any other known algorithm in the $k = 3$ case.

Another interesting problem is to better understand the connection between the number of satisfying assignments and the complexity of finding a satisfying assignment [2]. A strong conjecture is that satisfiability for formulas with many satisfying assignments is strictly easier than for formulas with fewer solutions.

Finally, an important open problem is to design an improved k-SAT algorithm which runs faster than the bounds presented in here for the unique k-SAT case.

Cross-References

- ▸ Exact Algorithms for k SAT Based on Local Search
- ▸ Exact Algorithms for Maximum Two-Satisfiability
- ▸ Parameterized SAT
- ▸ Thresholds of Random k-SAT

Recommended Reading

1. Baumer S, Schuler R (2003) Improving a probabilistic 3-SAT algorithm by dynamic search and independent clause pairs. In: SAT, Santa Margherita Ligure, pp 150–161
2. Calabro C, Impagliazzo R, Kabanets V, Paturi R (2003) The complexity of unique k-SAT: an isolation lemma for k-CNFs. In: Proceedings of the eighteenth IEEE conference on computational complexity, Aarhus
3. Dantsin E, Goerdt A, Hirsch EA, Kannan R, Kleinberg J, Papadimitriou C, Raghavan P, Schöning U (2002) A deterministic $(2 - \frac{2}{k+1})^n$ algorithm for k-SAT based on local search. Theor Comp Sci 289(1):69–83
4. Davis M, Logemann G, Loveland D (1962) A machine program for theorem proving. Commun ACM 5:394–397
5. Hofmeister T, Schöning U, Schuler R, Watanabe O (2002) A probabilistic 3-SAT algorithm further improved. In: STACS, Antibes Juan-les-Pins. LNCS, vol 2285, Springer, Berlin, pp 192–202
6. Iwama K, Tamaki S (2004) Improved upper bounds for 3-SAT. In: Proceedings of the fifteenth annual ACM-SIAM symposium on discrete algorithms, New Orleans, pp 328–329
7. Kullmann O (1999) New methods for 3-SAT decision and worst-case analysis. Theor Comp Sci 223(1–2):1–72
8. Monien B, Speckenmeyer E (1985) Solving satisfiability in less than 2^n steps. Discret Appl Math 10:287–295
9. Paturi R, Pudlák P, Saks M, Zane F (2005) An improved exponential-time algorithm for k-SAT. J ACM 52(3):337–364. (An earlier version presented in Proceedings of the 39th annual IEEE symposium on foundations of computer science, 1998, pp 628–637)
10. Paturi R, Pudlák P, Zane F (1999) Satisfiability Coding Lemma. In: Proceedings of the 38th annual IEEE symposium on foundations of computer science, Miami Beach, 1997, pp. 566–574. Chicago J Theor Comput Sci. http://cjtcs.cs.uchicago.edu/
11. Rolf D (2003) 3-SAT $\in$ *RTIME* (1.32971^n). In: ECCC TR03-054
12. Schöning U (2002) A probabilistic algorithm for k-SAT based on limited local search and restart. Algorithmica 32:615–623. (An earlier version appeared in 40th annual symposium on foundations of computer science (FOCS '99), pp 410–414)

Bargaining Networks

Mohammad Taghi Hajiaghayi[1] and Hamid Mahini[1]
Department of Computer Science, University of Maryland, College Park, MD, USA

Keywords

Balanced; Bargaining; Cooperative game theory; Core; Kernel; Networks; Stable

Years and Authors of Summarized Original Work

2008; Kleinberg, Tardös
2010; Bateni, Hajiaghayi, Immorlica, Mahini
2013; Farczadi, Georgiou, Köenemann

Problem Definition

A network bargaining game can be represented by a graph $G = (V, E)$ along with a set of node capacities $\{c_i | i \in V\}$ and a set of edge weights

$\{w_{ij}|(i,j) \in E\}$, where V is a set of n agents, E is the set of all possible contracts, each agent $i \in V$ has a *capacity* c_i which the maximum number of contracts in which agent i may participate, and each edge $(i,j) \in E$ has a *weight* w_{ij} which represents the surplus of a possible contract between agent i and agent j which should be divided between agents i and j upon an agreement. The main goal is to find the outcome of bargaining among agents which is a set of contracts $M \subseteq E$ and the division of surplus $\{z_{ij}\}$ for all contracts in M.

Problem 1 (Computing the Final Outcome)

INPUT: *A network bargaining game* $G = (V, E)$ *along with capacities* $\{c_i|i \in V\}$ *and weights* $\{w_{ij}|(i,j) \in E\}$.

OUTPUT: *The final outcome of bargaining among agents.*

Solution Concept

Feasible Solution

The final outcome of the bargaining process might have many properties. The main one is the feasibility. A solution $(M, \{z_{ij}\})$ is *feasible* if and only if it has the following properties:

- The degree of each node i should be at most c_i in set M.
- For each edge $(i,j) \in M$, we should have $z_{ij} + z_{ji} = w_{ij}$. This means if there is a contract between agents i and j, the surplus should be divided between these two agents.
- For each edge $(i,j) \notin M$, we should have $z_{ij} = z_{ji} = 0$.

Outside Option

Given a feasible solution $(M, \{z_{ij}\})$, the *outside* option of agent is the best deal she can make outside of set M. For each edge $(i,k) \in E - M$, agent i has an outside option by offering agent k her current worst offer. In particular, if k has less than c_k contracts in M, agent i can offer agent k exactly 0, and thus the outside option of agent i regarding agent k would be w_{ik}. On the other hand, if k has exactly c_k contracts in M, agent i may offer agent k the minimum of z_{kj} for all $(k,j) \in M$, and thus the outside option of agent i regarding agent k would be $w_{ik} - \min_{(k,j)\in M}\{z_{kj}\}$. Therefore, the outside option α_i of agent i regarding solution $(M, \{z_{ij}\})$ is defined as $\alpha_i = \max_{(i,k)\in E-M}\{w_{ik} - \eta_{ik}\}$ where

$$\eta_{ik} = \begin{cases} 0 & \text{if } k \text{ has less than } c_i \text{ contracts in } M \\ \min_{(k,j)\in M}\{z_{kj}\} & \text{if } k \text{ has exactly } c_i \text{ contracts in } M \end{cases}$$

Stable Solution

A solution $(M, \{z_{ij}\})$ is *stable* if for each contract $(i,j) \in M$, we have $z_{ij} \geq \alpha_i$ and for each agent i with less than c_i contracts in M, we have $\alpha_i = 0$. Otherwise, agent i has an incentive to deviate from M and makes a contract with agent k such that $(i,k) \notin M$.

Balanced Solution

John Nash [6] proposed a solution for the outcome of bargaining process between two agents. In his solution, known as the *Nash bargaining solution*, both agents will enjoy their outside options and then divide the surplus equally. One can leverage the intuition behind the Nash bargaining solution and defines the balanced solution in the network bargaining game. A feasible solution $(M, \{z_{ij}\})$ is *balanced* if for each contract (i,j) in M, the participants divide the net surplus equally, i.e., $z_{ik} = \alpha_i + \frac{w_{ik} - \alpha_i - \alpha_j}{2}$.

Problem 2 (Computing a Stable Solution)

INPUT: *A network bargaining game* $G = (V, E)$ *along with capacities* $\{c_i|i \in V\}$ *and weights* $\{w_{ij}|(i,j) \in E\}$.

OUTPUT: *A stable solution.*

Problem 3 (Computing a Balanced Solution)

INPUT: *A network bargaining game* $G = (V, E)$ *along with capacities* $\{c_i|i \in V\}$ *and weights* $\{w_{ij}|(i,j) \in E\}$.

OUTPUT: *A balanced solution.*

Key Results

The main goal of studying the network bargaining games is to find the right outcome of the game. Stable and balanced solutions are known to be good candidates. However, they might be too large, and moreover, some network bargaining games do not have stable and balanced solutions.

Existence of Stable and Balanced Solutions

It has been proved that a network bargaining game $G = (V, E)$ with set of weights $\{w_{ij} | (i, j) \in E\}$ has at least one stable solution if and only if the following linear program for the corresponding maximum weighted matching problem has an integral optimum solution [4,5]:

$$\begin{array}{lll} \text{maximize} & \sum_{(i,j)\in E} x_{ij} w_{ij} & \\ \text{subject to} & \sum_{(i,j)\in E} x_{ij} \le c_i, & \forall i \in V \\ & x_{ij} \le 1, & \forall (i,j) \in E \end{array} \quad \text{(LP1)}$$

Kleinberg and Tardös [5] also study network bargaining games with unit capacities, i.e., $c_i = 1$ for each agent i, and show these games have at least one balanced solution if and only if they have a stable solution. Farczadi et al. [3] generalize this result and prove the same result for network bargaining games with general capacities.

Cooperative Game Theory Perspective

One can study network bargaining games from cooperative game theory perspective. A cooperative game is defined by a set of agents V and a value function $v : 2^V \to \mathcal{R}$, where $v(S)$ represents the surplus that all agents in S alone can generate. In order to consider our bargaining game as a cooperative game, we should first define a value function for our bargaining game. The value function $v(S)$ can be defined as the size of the maximum weighted c-matching of S.

Core

An outcome $\{x_i | i \in V\}$ is in core if for each subset of agents S, we have $\sum_{i\in S} x_i \ge v(S)$ and for set V we have $\sum_{i\in V} x_i = v(V)$. This means the agents should divide the total surplus $v(V)$ such that each subset of agents earns at least as much as they alone can generate.

Prekernel

Consider an outcome $\{x_i | i \in V\}$. The *power* of agent i over agent j regarding outcome $\{x_i | i \in V\}$ is defined as $s_{ij}(x) = \max\{v(S) - \sum_{i\in S} x_i | S \subseteq V, i \in S, j \in V - S\}$. An outcome $\{x_i | i \in V\}$ is in prekernel if for every two agents i and j, we have $s_{ij}(x) = s_{ji}(x)$.

Nucleolus

Consider an outcome $\{x_i | i \in V\}$. The *excess* of set S is defined as $\epsilon(S) = v(S) - \sum_{i\in S} x_i$. Let ϵ be the vector of all possible $2^{|V|}$ excesses which are sorted in nondecreasing order. The *nucleolus* is the outcome which lexicographically maximizes vector ϵ.

There is a nice connection between stable and balanced solutions in network bargaining games and core and prekernel outcomes in cooperative games [1,2]. Bateni et al. [2] prove in a bipartite network where all nodes on one side have unit capacity, the set of stable solutions and the core coincide. Moreover, they map the set of balanced solutions to the prekernel in the same setting. Note that it is shown that this equivalence cannot be extended to a general bipartite network where nodes on both sides have general capacities [2,3].

The set of stable and balanced solutions are quite large for many instances and thus may not be used for predicting the outcome of the game. Both Azar et al. [1] and Bateni et al. [2] leverage the connection between network bargaining games and cooperative games and suggest the nucleolus as a symmetric and unique solution for the outcome of a network bargaining game [1, 2]. Bateni et al. [2] also propose a polynomial-time algorithm for finding nucleolus in bipartite networks with unit capacities on one side.

Finding Stable and Balanced Solutions

Designing a polynomial-time algorithm for finding stable and balanced solutions of a network bargaining game is a well-known problem. Kleinberg and Tardös [5] were the first who studied this problem and proposed a polynomial-time

algorithm which characterizes stable and balanced solutions when all agents have unit capacities. Their solution draws connection to the structure of matchings and the Edmonds-Gallai decomposition. Bateni et al. [2] generalize this results and design a polynomial-time algorithm for bipartite graphs where all agents on one side have general capacities and the other ones have unit capacities. They leverage the correspondence between the set of balanced solutions and the intersection of the core and prekernel and use known algorithms for finding a point in prekernel to solve the problem. Last but not least, Farczadi et al. [3] propose an algorithm for computing a balanced solution for general capacities. The main idea of their solution is to reduce an instance with general capacities to a network bargaining game with unit capacities.

Open Problems

- What is the right outcome of a network bargaining game on a general graph?
- How can we compute a proper outcome of a network bargaining game on a general graph in a polynomial time?

Cross-References

▸ Market Games and Content Distribution
▸ Matching Market Equilibrium Algorithms

Recommended Reading

1. Azar Y, Devanur NR, Jain K, Rabani Y (2010) Monotonicity in bargaining networks. In: SODA, Austin, Texas, USA, pp 817–826
2. Bateni MH, Hajiaghayi MT, Immorlica N, Mahini H (2010) The cooperative game theory foundations of network bargaining games. In: ICALP 2010, Bordeaux, France, pp 67–78
3. Farczadi L, Georgiou K, Köenemann J (2013) Network bargaining with general capacities. In: ESA, Sophia Antipolis, France, pp 433–444
4. Bayati M, Borgs C, Chayes J, Kanoria Y, Montanari A (2014) Bargaining dynamics in exchange networks. Journal of Economic Theory 156:417–454
5. Kleinberg J, Tardos E (2008) Balanced outcomes in social exchange networks. In: STOC, Victoria, BC, Canada, pp 295–304
6. Nash JF (1950) The bargaining problem. Econometrica 18:155–162

Bend Minimization for Orthogonal Drawings of Plane Graphs

Maurizio Patrignani
Engineering Department, Roma Tre University, Rome, Italy

Keywords

Bends; Grid embedding; Network flow; Orthogonal drawings; Planar embedding

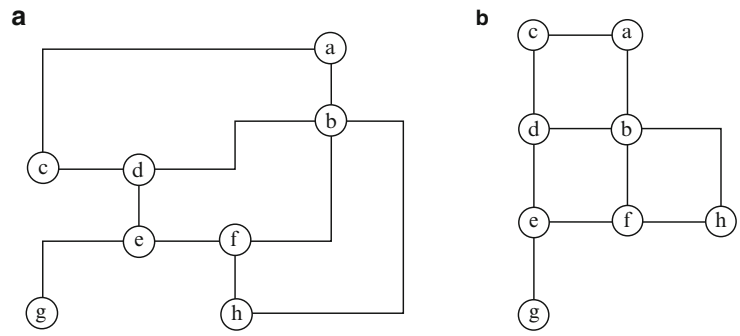
Years and Authors of Summarized Original Work

1987; Tamassia

Problem Definition

A *drawing* of a graph $G = (V, E)$ maps each vertex $v \in V$ to a distinct point of the plane and each edge $e \in E$ to a simple open Jordan curve joining its end vertices. A drawing is *planar* if the edges do not intersect. A graph is *planar* if it admits a planar drawing. A planar drawing of a graph partitions the plane into connected regions called *faces*. The unbounded face is called *external face*. Two drawings of G are *equivalent* if they induce the same circular order of the edges incident to the vertices. A *planar embedding* of G is an equivalence class of such drawings. A *plane* graph is a planar graph together with a planar embedding and the specification of the external face.

A drawing of a graph is *orthogonal* if each edge is a sequence of alternate horizontal and vertical segments. Only planar graphs of maximum degree four admit planar orthogonal drawings.



Bend Minimization for Orthogonal Drawings of Plane Graphs, Fig. 1 (**a**) An orthogonal drawing of a graph. (**b**) An orthogonal drawing of the same graph with the minimum number of bends

The points in common between two subsequent segments of the same edge are called *bends*. Figure 1 shows two orthogonal drawings of the same plane graph with seven bends and one bend, respectively.

Bend Minimization Problem

Formally, the main research problem can be defined as follows.

INPUT: *A plane graph $G = (V, E)$ of maximum degree four.*
OUTPUT: *An orthogonal drawing of G with the minimum number of bends.*

Since, given the shape of the faces, an orthogonal drawing of G with integer coordinates for vertices and bends can be computed in linear time, this problem may be alternatively viewed as that of embedding a 4-plane graph in the orthogonal grid with the minimum number of bends. Observe that if the planar embedding of the graph is not fixed, the problem of finding a minimum-bend orthogonal drawing is known to be NP-complete [9], unless the input graph has maximum degree three [5].

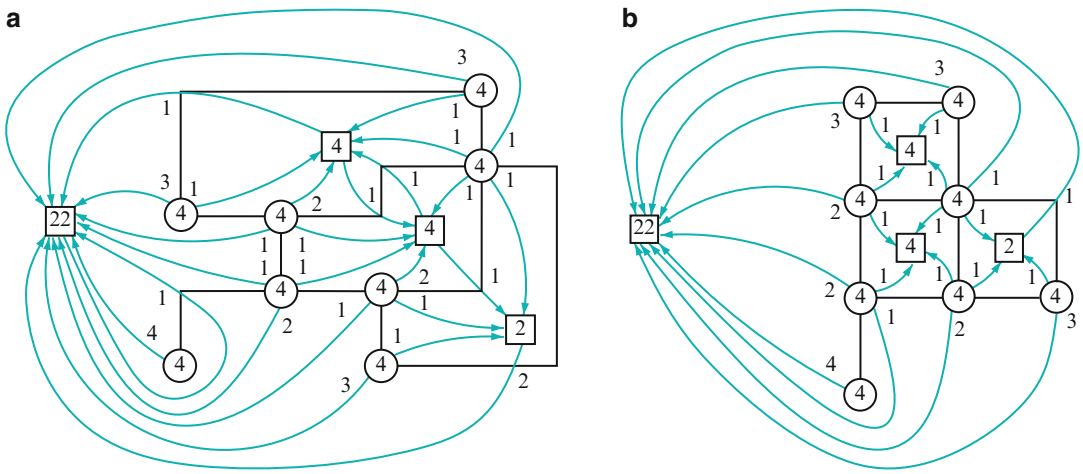
Key Results

The bend minimization problem can be solved in polynomial time by reducing it to that of finding a minimum-cost integer flow of a suitable network. Here, rather than describing the original model of [11], we describe the more intuitive model of [6].

Any orthogonal drawing of a maximum degree four plane graph $G = (V, E)$ corresponds to an integer flow in a network $\mathcal{N}(G)$ with value $4 \times n$, with $n = |V|$, where:

1. For each vertex $v \in V$, $\mathcal{N}(G)$ has a node n_v which is a source of 4 units of flow.
2. For each face f of G, $\mathcal{N}(G)$ has a node n_f which is a sink of $2\deg(f) - 4$ units if f is an internal face, or $2\deg(f) + 4$, otherwise, where $\deg(f)$ is the number of vertices encountered while traversing the boundary of face f (the same vertex may be counted multiple times).
3. For each edge $e \in E$, with adjacent faces f and g, $\mathcal{N}(G)$ has two arcs (n_f, n_g) and (n_g, n_f), both with cost 1 and lower bound 0.
4. For each vertex $v \in V$ incident to a face f of G, $\mathcal{N}(G)$ has an arc (n_v, n_f) with cost 0 and lower bound 1. Multiple incidences of the same vertex to the same face yield multiple arcs of $\mathcal{N}(G)$.

Figure 2 shows the two flows of cost 7 and 1, respectively, corresponding to the orthogonal drawings of Fig. 1. Intuitively, a flow of $\mathcal{N}(G)$ describes how 90° angles are distributed in the orthogonal drawing of G. Namely, each vertex has four 90° angles around it, hence "producing" four units of flow. The number of 90° angles needed to close a face f is given by the formula in [12], that is, $2\deg(f) - 4$ units if f is an internal face, and $2\deg(f) + 4$, otherwise. Finally, the flows traversing the edges account for their bends, where each bend allows a face



Bend Minimization for Orthogonal Drawings of Plane Graphs, Fig. 2 (**a**) The flow associated with the drawing of Fig. 1a has cost 7. (**b**) The flow associated with the drawing of Fig. 1b has cost 1

to "lose" a 90° angle and the adjacent face to "gain" it. More formally, we have the following theorem.

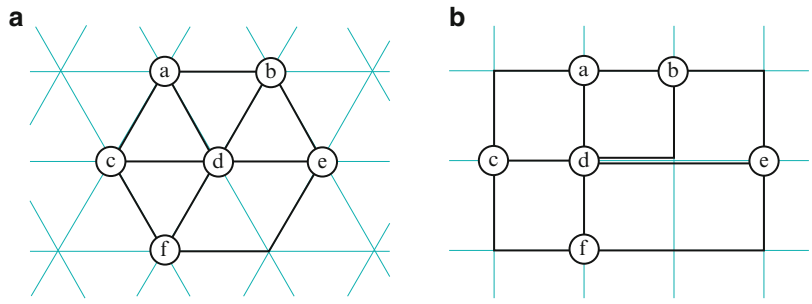
Theorem 1 *Let $G = (V, E)$ be a four-plane graph. For each orthogonal drawing of G with b bends, there exists an integer flow in $\mathcal{N}(G)$ whose value is $4 \times |V|$ and whose cost is b.*

Although several orthogonal drawings of G (e.g., with the order of the bends along edges permuted) may correspond to the same flow of $\mathcal{N}(G)$, starting from any flow, one of such drawings may be obtained in linear time. Namely, once the orthogonal shape of each face is fixed, it is possible to greedily add as many dummy edges and nodes as are needed to split the face into rectangular faces (the external face may require the addition of dummy vertices in the corners). Integer edge lengths can be consistently assigned to the sides of these rectangular faces, obtaining a grid embedding (a linear-time algorithm for doing this is described in [6]). The removal of dummy nodes and edges yields the desired orthogonal drawing. Hence, we have the following theorem.

Theorem 2 *Let $G = (V, E)$ be a four-plane graph. Given an integer flow in $\mathcal{N}(G)$ whose value is $4 \times |V|$ and whose cost is b, an orthogonal drawing of G with b bends can be found in linear time.*

Since each bend of the drawing corresponds to a unit of cost for the flow, when the total cost of the flow is minimum, any orthogonal drawing that can be obtained from it has the minimum number of bends [11].

Hence, given a plane graph $G = (V, E)$ of maximum degree four, an orthogonal drawing of G with the minimum number of bends can be computed with the same asymptotic complexity of finding a minimum-cost integer flow of $\mathcal{N}(G)$. The solution to this problem proposed in [11] is based on an iterative augmentation algorithm. Namely, starting from the initial zero flow, the final $4 \times n$ flow is computed by augmenting the flow at each of the $O(n)$ steps along a minimum-cost path. Such a path can be found with the $O(n \log n)$-implementation of Dijkstra's algorithm that exploits a priority queue. The overall $O(n^2 \log n)$ time complexity was lowered first to $O(n^{7/4}\sqrt{\log n})$ [8] and then to $O(n^{3/2})$, exploiting the planarity of the flow network [4]. However, the latter time bound is increased by an additional logarithmic factor if some edges have constraints on the number of allowed bends [4] or if the Dijkstra's algorithm for the shortest path computation is preferred with respect to the rather theoretical algorithm in [10].



Bend Minimization for Orthogonal Drawings of Plane Graphs, Fig. 3 (**a**) A drawing on the hexagonal grid. (**b**) A drawing of the same graph in the Kandinsky model

Applications

Orthogonal drawings with the minimum number of bends are of interest to VLSI circuit design, architectural floor plan layout, and aesthetic layout of diagrams used in information systems design. In particular, the orthogonal drawing standard is adopted for a wide range of diagrams, including entity-relationship diagrams, relational schemes, data-flow diagrams, flow charts, UML class diagrams, etc.

Open Problems

Several generalizations of the model have been proposed in order to deal with graphs of degree greater than four. The hexagonal grid, for example, would allow for vertices of maximum degree six (see Fig. 3a. Although the bend minimization problem is polynomial on such a grid [11], deciding edge lengths becomes NP-hard [2].

One of the most popular generalizations is the *Kandinsky* orthogonal drawing standard [7] where vertices of arbitrary degree are represented as small squares or circles of the same dimensions, while the first segments of the edges that leave a vertex in the same direction run very close together (see, e.g., Fig. 3b). Although the bend minimization problem in the Kandinsky orthogonal drawing standard has been shown to be NP-hard [3], this model is of great interest for applications. An extension of the flow model that makes it possible to solve this problem in polynomial time for a meaningful subfamily of Kandinsky orthogonal drawings has been proposed in [1]. Namely, in addition to the drawing conventions of the Kandinsky model, each vertex with degree greater than four has at least one incident edge on each side, and each edge leaving a vertex either has no bend or has its first bend on the right.

Cross-References

- Planarisation and Crossing Minimisation
- Single-Source Shortest Paths

Recommended Reading

1. Bertolazzi P, Di Battista G, Didimo W (2000) Computing orthogonal drawings with the minimum number of bends. IEEE Trans Comput 49(8):826–840
2. Biedl TC (2007) Realizations of hexagonal graph representations. In: Bose P (ed) CCCG. Carleton University, Ottawa, pp 89–92
3. Bläsius T, Brückner G, Rutter I (2014) Complexity of higher-degree orthogonal graph embedding in the Kandinsky model. In: Schulz AS, Wagner D (eds) Algorithms – ESA 2014, Wroclaw. Lecture notes in computer science, vol 8737. Springer, Berlin/Heidelberg, pp 161–172
4. Cornelsen S, Karrenbauer A (2012) Accelerated bend minimization. J Graph Algorithms Appl 16(3):635–650
5. Di Battista G, Liotta G, Vargiu F (1998) Spirality and optimal orthogonal drawings. SIAM J Comput 27(6):1764–1811
6. Di Battista G, Eades P, Tamassia R, Tollis IG (1999) Graph drawing: algorithms for the visualization of graphs. Prentice-Hall, Upper Saddle River
7. Fößmeier U, Kaufmann M (1996) Drawing high degree graphs with low bend numbers. In: Brandenburg FJ (ed) Graph drawing. Lecture notes in computer science, vol 1027. Springer, Berlin/Heidelberg, pp 254–266

8. Garg A, Tamassia R (1996) A new minimum cost flow algorithm with applications to graph drawing. In: North SC (ed). Graph Drawing, Lecture notes in computer science, vol 1190. Springer, Berlin/Heidelberg, pp 201–216
9. Garg A, Tamassia R (2001) On the computational complexity of upward and rectilinear planarity testing. SIAM J Comput 31(2):601–625
10. Henzinger MR, Klein PN, Rao S, Subramanian S (1997) Faster shortest-path algorithms for planar graphs. J Comput Syst Sci 55(1):3–23
11. Tamassia R (1987) On embedding a graph in the grid with the minimum number of bends. SIAM J Comput 16(3):421–444
12. Vijayan G, Wigderson A (1985) Rectilinear graphs and their embeddings. SIAM J Comput 14(2):355–372

Best Response Algorithms for Selfish Routing

Paul (Pavlos) Spirakis
Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece
Computer Science, University of Liverpool, Liverpool, UK
Computer Technology Institute (CTI), Patras, Greece

Keywords

Atomic selfish flows

Years and Authors of Summarized Original Work

2005; Fotakis, Kontogiannis, Spirakis

Problem Definition

A setting is assumed in which n selfish users compete for routing their loads in a network. The network is an $s-t$ directed graph with a single source vertex s and a single destination vertex t. The users are ordered sequentially. It is assumed that each user plays after the user before her in the ordering, and the desired end result is a Pure Nash Equilibrium (PNE for short). It is assumed that, when a user plays (i.e., when she selects an $s-t$ path to route her load), the play is a best response (i.e., minimum delay), given the paths and loads of users currently in the net. The problem then is to find the class of directed graphs for which such an ordering exists so that the implied sequence of best responses leads indeed to a Pure Nash Equilibrium.

The Model

A *network congestion game* is a tuple $((w_i)_{i\in N}, G, (d_e)_{e\in E})$ where $N = \{1,\ldots,n\}$ is the set of users where user i controls w_i units of traffic demand. In *unweighted* congestion games $w_i = 1$ for $i = 1,\ldots,n$. $G(V,E)$ is a directed graph representing the communications network and d_e is the latency function associated with edge $e \in E$. It is assumed that the d_e's are non-negative and non-decreasing functions of the edge loads. The edges are called *identical* if $d_e(x) = x$, $\forall e \in E$. The model is further restricted to single-commodity network congestion games, where G has a single source s and destination t and the set of users' strategies is the set of $s-t$ paths, denoted P. Without loss of generality it is assumed that G is connected and that every vertex of G lies on a directed $s-t$ path.

A vector $P = (p_1,\ldots,p_n)$ consisting of an $s-t$ path p_i for each user i is a *pure strategies profile*. Let $l_e(P) = \sum_{i:e\in p_i} w_i$ be the load of edge e in P. The authors define *the cost* $\lambda^i_p(P)$ for user i routing her demand on path p in the profile P to be

$$\lambda^i_p(P) = \sum_{e\in p\cap p_i} d_e\left(l_e(P)\right) + \sum_{e\in p\smallsetminus p_i} d_e\left(l_e(P)+w_i\right).$$

The cost $\lambda^i(P)$ of user i in P is just $\lambda^i_{p_i}(P)$, i.e., the total delay along her path.

A pure strategies profile P is a Pure Nash Equilibrium (PNE) iff no user can reduce her total

delay by *unilaterally deviating* i.e., by selecting another $s-t$ path for her load, while all other users keep their paths.

Best Response

Let p_i be the path of user i and $P^i = (p_1, \ldots, p_i)$ be the pure strategies profile for users $1, \ldots, i$. Then the *best response* of user $i+1$ is a path p_{i+1} so that

$$p_{i+1} = avg \min_{p \in P^i} \left\{ \sum_{e \in p} \left(d_e \left(l_e \left(P^i \right) + w_{i+1} \right) \right) \right\}.$$

Flows and Common Best Response

A (feasible) flow on the set P of $s-t$ paths of G is a function $f : P \to \Re_{\geq 0}$ so that

$$\sum_{p \in P} f_p = \sum_{i=1}^{n} w_i .$$

The single-commodity network congestion game $((w_i)_{i \in N}, G, (d_e)_{e \in E})$ has the Common Best Response property if for every initial flow f (not necessarily feasible), all users have the same set of best responses with respect to f. That is, if a path p is a best response with respect to f for some user, then for all users j and all paths p'

$$\sum_{e \in p'} d_e \left(f_e + w_j \right) \geq \sum_{e \in p} d_e \left(f_e + w_j \right).$$

Furthermore, every segment π of a best response path p is a best response for routing the demand of any user between π's endpoints. It is allowed here that some users may already have contributed to the initial flow f.

Layered and Series-Parallel Graphs

A directed (multi)graph $G(V, E)$ with a distinguished source s and destination t is *layered* iff all directed $s-t$ paths have exactly the same length and each vertex lies on some directed $s-t$ path.

A multigraph is *series-parallel* with *terminals* (s, t) if

1. it is a single edge (s, t) or
2. it is obtained from two series-parallel graphs G_1, G_2 with terminals (s_1, t_1) and (s_2, t_2) by connecting them either in *series* or in *parallel*. In a series connection, t_1 is identified with s_2 and s_1 becomes s and t_2 becomes t. In a parallel connection, $s_1 = s_2 = s$ and $t_1 = t_2 = t$.

Key Results

The Greedy Best Response Algorithm (GBR)

GBR considers the users one-by-one in *non-increasing* order of weight (i.e., $w_1 \geq w_2 \geq \cdots \geq w_n$). Each user adopts her best response strategy on the set of (already adopted in the net) best responses of previous users. No user can change her strategy in the future. Formally, GBR *succeeds* if the eventual profile P is a Pure Nash Equilibrium (PNE).

The Characterization

In [3] it is shown:

Theorem 1 *If G is an $(s-t)$ series-parallel graph and the game $((w_i)_{i \in N}, G, (d_e)_{e \in E})$ has the common best response property, then GBR succeeds.*

Theorem 2 *A weighted single-commodity network congestion game in a layered network with identical edges has the common best response property for any set of user weights.*

Theorem 3 *For any single-commodity network congestion game in series-parallel networks, GBR succeeds if*

1. *The users are identical (if $w_i = 1$ for all i) and the edge-delays are arbitrary but non-decreasing or*
2. *The graph is layered and the edges are identical (for arbitrary user weights)*

Theorem 4 *If the network consists of bunches of parallel-links connected in series, then a PNE is obtained by applying GBR to each bunch.*

Theorem 5

1. *If the network is not series-parallel then there exist games where GBR fails, even for 2 identical users and identical edges.*
2. *If the network does not have the common best response property (and is not a sequence of parallel links graphs connected in series) then there exist games where GBR fails, even for 2-layered series-parallel graphs.*

Examples of such games are provided in [3].

Applications

GBR has a natural distributed implementation based on a leader election algorithm. Each player is now represented by a process. It is assumed that processes know the network and the edge latency functions. The existence of a message passing subsystem and an underlying synchronization mechanism (e.g., logical timestamps) is assumed, that allows a distributed protocol to proceed in logical rounds.

Initially all processes are active. In each round they run a leader election algorithm and determine the process of largest weight (among the active ones). This process routes its demand on its best response path, announces its strategy to all active processes, and becomes passive. Notice that each process can compute its best response locally.

Open Problems

What is the class of networks where (identical) users can achieve a PNE by a k-round repetition of a best responses sequence? What happens to weighted users? In general, how the network topology affects best response sequences? Such open problems are a subject of current research.

Cross-References

▸ General Equilibrium

Recommended Reading

1. Awerbuch B, Azar Y, Epstein A (2005) The price of routing unsplittable flows. In: Proceedings of the ACM symposium on theory of computing (STOC) 2005. ACM, New York, pp 57–66
2. Duffin RJ (1965) Topology of series-parallel networks. J Math Anal Appl 10:303–318
3. Fotakis D, Kontogiannis S, Spirakis P (2006) Symmetry in network congestion games: pure equilibria and anarchy cost. In: Proceedings of the 3rd workshop of approximate and on-line algorithms (WAOA 2005). Lecture notes in computer science (LNCS), vol 3879. Springer, Berlin/Heidelberg, pp 161–175
4. Fotakis D, Kontogiannis S, Spirakis P (2005) Selfish unsplittable flows. J Theor Comput Sci 348:226–239
5. Libman L, Orda A (2001) Atomic resource sharing in noncooperative networks. Telecommun Syst 17(4):385–409

Beyond Evolutionary Trees

Riccardo Dondi[1] and Yuri Pirola[2]
[1]Università degli Studi di Bergamo, Bergamo, Italy
[2]Università degli Studi di Milano-Bicocca, Milan, Italy

Keywords

Consensus networks; Lateral gene transfer; Phylogenetic networks

Years and Authors of Summarized Original Work

2005; Choy, Jansson, Sadakane, Sung
2010; Della Vedova, Dondi, Jiang, Pavesi, Pirola, Wang
2011; Tofigh, Hallett, Lagergren
2011; van Iersel, Kelk
2014; Kelk, Scornavacca

Problem Definition

Recent developments in phylogenetics have provided evidences that evolutionary histories

cannot always be represented as a single tree; thus, more sophisticated representations are needed. *Phylogenetic networks* are natural extensions of phylogenetic trees that recently gathered general consensus in literature. Let Λ be a finite set of labels, representing a set of extant species (taxa). A *rooted phylogenetic* N *over* Λ (or, simply, *phylogenetic network* or *network*) is a directed acyclic connected graph $N = \langle V(N), A(N) \rangle$ containing a unique vertex with no incoming arcs, called *root* of N, and a labeling function from the set $L(N)$ of leaves of N to the set of labels Λ. The set of labels associated with the leaves of N is denoted by $\Lambda(N)$, and phylogenetic networks whose leaves are in bijection with the set of labels are called *uniquely labeled*. The undirected edges underlying the set $A(N)$ are denoted with $E(N)$.

We will discuss two important families of problems where phylogenetic networks have been introduced: consensus network computation and tree reconciliation. Other models (and the related problems) for representing and reconstructing non-treelike evolutionary scenarios are presented in [7]. The family of the consensus network computation problems asks for a single phylogenetic network (called consensus network) that best summarizes all the information provided by a collection of "structures" representing the evolutionary relationships among the set of taxa Λ. The family of tree reconciliation problems, instead, analyzes the evolution of a gene in order to either reconstruct the evolution of a set of species or infer the evolutionary scenario of the considered gene.

Observe that it is possible to topologically sort the vertices of a phylogenetic network, so that each vertex always appears after all its predecessors; hence, it is possible to define the *children*, *parents*, *ancestors*, and *descendants* of a given vertex, as usual for trees. Furthermore, as for trees, we can define the *least common ancestor* of a set of nodes. Given a subset A of nodes of a phylogenetic network N, then the least common ancestor (or, shortly, lca) of A in N is a node x of N that is an ancestor of each node in A and that is the furthest such node from the root.

Consensus Network Reconstruction

The aim of consensus network reconstruction problems is computing a unique phylogenetic network (called *consensus network*) that best summarizes all the information provided by a collection of "structures" representing the evolutionary relationships among the set of taxa Λ. Different specific computational problems have been defined in the literature depending (i) on the kind of input structures considered and (ii) on the definition of the optimality criterion used to choose the best consensus network. Simple formulations (such as *maximum agreement subtree* (MAST), *maximum compatible tree* (MCT), *maximum agreement supertree* (MASP)) compute a consensus network which is actually a phylogenetic tree. However, trees are not always sufficient for describing conflicting information and real evolutionary scenarios; hence, formulations which attempt to reconstruct phylogenetic networks have been proposed.

In terms of optimality criterion, two aims can be pursued: either finding the largest set of taxa that "share" (as defined below) a common substructure or finding the "simplest" network that represents all taxa. For measuring the complexity of a network, two natural parameters are considered: the *reticulation number* and the *level* of the network. *Reticulation* (or *hybrid*) nodes are nodes of the network with more than one parent. The *reticulation number* of a network N is defined as $|E(N)|-|V(N)|+1$ and represents how "far" the network is from a phylogenetic tree (which has reticulation number 0). If all reticulation nodes have indegree 2 (which is often assumed in the literature), then the reticulation number is equal to the number of reticulation nodes. The *level* of a network N is defined as the maximum number of reticulation nodes in a biconnected component of $\langle V(N), E(N) \rangle$ (i.e., the undirected graph obtained from the network) [6]. Phylogenetic trees are level-0 networks, while level-1 networks are often called *galled trees* [13].

In terms of kinds of input structures, several options have been studied, and, among them, the most important ones are phylogenetic networks, triplets/quartets, and clusters. When input structures are phylogenetic networks, the usual aim is to reconstruct their *maximum agreement*

subnetwork (MASN) [6], that is, a level-k phylogenetic network A (for some fixed k) uniquely labeled with a set $\Lambda' \subseteq \Lambda$ of maximum cardinality such that A is a subgraph of the topological restriction of each input network w.r.t. Λ'. The topological restriction of a uniquely labeled network N to a subset $\Lambda' \subseteq \Lambda$ of labels is defined as the network obtained by first deleting all nodes which are not on any directed path from the root to one of the leaves labeled in Λ' along with their incident edges, and then, for every node with outdegree 1 and indegree less than 2, contracting its outgoing edges. (Notice that the MAST problem is a special case of the MASN problem when $k = 0$.)

Triplets are rooted binary phylogenetic trees on exactly three species/leaves. They are generally represented as $xy|z$, indicating that the parent of x and y is a child of the parent of z. The consensus network reconstruction problem from triplets is the problem of finding, if possible, a phylogenetic network N *consistent* with each triplet given as input (or with the maximum number of them). A phylogenetic network N is said to be consistent with a triplet $xy|z$ if N contains two distinct vertices u, v and the four pairwise internally vertex-disjoint paths $u \to x$, $u \to y$, $v \to u$, $v \to z$. The resulting phylogenetic network is required to either have minimum level or have fixed level (possibly with minimum reticulation number) [18,26,27]. The related problem on *quartets* (i.e., unrooted phylogenetic trees on four species) has been also proposed [8].

Clusters are (strict) subsets of Λ. The consensus network reconstruction problem from clusters is the problems of finding a phylogenetic network N that *represents* all clusters given as input. There exist two main definitions for "represents," commonly referred to as the "hardwired" and the "softwired" definitions. A network N represents a cluster $\Lambda' \subset \Lambda$ in the *hardwired* sense if there exists an arc $(u, v) \in A(N)$ such that Λ' is exactly the set of labels associated with the leaves of a subnetwork rooted in v [16]. Instead, a network N represents a cluster $\Lambda' \subset \Lambda$ in the *softwired* sense if there exists an arc $(u, v) \in A(N)$ such that Λ' is the set of labels associated with the leaves of a subtree rooted in v obtained by removing, for each reticulation node, all edges but one directed to that node [15]. In both cases, the computational problems focus on reconstructing networks with minimum level [29].

Reconciliation of Gene Trees and Species Trees

The evolution of a family of homologous genes in a given set of species is usually represented as a phylogenetic tree, called a *gene tree*, where each label can be associated with more than one leaf, while the evolution of the considered set of species is called a *species tree*, which is a uniquely labeled tree. Due to different evolutionary events that affect gene evolution (*duplications*, *losses*, *lateral gene transfer*), the evolution represented by a gene tree and a species tree (or by two different gene trees) can be different.

Two fundamental combinatorial problems have been studied in this field: the *reconstruction* of the species tree associated with the homologous genes considered [5, 12, 21] and the *reconciliation* of a gene tree with a given species tree [3, 4, 9, 21, 23], whose goal is the inference of the evolutionary events that occurred in the genes evolution. Here, we consider only the latter problem; hence, we assume that a gene tree (or a set of gene trees) and a (correct) species tree are given.

Given a set S of taxa, a species tree T_S and a gene tree T_G are two rooted binary trees, leaf-labeled by S, with $\Lambda(T_G) \subseteq \Lambda(T_S)$. Two nodes of a tree T are comparable when one is an ancestor of the other. T_G and T_S are compared introducing a mapping $\lambda : V(T_G) \to V(T_S)$, which usually corresponds to the *least common ancestor mapping*. Three biological events are considered for gene families' evolution: *duplications*, *losses*, and *lateral gene transfers*.

A *duplication* is a copy of a given gene, after which the two copies evolve independently. A duplication occurs in an internal node g of T_G if and only if $\lambda(f(g)) = \lambda(g)$, for a child $f(g)$ of g. A *loss* of a gene in some species consists in a copy of a gene disappearing during the evolution of a given gene family. The losses can be computed from the mapping λ between T_G and T_S [21].

When duplications and losses are the only evolutionary events considered, a gene tree and a species tree are compared with a *reconciled tree* $R(T_G, T_S)$ [2,5,9–11,21,23]. The reconciled tree is a binary tree that represents an embedding of a gene tree inside a species tree, and it allows to identify when duplications and losses occur. However, when considering also *lateral gene transfers* (also called horizontal gene transfers), the scenario changes, and the evolutionary history of a gene family must be represented by a phylogenetic network. A lateral gene transfer occurs when some genetic material is transferred from a taxon to another taxon which is not a descendant of the first taxon.

In order to represent a reconciliation of a gene tree T_G and a species tree T_S considering as evolutionary events, duplications, losses, and lateral gene transfers, the definition of *duplication-transfer-loss scenario* (DTL-scenario) has been introduced [25]. Notice that other models of reconciliation have been proposed, notably [10].

Definition 1 A DTL-scenario is a tuple $\mathcal{S} = (T_S, T_G, \sigma, \lambda^*, \Sigma, \Delta, \Gamma, \Theta)$ where T_S is a species tree; T_G is gene tree; σ maps each leaf of T_G with the corresponding leaf of T_S; λ^* maps each node of T_G in a node of T_S (λ^* can be considered as a generalization of the least common ancestor mapping λ); $\{\Sigma, \Delta, \Gamma\}$ is a tripartition of the internal nodes of T_G in speciation nodes, duplication nodes, transfer nodes, respectively, while Θ is a subset of the edges of T_G; and the following properties hold:

1. For each leaf of T_G, $\lambda^*(u) = \sigma(u)$.
2. Consider a node x with children x_l and x_r, then $\lambda^*(x)$ is not a proper descendant of one of $\lambda^*(x_r)$, $\lambda^*(x_l)$, and one of $\lambda^*(x_r)$, $\lambda^*(x_l)$ is a descendant of $\lambda^*(x)$.
3. Given an edge (x, y) of T_G, then $(x, y) \in \Theta$ if and only if x is not comparable with y.
4. Given nodes x of T_G with children x_r and x_l, then:
   (a) $x \in \Gamma$ if and only if $(x, x_l) \in \Theta$ or $(x, x_r) \in \Theta$.
   (b) $x \in \Sigma$ only if $\lambda^*(x)$ is the least common ancestor of $\lambda^*(x_l)$ and $\lambda^*(x_r)$, and $\lambda^*(x_l)$ and $\lambda^*(x_r)$ are not comparable.
   (c) $x \in \Delta$ only if $\lambda^*(x)$ is an ancestor of the least common ancestor of $\lambda^*(x_l)$ and $\lambda^*(x_r)$, and $\lambda^*(x_l)$ and $\lambda^*(x_r)$ are comparable.
5. Consider two edges $(x, x') \in \Theta$ and $(y, y') \in \Theta$, with x' an ancestor of y', then $\lambda^*(x')$ is an ancestor of $\lambda^*(y)$.

The number of losses is directly inferred from a given scenario $\mathcal{S}$ [1,25].

Now, we briefly discuss the biological motivations for the conditions introduced. Condition 1 guarantees the correspondence of each leaf T_G with the corresponding species (leaf) of T_S. Condition 2 guarantees that the order on the nodes of T_G is preserved by the mapping λ^*. Condition 3 defines the edges associated with a lateral gene transfer. Condition 4 establishes that the nodes of T_G can be either associated with a lateral gene transfer (condition 4.a), with a speciation (condition 4.b, then each node x and its two children must be mapped in different nodes of T_S), or with a duplication (condition 4.c, then each node x and at least one of its two children must be mapped in the same node of T_S). The last condition (condition 5) is introduced to ensure that different lateral gene transfers are biologically meaningful, that is, those events relate coexisting species, and that if (x, x') is lateral gene transfer, then there is no lateral gene transfer (y, y'), where y is a proper ancestor of x and y' is a proper descendant of x'.

Key Results

Consensus Networks

The maximum agreement subnetwork (MASN) problem is NP-hard even if the input is composed of a binary tree and an unbounded-level network [17]. If the input is composed of two level-1 networks, the problem can be solved in time $O(n^2)$, and it is fixed-parameter tractable if the input is composed of two level-k networks (where k is the parameter) [6].

Given a set of triplets, constructing a level-k phylogenetic network consistent with all of them is NP-hard for all $k \geq 1$, while it is NP-hard

for all $k \geq 0$ if we want to construct a level-k network consistent with the maximum number of them [28]. If the input set $\mathcal{T}$ of triplets is *dense* (i.e., it contains a triplet for each cardinality three subset of taxa), then a level-k network consistent with all triplets can be found (if any) in time $O(\mathcal{T})$ for $k = 1$ [18], in time $O(\mathcal{T}^3)$ for $k = 2$ [27], and in polynomial time for any fixed k [24]. Recently, these results have been extended in order to minimize the reticulation number of the computed network [14, 26].

The reconstruction of a consensus network from a set of clusters in the hardwired sense has been tackled in [16], where an algorithm for reconstructing a phylogenetic network that represents a set $\mathcal{C}$ of clusters (and only $\mathcal{C}$) with $O(|\mathcal{C}|)$ nodes and $O(|\mathcal{C}|^2)$ edges is presented. An algorithm for reconstructing a level-1 network from clusters in the softwired sense has been shown in [15], which has been later extended in order to compute in polynomial time a level-k network or a network with reticulation number k for every fixed k [19]. An efficient algorithm for computing a level-k network (with $k \in \{1, 2\}$) representing a set of cluster in the softwired sense which also attempts to minimize the reticulation number has also been presented [29].

Reconciliation

The main combinatorial problem related to reconciliation is, given a species tree and a gene tree, the computation of a DTL-scenario of minimum cost, for some function that assigns positive cost to duplications, losses, and lateral gene transfers. The problem is known to be NP-hard [22, 25]. However, two tractable variants of the problem have been considered.

A first variant, called *cyclic DTL-scenario*, does not consider condition 5 of Definition 1. Computing a cyclic DTL-scenario of minimum cost is polynomial time solvable. First, an algorithm for cost function that assigns positive cost only to duplications and lateral gene transfers was presented [25]. Later a linear time algorithm for a general cost function (hence losses are assigned a positive cost) was given [1].

A second variant considered is the *dated version*, where nodes of the species are associated with labels that represent the divergence time and a lateral gene transfer is possible only between coexisting species. In this case, computing an acyclic DTL-scenario of minimum cost is polynomial time solvable [20].

Open Problems

The fixed parameter tractability of computing a minimum reticulation-number phylogenetic network that combines a set of nonbinary trees has not yet be assessed, and only a few attempts focus on approximate solutions. Answers to both questions could be of interest.

An interesting open problem related to the computation of a DTL-scenario of minimum cost is the investigation of the parameterized complexity for acyclic DTL-scenarios, when parameterized by the cost of the solution or by the number of lateral gene transfers. Another interesting direction for this problem is to investigate its approximation complexity.

Cross-References

- Algorithms for Combining Rooted Triplets into a Galled Phylogenetic Network
- Maximum Agreement Subtree (of 2 Binary Trees)
- Maximum Agreement Subtree (of 3 or More Trees)
- Maximum Agreement Supertree
- Maximum Compatible Tree

Recommended Reading

1. Bansal MS, Alm EJ, Kellis M (2012) Efficient algorithms for the reconciliation problem with gene duplication, horizontal transfer and loss. Bioinformatics 28(12):283–291. doi:10.1093/bioinformatics/bts225
2. Bonizzoni P, Della Vedova G, Dondi R (2005) Reconciling a gene tree to a species tree under the duplication cost model. Theor Comput Sci 347(1–2):36–53. doi:10.1016/j.tcs.2005.05.016

3. Burleigh JG, Bansal MS, Wehe A, Eulenstein O (2008) Locating multiple gene duplications through reconciled trees. In: Proceedings of the 12th annual international conference on research in computational molecular biology, RECOMB 2008, Singapore. LNCS, vol 4955. Springer, pp 273–284. doi:10.1007/978-3-540-78839-3_24
4. Chang WC, Eulenstein O (2006) Reconciling gene trees with apparent polytomies. In: Proceedings of the 12th annual international conference on computing and combinatorics, COCOON 2006, Taipei. LNCS, vol 4112. Springer, pp 235–244. doi:10.1007/11809678_26
5. Chauve C, El-Mabrouk N (2009) New perspectives on gene family evolution: losses in reconciliation and a link with supertrees. In: Proceedings of the 13th annual international conference on research in computational molecular biology, RECOMB 2009, Tucson. LNCS, vol 5541. Springer, pp 46–58. doi:10.1007/978-3-642-02008-7_4
6. Choy C, Jansson J, Sadakane K, Sung WK (2005) Computing the maximum agreement of phylogenetic networks. Theor Comput Sci 335(1):93–107. doi:10.1016/j.tcs.2004.12.012
7. Della Vedova G, Dondi R, Jiang T, Pavesi G, Pirola Y, Wang L (2010) Beyond evolutionary trees. Nat Comput 9(2):421–435. doi:10.1007/s11047-009-9156-6
8. Gambette P, Berry V, Paul C (2012) Quartets and unrooted phylogenetic networks. J Bioinform Comput Biol 10(4). doi:10.1142/S0219720012500047
9. Goodman M, Czelusniak J, Moore GW, Romero-Herrera AE, Matsuda G (1979) Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. Syst Zool 28(2):132–163. doi:10.1093/sysbio/28.2.132
10. Gòrecki P (2004) Reconciliation problems for duplication, loss and horizontal gene transfer. In: Proceedings of the 8th annual international conference on computational molecular biology, RECOMB 2004, San Diego. ACM, pp 316–325. doi:10.1145/974614.974656
11. Gòrecki P, Tiuryn J (2006) DLS-trees: a model of evolutionary scenarios. Theor Comput Sci 359(1–3):378–399. doi:10.1016/j.tcs.2006.05.019
12. Guigò R, Muchnik I, Smith T (1996) Reconstruction of ancient molecular phylogeny. Mol Phylogenet Evol 6(2):189–213. doi:10.1006/mpev.1996.0071
13. Gusfield D, Eddhu S, Langley CH (2004) Optimal, efficient reconstruction of phylogenetic networks with constrained recombination. J Bioinform Comput Biol 2(1):173–214. doi:10.1142/S0219720004000521
14. Habib M, To TH (2012) Constructing a minimum phylogenetic network from a dense triplet set. J Bioinform Comput Biol 10(5). doi:10.1142/S0219720012500138
15. Huson DH, Klöpper TH (2007) Beyond galled trees – decomposition and computation of galled networks. In: Proceedings of the 11th annual international conference on research in computational molecular biology, RECOMB 2007, Oakland. LNCS, vol 4453. Springer, pp 211–225. doi:10.1007/978-3-540-71681-5_15
16. Huson DH, Rupp R (2008) Summarizing multiple gene trees using cluster networks. In: Proceedings of the 8th international workshop on algorithms in bioinformatics, WABI 2008, Karlsruhe. LNCS, vol 5251. Springer, pp 296–305. doi:10.1007/978-3-540-87361-7_25
17. Jansson J, Sung WK (2004) The maximum agreement of two nested phylogenetic networks. In: Proceedings of the 15th international symposium on algorithms and computation, ISAAC 2004, Hong Kong. LNCS, vol 3341. Springer, pp 581–593. doi:10.1007/978-3-540-30551-4_51
18. Jansson J, Nguyen NB, Sung WK (2006) Algorithms for combining rooted triplets into a galled phylogenetic network. SIAM J Comput 35(5):1098–1121. doi:10.1137/S0097539704446529
19. Kelk S, Scornavacca C (2014) Constructing minimal phylogenetic networks from softwired clusters is fixed parameter tractable. Algorithmica 68(4):886–915. doi:10.1007/s00453-012-9708-5
20. Libeskind-Hadas R, Charleston MA (2009) On the computational complexity of the reticulate cophylogeny reconstruction problem. J Comput Biol 16(1):105–117. doi:10.1089/ cmb.2008.0084
21. Ma B, Li M, Zhang L (2000) From gene trees to species trees. SIAM J Comput 30(3):729–752. doi:10.1137/S0097539798343362
22. Ovadia Y, Fielder D, Conow C, Libeskind-Hadas R (2011) The cophylogeny reconstruction problem is NP-complete. J Comput Biol 18(1):59–65. doi:10.1089/cmb.2009.0240
23. Page R (1994) Maps between trees and cladistic analysis of historical associations among genes. Syst Biol 43(1):58–77. doi:10.1093/sysbio/43.1.58
24. To TH, Habib M (2009) Level-k phylogenetic networks are constructable from a dense triplet set in polynomial time. In: Proceedings of the 20th annual symposium on combinatorial pattern matching, CPM 2009, Lille. LNCS, vol 5577. Springer, pp 275–288. doi:10.1007/978-3-642-02441-2_25
25. Tofigh A, Hallett MT, Lagergren J (2011) Simultaneous identification of duplications and lateral gene transfers. IEEE/ACM Trans Comput Biol Bioinform 8(2):517–535. doi:10.1109/TCBB.2010.14
26. van Iersel L, Kelk S (2011) Constructing the simplest possible phylogenetic network from triplets. Algorithmica 60(2):207–235. doi:10.1007/s00453-009-9333-0
27. van Iersel L, Keijsper J, Kelk S, Stougie L, Hagen F, Boekhout T (2009) Constructing level-2 phylogenetic networks from triplets. IEEE/ACM Trans Comput Biol Bioinform 6(4):667–681. doi:10.1145/1671403.1671415
28. van Iersel L, Kelk S, Mnich M (2009) Uniqueness, intractability and exact algorithms: reflections on level-

k phylogenetic networks. J Bioinform Comput Biol 7(4):597–623. doi:10.1142/S0219720009004308
29. van Iersel L, Kelk S, Rupp R, Huson D (2010) Phylogenetic networks do not need to be complex: using fewer reticulations to represent conflicting clusters. Bioinformatics 26(12):i124–i131. doi:10.1093/bioinformatics/btq202

Beyond Hypergraph Dualization

Lhouari Nourine[1] and Jean-Marc Petit[2]
[1]Clermont-Université, Université Blaise Pascal, LIMOS, CNRS, Aubière, France
[2]Université de Lyon, CNRS, INSA Lyon, LIRIS, Lyon, France

Keywords

Dualization; Enumeration; Hypergraph transversal; Partially ordered set

Years and Authors of Summarized Original Work

2012; Nourine, Petit

Problem Definition

This problem concerns hypergraph dualization and generalization to poset dualization.

A *hypergraph* $\mathcal{H} = (V, \mathcal{E})$ consists of a finite collection $\mathcal{E}$ of sets over a finite set V, i.e., $\mathcal{E} \subseteq \mathcal{P}(V)$ (the powerset of V). The elements of $\mathcal{E}$ are called *hyperedges*, or simply *edges*. A hypergraph is said simple if none of its edges is contained within another. A *transversal* (or *hitting set*) of $\mathcal{H}$ is a set $T \subseteq V$ that intersects every edge of $\mathcal{E}$. A transversal is *minimal* if it does not contain any other transversal as a subset. The set of all minimal transversal of $\mathcal{H}$ is denoted by $Tr(\mathcal{H})$. The hypergraph $(V, Tr(\mathcal{H}))$ is called the *transversal hypergraph* of $\mathcal{H}$. Given a simple hypergraph $\mathcal{H}$, the hypergraph dualization problem (TRANS-ENUM for short) concerns the enumeration without repetitions of $Tr(\mathcal{H})$.

The TRANS-ENUM problem can also be formulated as a dualization problem in posets. Let $(P, \leq)$ be a poset (i.e., $\leq$ is a reflexive, antisymmetric, and transitive relation on the set P). For $A \subseteq P$, $\downarrow A$ (resp. $\uparrow A$) is the downward (resp. upward) closure of A under the relation $\leq$ (i.e., $\downarrow A$ is an ideal and $\uparrow A$ a filter of $(P, \leq)$). Two antichains $(\mathcal{B}^+, \mathcal{B}^-)$ of P are said to be dual if $\downarrow \mathcal{B}^+ \cup \uparrow \mathcal{B}^- = P$ and $\downarrow \mathcal{B}^+ \cap \uparrow \mathcal{B}^- = \emptyset$. Given an implicit description of a poset P and an antichain $\mathcal{B}^+$ (resp. $\mathcal{B}^-$) of P, the poset dualization problem (DUAL-ENUM for short) enumerates the set $\mathcal{B}^-$ (resp. $\mathcal{B}^+$), denoted by $\text{Dual}(\mathcal{B}^+) = \mathcal{B}^-$ (resp. $\text{Dual}(\mathcal{B}^-) = \mathcal{B}^+$). Notice that the function dual is self-dual or idempotent, i.e., $\text{Dual}(\text{Dual}(\mathcal{B})) = \mathcal{B}$.

TRANS-ENUM is a particular case of DUAL-ENUM. Indeed, consider P the poset $(\mathcal{P}(V), \subseteq)$ for some set V. Then for every dual set $(\mathcal{B}^+, \mathcal{B}^-)$ of P, we have $\mathcal{B}^- = Tr(\overline{\mathcal{B}^+}) = \text{Dual}(\mathcal{B}^+)$, or equivalently $\mathcal{B}^+ = \overline{Tr(\mathcal{B}^-)} = \text{Dual}(\mathcal{B}^-)$ with $\overline{\mathcal{E}} = \{V \setminus E \mid E \in \mathcal{E}\}$ where $\mathcal{E} \subseteq \mathcal{P}(V)$.

Now we ask the following question: Which posets DUAL-ENUM can be reduced to TRANS-ENUM? To do so, we introduce the notions of duality gap, convex embedding, and poset reflexion.

Let $(P, \leq P)$ and $(Q, \leq_Q)$ be two posets and $f : P \to Q$ an *injective reflection*, i.e., for all $x, y \in P, f(x) \leq_Q f(y)$ implies $x \leq_P y$. Notice that the reflection f preserves incomparability, i.e., if x and y are incomparable in P, then $f(x)$ and $f(y)$ are incomparable in Q. Therefore, for every dual set $(\mathcal{B}^+, \mathcal{B}^-)$ of P, $\text{Dual}(f(\mathcal{B}^+))$ contains $f(\mathcal{B}^-)$. The difference between the size of $\text{Dual}(f(\mathcal{B}^+))$ and the size of $f(\mathcal{B}^-)$ is a positive integer, called the *duality gap*. We speak about *weak duality* when the gap is strictly positive, *strong duality* otherwise.

Duality gaps are important in enumeration problems because they provide an upper bound on the difference between the number of enumerated solutions and the number of solutions of the original problem.

Key Results

TRANS-ENUM has been intensively studied in the last two decades, and several results show that it is equivalent to many problems in computer science area (see the paper by Eiter and Gottlob [3]). The question whether TRANS-ENUM admits an output-polynomial time algorithm is still open. In fact, despite the number of papers on TRANS-ENUM, the best known algorithm is the one by Fredman and Khachiyan [8] which runs in time $O(n^{\log(n)})$ where n is the size of the hypergraph plus the number of minimal transversals. Other results on complexity can be found in [5, 6, 11, 12, 14]. For general posets, it is shown in [7] that the dualization over the products of some posets can be done with the same complexity as TRANS-ENUM. Recently, Nourine and Petit [16] have investigated dualization problems in general posets for which the duality gap is bounded by a polynomial.

Strong Duality

The following characterization theorem of the zero gap is a reformulation of a known result in [10, 15], where the poset Q is the powerset for some set.

Theorem 1 *Let $(P, \leq_P)$ and $(Q, \leq_Q)$ be two posets. Then the duality gap is zero iff there exists a map $f : P \to Q$ such that f is a bijective embedding, i.e., for all $x, y \in P$ $f(x) \leq_Q f(y)$ iff $x \leq_P y$.*

Many instances of problems have such a property, for example, frequent itemsets, monotone Boolean functions, minimal keys, inclusion dependencies, or minimal dominating sets [10, 13, 15]. Nevertheless, the bijective embedding between two posets does not always exist. In the following we give a relaxation of the bijection embedding in order to capture some polynomial reductions between enumeration problems.

Weak Duality

Let $(P, \leq_P)$ and $(Q, \leq_Q)$ be posets. A function $f : P \to Q$ is a *convex embedding* if for all $x, y \in P$ and $z \in Q$, $x \leq_P y$ iff $f(x) \leq_Q f(y)$ and $f(x) \leq_Q z \leq_Q f(y)$ implies there exists $t \in P$ such that $f(t) = z$.

The following result can be seen as a relaxation of the bijective embedding given in Theorem 1.

Proposition 1 *Let $(P, \leq_P)$ and $(Q, \leq_Q)$ be two posets and $f : P \to Q$ a convex embedding. Then there exist two antichains $\mathcal{B}_0^+$, $\mathcal{B}_0^-$ of Q such that $P \setminus \{\bot_P\}$ is isomorphic to $Q \setminus (\downarrow \mathcal{B}_0^+ \cup \uparrow \mathcal{B}_0^-)$, where $\bot_P$ is the bottom of P if it exists. Furthermore, the duality gap is bounded by $| \mathcal{B}_0^+ | + | \mathcal{B}_0^- |$.*

Complexity

For strong duality, [10, 15] points out how the result of Fredman and Khachiyan [8] can be reused to devise an incremental quasi-polynomial time algorithm, called `Dualize and Advance`, for some pattern mining problems. For weak duality, whenever the duality gap remains polynomial in the size of the problem and $(Q, \leq_Q)$ isomorphic to $(\mathcal{P}(E), \subseteq)$ for some set E, the `Dualize and Advance` algorithm can be reused with the same complexity if the following assumptions hold:

1. The reflexion f of $(P, \leq)$ to $(\mathcal{P}(E), \subseteq)$ and its inverse is computable in polynomial time.
2. Given two elements $x, y \in P$, checking $x \leq y$ is polynomial time.

Applications

The hypergraph dualization is a crucial step in many applications in logics, databases, artificial intelligence, and pattern mining [3, 4, 8, 11, 15], especially for hypergraphs, i.e., Boolean lattices. The main application domain concerns pattern mining problems, i.e., the identification of maximal interesting patterns in database by asking membership queries (predicate) to a database. In the rest of this section, we give two examples of pattern mining problems related to DUAL-ENUM and weak duality.

Frequent Conjunctive Queries

We consider the problem statement defined in [9]. Let $\mathbf{R} = \{R_1, \ldots, R_n\}$ be a database schema, $\mathcal{D}$ the domain of $\mathbf{R}$ and $\text{sch}(\mathbf{R}) = \{R_i.A | R_i \in \mathbf{R}, A \in R_i\}$. A (simple) conjunctive query Q

over $\mathbf{R}$ is of the form $\pi_X(\sigma_F(R_1 \times \ldots \times R_n))$ ($\pi_X(\sigma_F)$ for short) where $X \subseteq \text{sch}(\mathbf{R})$ and F a conjunction of equalities of the form $R_i.A = R_j.B$ or $R_i.A = c$ with $R_i.A, R_j.B \in \text{sch}(\mathbf{R})$ and $c \in \mathcal{D}$. Let $\mathcal{Q}_r$ be the set of all possible conjunctive queries over $\mathbf{R}$. For a given database d over $\mathbf{R}$, we note $\text{Adom}(d) \subseteq \mathcal{D}$ is the active domain of d and $Q(d)$ the result of the evaluation of Q against d. We note $\mathcal{F}$ is the finite set of all possible selection formula over $\mathbf{R}$ and $\text{Adom}(d)$, i.e., $\mathcal{F} = \{\{A, B\} \mid A \neq B, A \in \mathbf{R}, B \in \mathbf{R} \cup \text{Adom}(d)\}$.

Let Q_1, Q_2 be two conjunctive queries over $\mathbf{R}$. Q_1 is *contained* in Q_2, denoted by $Q_1 \subseteq Q_2$, if for every database d over $\mathbf{R}$, $Q_1(d) \subseteq Q_2(d)$. Q_1 is *diagonally contained* in Q_2, denoted $Q_1 \subseteq^{\Delta} Q_2$, if Q_1 is contained in a projection of Q_2, i.e., $Q_1 \subseteq \pi_X(Q_2)$. The frequency of $\pi_X(\sigma_F)$ in d is defined by $|\pi_X(\sigma_F)(d)|$. A query $\pi_X(\sigma_F)$ is *frequent* in d with respect to a given threshold ϵ if $|\pi_X(\sigma_F)(d)| \geq \epsilon$. The frequency is anti-monotonic with respect to $\subseteq^{\Delta}$ [9].

Proposition 2 *Let $Q_1 = \pi_{X_1}(\sigma_{F_1})$ and $Q_2 = \pi_{X_2}(\sigma_{F_2})$ be two queries of $\mathcal{Q}_r$. Then $Q_1 \subseteq^{\Delta} Q_2$ iff $X_1 \subseteq X_2$ and $F_2 \subseteq F_1$. Equivalently, $Q_1 \subseteq^{\Delta} Q_2$ iff $X_1 \cup (\mathcal{F} \setminus F_1) \subseteq X_2 \cup (\mathcal{F} \setminus F_2)$.*

From Proposition 2, $f : \mathcal{Q}_r \to \mathcal{P}(\mathbf{R} \cup \mathcal{F})$ with $f(\pi_X(\sigma_F)) = X \cup (\mathcal{F} \setminus F)$ is a bijective embedding. Thus $\mathcal{Q}_r$ ordered under $\subseteq^{\Delta}$ is a Boolean lattice and Theorem 1 can be applied. It is interesting to consider the subclass of $\mathcal{Q}_r$ restricted to *consistent queries*, i.e., queries for which there exists at least one database such that their evaluations return values different from zero. For instance, $\sigma(B = 1 \wedge B = 2)$ and $\sigma(A = B \wedge A = 1 \wedge B = 2)$ are not consistent. Let us consider the set $\mathcal{Q}_C \subset \mathcal{Q}_r$ of all consistent queries.

Lemma 1 *Let $Q_1 = \pi_{X_1}(\sigma_{F_1})$ and $Q_2 = \pi_{X_2}(\sigma_{F_2})$ be two queries of $\mathcal{Q}_r$ such that $Q_1 \subseteq^{\Delta} Q_2$. Then if Q_2 is consistent, it implies Q_1 is consistent.*

Notice that the restriction of f to $\mathcal{Q}_C$ is still a convex embedding, but no longer bijective. More interestingly, the associated duality gap is not polynomial. Indeed, $B_0^+ = \emptyset$ but B_0^- has a size exponential in the size of $\mathbf{R} \cup \text{Adom}(d)$ since the number of selections of the form $\sigma(A_1 = A_2 \wedge \ldots \wedge A_{n-1} = A_n \wedge A_1 = v \wedge A_n = v')$ is exponential in the number of attributes.

Rigid Sequences

Let us consider sequences with or without wildcard (denoted $\star$); see, e.g., [1]. Let Σ be an alphabet and $\star \notin \Sigma$. A rigid sequence $s[n]$ is a word of size n of $(\Sigma \cup \{\star\})^*$ such that $s[1] \neq \star$ and $s[n] \neq \star$. The set of all rigid sequences of size at most n are denoted by Σ_R^n and the empty sequence by ϵ. Let $s[l], t[k] \in \Sigma_R^n$. We consider the following classical (prefix and factor) partial orders on rigid sequences:

- $s \sqsubseteq_f t$, if there exists $j \in [1 \ldots k]$ such that for every $i \in [1 \ldots l]$, either $s[i] = t[j+i-1]$ or $s[i] = \star$ (factor).
- $s \sqsubseteq_p t$, if for every $i \in [1 \ldots l]$, either $s[i] = t[i]$ or $s[i] = \star$ (prefix).

The following theorem shows that the duality gap between the dualization in prefix posets of rigid sequences and TRANS-ENUM is bounded by a polynomial in n and $|\Sigma|$.

Theorem 2 ([16]) *Let $f : (\Sigma_R^n \setminus \{\epsilon\}, \sqsubseteq_p) \to (\mathcal{P}(\{1, \ldots, n\} \times \Sigma), \subseteq)$ be a function defined by $f(s) = \{(i, s[i]) \mid s[i] \neq \star, i \leq n\}$. Then f is a convex embedding with $\mathcal{B}_0^+ = \{\{(i, x) \mid x \in \Sigma, i \in [2 \ldots n]\}\}$ and $\mathcal{B}_0^- = \{\{(1, x), (1, y)\} \mid x, y \in \Sigma, x \neq y\} \cup \{\{(1, x), (i, y), (i, z)\} \mid x, y, z \in \Sigma, y \neq z, i \in [2 \ldots n]\}$.*

Proposition 3 ([16]) *There is a poset reflection $f : (\Sigma_R^n, \sqsubseteq_f) \to (\Sigma_R^n, \sqsubseteq_p)$ with a duality gap bounded by a polynomial in n.*

Using Theorem 2 and Proposition 3, we conclude that the duality gap between the dualization in factor posets of rigid sequences and TRANS-ENUM is bounded by a polynomial the size of Σ and n [16].

Open Problems

1. The challenging question is to find an output-polynomial time algorithm for TRANS-ENUM.

2. Lattices are a particular class of posets. For example, the dualization over product of chains can be done with the same complexity as TRANS-ENUM which is equivalent to dualization in Boolean lattices. For distributive lattices class which contains Boolean lattice and the product of chains, the dualization is open.
3. Many connections have to be done between TRANS-ENUM and graph theory problems, such as minimal dominating sets [13].
4. Many problems in data mining can be formulated as dualization in posets, e.g., frequent subgraphs or frequent subtrees. An interesting direction is to identify posets for which the dualization is equivalent to TRANS-ENUM.

URLs to Code and Data Sets

Program Codes and Instances for Hypergraph Dualization can be found on the Takeaki Uno's webpage at http://research.nii.ac.jp/~uno/dualization.html. Some pattern mining problems, reducible to TRANS-ENUM with strong duality, can be found on the iZi webpage at http://liris.cnrs.fr/izi/.

Cross-References

- Efficient Polynomial Time Approximation Scheme for Scheduling Jobs on Uniform Processors
- Minimal Dominating Set Enumeration

Recommended Reading

1. Arimura H, Uno T (2009) Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In: SDM, Sparks, pp 1087–1098
2. Boros E, Makino K (2009) A fast and simple parallel algorithm for the monotone duality problem. In: Albers S, Marchetti-Spaccamela A, Matias Y, Nikoletseas SE, Thomas W (eds) ICALP (Part I), Rhodes. LNCS, vol 5555. Springer, pp 183–194
3. Eiter T, Gottlob G (1995) Identifying the minimal transversals of a hypergraph and related problems. SIAM J Comput 24(6):1278–1304
4. Eiter T, Gottlob G, Makino K (2003) New results on monotone dualization and generating hypergraph transversals. SIAM J Comput 32:514–537
5. Eiter T, Gottlob G, Makino K (2003) New results on monotone dualization and generating hypergraph transversals. SIAM J Comput 32(2):514–537
6. Elbassioni KM (2008) On the complexity of monotone dualization and generating minimal hypergraph transversals. Discret Appl Math 156(11): 2109–2123
7. Elbassioni KM (2009) Algorithms for dualization over products of partially ordered sets. SIAM J Discret Math 23(1):487–510
8. Fredman ML, Khachiyan L (1996) On the complexity of dualization of monotone disjunctive normal forms. J Algorithms 21(3):618–628
9. Goethals B, Van den Bussche J (2002) Relational association rules: getting warmer. In: Pattern detection and discovery, London, pp 125–139
10. Gunopulos D, Khardon R, Mannila H, Saluja S, Toivonen H, Sharm RS (2003) Discovering all most specific sentences. ACM Trans Database Syst 28(2):140–174
11. Gottlob G (2013) Deciding monotone duality and identifying frequent itemsets in quadratic logspace. In: PODS, NewYork, pp 25–36
12. Hagen M (2008) Algorithmic and computational complexity issues of MONET. Cuvillier Verlag ISBN 978-3-86727-826-3, 141 pages
13. Kanté MM, Limouzy V, Mary A, Nourine L (2014) On the enumeration of minimal dominating sets and related notions To appear in SIAM on Discrete Math.
14. Kavvadias DJ, Stavropoulos EC (2003) Monotone boolean dualization is in co-NP[log2n]. Inf Process Lett 85:1–6
15. Mannila H, Toivonen H (1997) Levelwise search and borders of theories in knowledge discovery. Data Min Knowl Discov 1(3):241–258
16. Nourine L, Petit J-M. (2012) Extending set-based dualization: application to pattern mining. In: ECAI Montpellier, pp 630–635

Beyond Worst Case Sensitivity in Private Data Analysis

Abhradeep Thakurta
Department of Computer Science, Stanford University, Stanford, CA, USA
Microsoft Research, CA, USA

Keywords

Algorithmic stability; Differential privacy; Robust estimators

Years and Authors of Summarized Original Work

2006; Cynthia Dwork, Frank McSherry, Kobbi Nissim, Adam Smith
2007; Kobbi Nissim, Sofya Raskhodnikova, Adam Smith
2009; Cynthia Dwork, Jing Lei
2013; Adam Smith, Abhradeep Thakurta

Problem Definition

Over the last few years, differential privacy [5, 6] has emerged as one of the most accepted notions of statistical data privacy. At a high level differential privacy ensures that from the output of an algorithm executed on a data set of potentially sensitive records, an adversary learns "almost" the same thing about an individual irrespective of his presence or absence in the data set. Formally, differential privacy is defined below (Definition 1). Setting the privacy parameters $\epsilon < 1$ and $\delta \ll \frac{1}{n^2}$ ensures semantically meaningful privacy guarantees. For a detailed survey on the semantics of differential privacy, see [2, 3, 8, 9].

Definition 1 ((ϵ, δ)-differential privacy [5, 6]) We call two data sets D and D' (with n records from a fixed domain τ) neighboring if they differ in exactly one entry, i.e., $|D \triangle D'| = 2$. An algorithm $\mathcal{A}$ is (ϵ, δ)-differentially private if, for all neighboring data sets D and D' and for all measurable events S in the range space of $\mathcal{A}$, we have

$$\Pr[\mathcal{A}(D) \in S] \leq e^{\epsilon} \Pr[\mathcal{A}(D') \in S] + \delta.$$

Initial efforts towards designing differentially private algorithms have concentrated on settings where the algorithms enjoy the same utility guarantees for *any* data set from the domain τ^n. (See [3] for a survey on these efforts.) However, due to the pessimistic nature of these algorithms, some perform poorly in non-worst-case scenarios. With the seminal work of [11], and followed by a series of results [4, 7, 10, 12, 13], the community started focusing on designing differentially private algorithms which are more useful in non-worst-case settings, but in pessimistic scenarios may only perform as poorly as the worst case algorithms. In this entry, we provide an overview of some of the recent efforts in this line of research.

Computing the Median: A Motivating Example

To provide a flavor of the nature of these algorithms, we start with the following simple example: Given a data set $D = \{d_1, \ldots, d_n\}$ of n real numbers in $[0, R]$ (with $R \in \mathbb{R}^+$ and n being odd), the task is to compute the median of D while preserving differential privacy. Notice that in the worst case, changing one entry in D can change the median by R. So intuitively, any algorithm that does not distinguish between worst case and non-worst-case scenario will introduce an error $\Omega(R)$ in the output.

Without loss of generality, assume that $d_1 \leq \cdots \leq d_n$ and let $m = \frac{n+1}{2}$. Now it is not hard to observe that by changing one data entry in D, the median d_m can change by at most $\max\{d_m - d_{m-1}, d_{m+1} - d_m\}$, which can be potentially much smaller than R. An algorithm that takes advantage of this observation can be much more useful in non-worst-case settings as compared to an algorithm that always introduces an error of $\Theta(R)$. With this example in mind, in the following section, we define some of the notions in the differential privacy literature that capture the non-worst-case change in the output of a given computation task on neighboring data sets.

Although the median might seem to be a very simple example, but interestingly, this intuition of capturing non-worst-case change extends to a large class of problems. Especially in many machine learning settings, where even for the non-private algorithms the error guarantees are over distributional assumptions on the data [1], this intuition is very helpful in designing effective differentially private variants.

N.B. Notice that in the case of computing the average, there is no distinction between worst case and non-worst change. Hence it is not a good example for the scenarios we address in this entry.

Notions of Sensitivity

We describe some of the concepts which help us capture non-worst-case changes in the output of a given function $f : \tau^n \to \mathbb{R}$ on pairs of neighboring data sets D and D'. Later we use them to design differentially private algorithms which capture non-worst-case behavior of the data sets.

Global Sensitivity [6]

This notion of sensitivity refers to the maximum change the function f can have on any pair of neighboring data sets from the domain. Formally,

$$\mathsf{GS}(f) = \max_{D, D' \in \tau^n, |D \triangle D'| = 2} |f(D) - f(D')|. \tag{1}$$

The following algorithm in (2) is $(\epsilon, 0)$-differentially private. In the literature this is also called the *Laplace mechanism* [6]. Here $\mathsf{Lap}(\lambda)$ refers to the Laplace distribution with standard deviation $\sqrt{2}\lambda$.

$$\textbf{Output}: f(D) + \mathsf{Lap}\left(\frac{\mathsf{GS}(f)}{\epsilon}\right). \tag{2}$$

Notice that in (2) the distribution on the noise that is added is the same for all data sets. In general these style of algorithms that introduce data independent randomness have weaker utility guarantees in non-worst-case scenarios.

Local Sensitivity

While global sensitivity captures the maximum change in the output of f for any pairs of neighboring data sets, local sensitivity relaxes this notion to capture the maximum change in the output of f for any neighboring data set of a given data set D. Formally,

$$\mathsf{LS}(f, D) = \max_{D' \in \tau^n, |D \triangle D'| = 2} |f(D) - f(D')|. \tag{3}$$

With the similarity between the definitions of local sensitivity and global sensitivity, it might be tempting to use the same algorithm as (2), with the GS replaced by LS. A careful observation indicates that this algorithm cannot be (ϵ, δ)-differentially private for any non-trivial choices of ϵ and δ. Consider the following setting where the data domain is $\{0, 1\}^n$, and the function f is the median value of D. Let D be a data set with $\lfloor \frac{n}{2} - 1 \rfloor$ entries as zero, and the rest as one. When n is odd, clearly $\mathsf{LS}(D)$ equals zero. But for a data set D' formed by changing one of the zeros in D to one, $\mathsf{LS}(D')$ equals one. So, if we replace GS by LS in (2), then for D there will be zero noise added, and for D' the noise will be $\Omega(1/\epsilon)$. Differential privacy prohibits this.

The counterexample above might give an impression that local sensitivity may not be a useful concept. However, in the following and in Algorithm 2, we show that one can use local sensitivity to obtain effective differentially private algorithms which are more useful in non-worst-case scenarios.

Smooth Sensitivity [11]

In the above example, we noticed that a direct use of local sensitivity in noise addition can result in trouble. However, using a related notion called smooth sensitivity, one can obtain a variant of the algorithm in (2) which is both differentially private and respects local properties of a given data set. At a high level, smooth sensitivity is an envelope over the local sensitivity which helps avoid abrupt change in the variance of the noise on neighboring data sets. Formally,

$$\mathsf{SS}(f, D, \beta) = \max_{D' \in \tau^n} \mathsf{LS}(f, D) e^{-\beta \cdot \mathsf{dist}(D, D')/2}. \tag{4}$$

Here dist is the symmetric difference between the two data sets D and D' and $\beta > 0$ is the smoothness parameter. Following observations on smooth sensitivity will be useful for designing differentially private algorithms.

1. **Observation 1 (Envelope on LS):** $\forall D, \beta > 0, \mathsf{SS}(f, D, \beta) \geq \mathsf{LS}(f, D)$.
2. **Observation 2 (Smaller than GS):** $\forall \beta > 0, D \in \tau^n, \mathsf{SS}(f, D, \beta) \leq \mathsf{GS}(f)$.
3. **Observation 3 (Smoothness):** For all neighbors D, D', $\mathsf{SS}(f, D, \beta) \leq e^{\beta} \mathsf{SS}(f, D', \beta)$.

Key Results

Using the concepts of local sensitivity and smooth sensitivity defined in the previous section, we provide two differentially private algorithmic frameworks which respect local (non-worst-case) properties of a given data set. Later in Applications 1 and 2, we instantiate them with specific problems.

Algorithm 1: Smooth Sensitivity Based

In order to use the notion of smooth sensitivity to design a differentially private algorithm analogous to (2), we need the following properties from the noise distribution to be added to $f(D)$. Let us define the following notation: For a subset S of $\mathbb{R}$, the set $S + \Delta$ defines $\{z + \Delta : z \in S\}$, and the set $e^{\lambda} \cdot S$ defines the set $\{e^{\lambda} \cdot z : z \in S\}$.

Definition 2 (Admissible noise distribution [11]) A probability distribution h on $\mathbb{R}$ is (α, β)-admissible if, for $\alpha = \alpha(\epsilon, \delta), \beta = \beta(\epsilon, \delta)$, the following conditions hold for all $|\Delta| \leq \alpha$ and $|\lambda| \leq \beta$, and for all subsets $S \subseteq \mathbb{R}$.

1. **Sliding property:** $\Pr_{Z \sim h}[Z \in S] \leq e^{\epsilon/2} \Pr_{Z \sim h}[Z \in S + \Delta] + \frac{\delta}{2}$.
2. **Dialation property:** $\Pr_{Z \sim h}[Z \in S] \leq e^{\epsilon/2} \Pr_{Z \sim h}\left[Z \in e^{\lambda} \cdot S\right] + \frac{\delta}{2}$.

With Definition 2 in hand, now we can define an algorithm which is analogous to (2). Let h be an (α, β)-admissible noise distribution and let Z be an independent sample from h. For a given data set D, the algorithm is as follows:

$$\textbf{Output}: f(D) + \frac{\mathsf{SS}(f, D, \beta)}{\alpha} \cdot Z. \qquad (5)$$

One can show that the above algorithm is (ϵ, δ)-differentially private [11]. An immediate question that arises: *Which natural distributions satisfy this property?* [11] showed that Laplace distribution $\frac{1}{2}e^{-|z|}$ is $\left(\epsilon/2, \frac{\epsilon}{2\ln(1/\delta)}\right)$-admissible, and $\mathcal{N}(0, 1)$ is $\left(\epsilon/\sqrt{\ln(1/\delta)}, \frac{\epsilon}{2\ln(1/\delta)}\right)$-admissible. Later we will see a concrete instantiation of (5) for the median problem.

Algorithm 2: Propose-Test-Release (PTR) Framework

In the previous section we saw a "noise-addition"-based algorithm that exploited the smooth upper bound on the local sensitivity to ensure differentially privacy. In this section, instead of obtaining a smooth bound on the local sensitivity, we seek an answer to the following question: *Given a proposed upper bound Λ on the local sensitivity of $f(D)$, how many data points (k) in D need to be changed to increase the local sensitivity beyond Λ?* If k is sufficiently large, then the algorithm uses the proposed bound Λ in (2) instead of $\mathsf{GS}(f)$; otherwise the algorithm outputs a $\perp$ and fails. Once formalized, this algorithmic paradigm can be shown to be (ϵ, δ)-differentially private. A major component in the design of algorithms using this paradigm is to come up with tight upper bounds on the local sensitivity. In Applications 1 and 2 we state two approaches for getting such bounds.

In the following, we formally introduce the propose-test-release framework. The version in Algorithm 1 is a variant of the ones appeared in [4] and [13].

Algorithm 1 Propose-test-release (PTR) framework

Input: Data set: $D \in \tau^n$, function $f : \tau^n \to \mathbb{R}$, proposed local sensitivity bound: Λ, privacy parameters: (ϵ, δ).

1: **Distance to instability:** $\mathsf{dist} \leftarrow$ Minimum $k \in [n]$ for which $\left[\max_{D', D \triangle D' = 2k} \mathsf{LS}(f, D')\right] > \Lambda$.
2: **Noisy distance to instability:** $\widetilde{\mathsf{dist}} \leftarrow \mathsf{dist} + \mathsf{Lap}\left(\frac{1}{\epsilon}\right)$.
3: **Test and release:** If $\widetilde{\mathsf{dist}} > \frac{1}{\epsilon}\log(1/\delta)$, then output Λ, else output $\perp$.

One can show that Algorithm 1 is (ϵ, δ)-differentially private. (See [13] for more details.) Additionally, by the tail properties of Laplace distribution, it is not hard to show that if $\mathsf{dist} > \frac{2}{\epsilon}\log(1/\delta)$, then with probability at least $1 - \delta$, the algorithm outputs Λ. In (6) we fit Algorithm 1 with the Laplace mechanism from (2) to obtain a differentially private estimate of $f(D)$. By the composition property of differential privacy [4, 5], the algorithm

(*PTR+Laplace mechanism*) in (6) is $(2\epsilon, \delta)$-differentially private.

$$\text{If PTR}(f, D, \Lambda, \epsilon, \delta) \neq \perp, \text{ then } \textbf{output } f(D) + \mathsf{Lap}\left(\frac{\Lambda}{\epsilon}\right), \text{ else } \textbf{fail}. \tag{6}$$

One can notice that if the proposed bound Λ is much smaller than $\mathsf{GS}(f)$, then whenever the algorithm succeeds, it would add much lesser noise to $f(D)$ as compared to (2). In Application 1 we will do a comparison between the global sensitivity based, the smooth sensitivity based, and the PTR-based algorithm for the problem of computing the median.

Application 1: Computing the Median

With the smooth sensitivity-based and the PTR-based algorithmic frameworks from the previous section in hand, we revisit the problem of computing the median. Let the data set $D = \{d_1, \ldots, d_n\} \in [0, R]^n$ with n being odd and $R \in \mathbb{R}$ being the range. W.l.o.g., assume that the entries in D are sorted in *ascending order*.

Smooth sensitivity-based algorithm for median computation. In order to use (5), we need to be able to efficiently compute the smooth sensitivity (4) of the median function for D with a given smoothness parameter β. The following theorem implies an $O(n \log n)$ algorithm for computing the smooth sensitivity.

Theorem 1 ([11]) *Let $m = \frac{n+1}{2}$. The smooth sensitivity of the median function with the smoothness parameter β is given by the following.*

$$\mathsf{SS}(\mathsf{Median}, D, \beta) = \max_{k=0,\ldots,n} \times \left(e^{-k\beta} \max_{t=0,\ldots,k+1} (d_{m+t} - d_{m+t-k-1})\right).$$

It can be computed in time $O(n \log n)$.

Once the smooth sensitivity bound is obtained, one can use it in (5) to obtain a differentially private approximation to the median of D. If Laplace distribution $(\frac{1}{2}e^{-|z|})$ is used as the noise, then set the admissible parameters $\alpha = \epsilon/2$ and $\beta = \frac{\epsilon}{2\ln(1/\delta)}$. An immediate question that arises is how does the noise added by the smooth sensitivity-based algorithm compare to the global sensitivity-based algorithm in (1). First notice that since global sensitivity is always an upper bound on smooth sensitivity, the noise added via smooth sensitivity can never be more than that via global sensitivity. Next we present a setting of the data set D where in fact the smooth sensitivity-based algorithm adds much lesser noise (and hence more accurate).

Consider the data set D where each $d_i = \frac{R \cdot i}{n}$ for all $i \in [n]$. In this case the term $A_k = \max_{t=0,\ldots,k+1} (d_{m+t} - d_{m+t-k-1}) = \frac{(k+1)R}{n}$. Thus the term $e^{-k\beta} A_k$ is maximized when $k = \frac{1}{\beta} - 1$. Assuming $\beta < 1$, the smooth sensitivity $\mathsf{SS}(\mathsf{Median}, D, \beta)$ is bounded by $\frac{R}{\beta n}$. If we use Laplace distribution in (5) to ensure (ϵ, δ)-differential privacy, then the noise that gets added to $\mathsf{Median}(D)$ is $O\left(\frac{R \log(1/\delta)}{\epsilon^2 n}\right)$. In comparison, the global sensitivity-based algorithm in (2) will add noise $O\left(\frac{R}{\epsilon}\right)$, which is much higher.

One might argue that the global sensitivity-based algorithm guarantees stronger differential privacy ($(\epsilon, 0)$ as opposed to (ϵ, δ)) and hence it is not a fair comparison. Even when one uses a more concentrated noise distribution like Gaussian distribution (which ensures (ϵ, δ)-differential privacy) instead of Laplace distribution in (2), the error still remains the same.

PTR-based algorithm for computing the median. We now instantiate the PTR-based algorithm for the same problem of computing the median. In order to do so, we first partition the real line $\mathbb{R}$ into bins of width $h = \frac{R}{n^{1/3}}$ (or any width $\frac{R}{n^{1/2+\gamma}}$ for any $\gamma > 0$). Call this set of bins $\mathcal{B}$. Additionally consider the set $\mathcal{B}_{(+h/2)}$, which is the set of bins shifted by $h/2$.

In Algorithm 1 we set the proposed sensitivity bound $\Lambda = h$. We compute the *distance to*

instability in Line 1 of Algorithm 1 by the following technique. Let k_1 be the minimum number of entries in D that needs to be changed to move the median from its bin in the set $\mathcal{B}$, and let k_2 be the corresponding minimum number for the set $\mathcal{B}_{(+h/2)}$. The distance to instability is $\mathsf{dist} \leftarrow \max\{k_1, k_2\}$. Now the rest of the algorithm follows as described for the PTR framework. The two sets of shifted bins $\mathcal{B}$ and $\mathcal{B}_{(+h/2)}$ were needed because the median might fall at the partition boundary of the bins. Notice that computing dist takes $O(n)$ time.

In terms of the utility guarantee for this algorithm, we have the following:

Theorem 2 ([4]) *Let the data set D be drawn i.i.d. from some fixed distribution $\mathcal{P}$, where the cumulative distribution function of $\mathcal{P}$ is differentiable with positive derivative at the median. Assuming the privacy parameter $\delta = 1/\mathsf{poly}(n)$, we have the following utility guarantees for the PTR-based median computation.*

$$\Pr[\mathsf{PTR}(D) = \perp] = O(e^{-\epsilon \log n})$$

and $\mathsf{PTR}(D)$ converges in probability to the median of $\mathcal{P}$ as $n \to \infty$.

Application 2: Selection from a Discrete Set

In this section we will see another application of the PTR framework. Although the exposition is fairly abstract, we will see that this tool is useful for a variety of machine learning problems, where we assume "very little" about the underlying learning algorithm. Some of the examples being sparse estimation, parameter tuning, and non-convex learning.

Given a data set $D \in \tau^n$, and a choice function $f : \tau^n \to \{S_1, \ldots, S_k\}$, the objective is to compute a differentially private approximation to $f(D)$. Here $\{S_1, \ldots, S_k\}$ form a discrete set of choices. In order to design the private algorithm, we instantiate the PTR framework in Algorithm 1, with $\Lambda = 0$ and f being the choice function. (Notice that $\Lambda = 0$ means that the output of the function does not change *at all* by changing any one entry in the data set.) If the output of the PTR framework is *not equal to* $\perp$, then output $f(D)$ exactly, and output $\perp$ otherwise. From the privacy property of the PTR framework, it follows that the above algorithm is (ϵ, δ)-differentially private. In terms of utility one can show the following.

Theorem 3 ([13]) *If the distance to instability of the choice function (Line 1 of Algorithm 1) is at least $2\log(1/\delta)$, then with probability at least $1 - \delta$, the above PTR instantiation outputs $f(D)$ exactly.*

At a high level, Theorem 3 says that if one needs to change sufficient number of entries $(2\log(1/\delta))$ in the data set D to change $f(D)$, then with high probability the PTR framework will output $f(D)$ exactly. One issue with the current instantiation of the PTR framework is that it is not clear a priori how to efficiently compute the distance to instability in Line 1 of Algorithm 1. In the following we circumvent this problem by instantiating the PTR framework with a proxy function $\hat{f}$ instead of f, for which the distance to instability is always efficiently computable. Moreover, if on a (sufficiently large) random subset D_{sub} of D, with probability at least $3/4$ one can guarantee $f(D_{\mathsf{sub}}) = f(D)$, then the PTR framework outputs $f(D)$ exactly with high probability.

Subsample and aggregate framework. The basic idea of subsample aggregate framework first appeared in [11] and the current version is from [13]. Here we use a variant of that framework for instantiating the proxy function $\hat{f}$ corresponding to f.

Let $q = \frac{\epsilon}{32\log(1/\delta)}$ and $m = \frac{\log(n/\delta)}{q^2}$. Sample data sets $D_1, \ldots, D_m$, where each D_i is generated from D by sampling each entry in D with probability q, and D_i's are i.i.d. Let S_{first} be the choice which appears maximum number of times in $\mathcal{F} = \{f(D_1), \ldots, f(D_m)\}$, and let S_{second} be the corresponding second. Let the proxy function

$\hat{f}(D)$ equal S_{first}. Let $\text{count}(S)$ be the number of times the choice S appears in $\mathcal{F}$. One can show that with probability at least $1-\delta$, the distance to instability of $\hat{f}(D)$ equals $\text{dist} \leftarrow \frac{\text{count}(S_{\text{first}})-\text{count}(S_{\text{second}})}{4mq} - 1$. From this, one can conclude that using $\hat{f}$ as the proxy for the choice function f in the PTR framework ensures $(\epsilon, 2\delta)$-differential privacy. In terms of utility, for this "proxy" instantiation of the PTR one can show the following. Notice that in both Theorems 3 and 4, there is no dependence on the number of possible choices (k).

Theorem 4 ([13]) *If for each D_i (defined above) $f(D_i) = f(D)$ with probability at least $3/4$, then with probability at least $1-2\delta$, the above instantiation of the PTR framework outputs $f(D)$ exactly.*

One of the classic setting where the above algorithms can be used in the case of model or feature selection in machine learning. A specific example is the LASSO estimator, where the PTR-based algorithm achieves the *optimal* sample complexity even under the constraint of differential privacy. (See [13] for details.) Another example is in finding the best regularization parameter for a given regression problem (Dwork and Thakurta, Differentially private parameter tuning using subsample and aggregate framework. Personal communication, 2014). Let $\Lambda = \{\lambda_1, \ldots, \lambda_k\}$ be a candidate set of regularization parameters, with each $\lambda_i \in \mathbb{R}$. The idea is to estimate the best regularization parameter from the set Λ for each of the sampled data sets $D_1, \ldots, D_m$, and use the estimation algorithm itself as the choice function f in the PTR framework.

Notice that we almost assumed nothing about the regularization parameter selection algorithm, apart from the fact that on random subsamples of the original data set D, the algorithm selects the same regularization parameter most of the times. The subsampling-based algorithm can also be used in the context of learning non-convex models while preserving differential privacy (Bilenko et al., Private and robust non-convex learning. Personal communication, 2014). For the purpose of brevity, we defer the exposition for differentially private learning with non-convex models.

Reference Notes

The global sensitivity-based and the smooth sensitivity-based algorithmic framework are due to [6] and [11] respectively. The propose-test-release (PTR) framework is initially due to [4], but the exposition in this note is from [4, 13]. The smooth sensitivity-based private median algorithm is due to [11], and the one based on PTR framework is due to [4]. The algorithm for privately selecting from a discrete set is from [13].

Recommended Reading

1. Anthony M, Bartlett PL (2009) Neural network learning: theoretical foundations. Cambridge University Press, Cambridge
2. Dwork C (2006) Differential privacy. In: 33rd international colloquium on automata, languages and programming, Venice, LNCS pp 1–12
3. Dwork C (2011) A firm foundation for private data analysis. Commun ACM 54(1):86–95
4. Dwork C, Lei J (2009) Differential privacy and robust statistics. In: Symposium on theory of computing (STOC), Bethesda, pp 371–380
5. Dwork C, Kenthapadi K, Mcsherry F, Mironov I, Naor M (2006) Our data, ourselves: privacy via distributed noise generation. In: EUROCRYPT. Springer, pp 486–503
6. Dwork C, McSherry F, Nissim K, Smith A (2006) Calibrating noise to sensitivity in private data analysis. In: Theory of cryptography conference. Springer, New York, pp 265–284
7. Hardt M, Roth A (2013) Beyond worst-case analysis in private singular vector computation. In: STOC, Palo Alto
8. Kasiviswanathan SP, Smith A (2008) A note on differential privacy: defining resistance to arbitrary side information. CoRR arXiv:0803.39461 [cs.CR]
9. Kifer D, Machanavajjhala A (2012) A rigorous and customizable framework for privacy. In: Principles of database systems (PODS 2012), Scottsdale
10. Kifer D, Smith A, Thakurta A (2012) Private convex empirical risk minimization and high-dimensional regression. In: Conference on learning theory, Edinburgh, pp 25.1–25.40

11. Nissim K, Raskhodnikova S, Smith A (2007) Smooth sensitivity and sampling in private data analysis. In: Symposium on theory of computing (STOC), San Diego, ACM, pp 75–84. Full paper: http://www.cse.psu.edu/~asmith/pubs/NRS07
12. Smith A (2011) Privacy-preserving statistical estimation with optimal convergence rates. In: Proceedings of the forty-third annual ACM symposium on theory of computing, San Jose, pp 813–822
13. Smith AD, Thakurta A (2013) Differentially private model selection via stability arguments and the robustness of the lasso. J Mach Learn Res Proc Track 30:819–850

BG Distributed Simulation Algorithm

Matthieu Roy
Laboratory of Analysis and Architecture of Systems (LAAS), Centre National de la Recherche Scientifique (CNRS), Université Toulouse, Toulouse, France

Keywords

Computability; Distributed tasks; Read/write shared memory; Reduction

Years and Authors of Summarized Original Work

1993; Borowsky, Gafni
2001; Borowsky, Gafni, Lynch, Rajsbaum

Problem Definition

How to effectively translate an algorithm from a distributed system model to another one?

Distributed systems come in diverse settings that are modeled by different assumptions (1) on the way processes communicate, e.g., using shared memory or messages, (2) on the fault model, (3) on synchrony assumptions, etc. Each of these parameters has a dramatic impact on the computing power of the model, and in practice, an algorithm or an impossibility result is usually tailored to a particular model and cannot be directly reused in another model.

This wide variety of models has given rise to many different impossibility theorems and numerous algorithms for many of the possible combinations of parameters that characterize them. Thus, a crucial question is the following: are there bridges between some models, i.e., is it possible to transfer an impossibility result or an algorithm from one model to another?

The Borowsky-Gafni simulation algorithm, or BG simulation, is one of the first steps toward direct translations of algorithms or impossibility results from one model to another. The BG simulation considers distributed systems made of asynchronous processes that communicate using a shared memory array. In a nutshell, this simulation allows a set of $t + 1$ asynchronous sequential processes, where up to t of them can stop during their execution, to simulate any set of $n \geq t + 1$ processes executing an algorithm that is designed to tolerate to up to t fail-stop failures.

The BG simulation has been used to prove solvability and unsolvability results for crash-prone asynchronous shared memory systems, paving the way for a more generic formal theory of reduction between problems in different models of distributed computing.

The BG-simulation algorithm is named after its authors, Elizabeth Borowsky and Eli Gafni, that introduced it as a side tool [3] in order to generalize the impossibility result of solving a weakened version of consensus, namely, k-set agreement [6]. It has been later on formalized and proven correct [4, 18] using the I/O automata formalism [17].

System Model

Processes

The simulation considers a system made of up to n asynchronous sequential processes that execute a distributed algorithm to solve a given colorless decision task, as defined below.

Failure Model

Processes may fail by stopping (crash failure). The simulation assumes that up to t processes can stop during the execution; $t < n$ is known before the execution, but the identity of processes that may crash is unknown to the simulation. This model of computation is referred to as the t-resilient model. A corner case of this model is the wait-free model where $t + 1$ processes execute concurrently and at most t of them may crash.

Communication

Processes communicate and coordinate using a reliable shared memory composed of n multiple-reader single-writer registers. Each process has the exclusive write access to one of these n registers, and processes can read all entries by invoking a *snapshot* operation, with the semantics that write and snapshot operations appear as if they are executed atomically. While using the snapshot abstraction eases the presentation of the algorithm, it has no impact on the power of the underlying computing model, since the snapshot/write model can be implemented wait-free using read/write registers [1].

Tasks

A colorless task is a distributed coordination problem in which every process p_i starts with a value, communicates with other processes, and has to decide eventually on a output value. Colorless tasks, or convergence tasks [12], are a restricted version of tasks in which a deciding process may adopt the decision value of any process, i.e., two participating processes may decide the same value. For more formal definitions of tasks using tools from algebraic topology, the reader should refer to [11].

Simulation

The simulation proceeds by executing concurrently, using $t + 1$ simulators $s_1, \ldots, s_{t+1}$, the code of $n > t$ processes that collaboratively solve a distributed colorless task. Hence, each simulator s_i is given the code of all simulated processes and handles the execution of n threads.

Key Results

Simulation of Memory

Each one of the $t + 1$ simulators s_i executes the sequential code of the n simulated processes p_j in parallel. By assumption, every simulated code is a sequence of instructions that are either (1) local processing, (2) a write operation into memory, or (3) a snapshot of the shared memory.

Every simulator s_i maintains its local view of the simulated memory for all simulated threads. These local views are synchronized between simulators by writing and reading (using snapshots) in a shared memory matrix array MEM that has one column per simulated thread and one row per *snapshot* instance.

To ensure global consistency between simulators that simulate concurrently all threads, operations on the memory must be coordinated between different simulators. This is achieved by ensuring that, for a given simulated thread, the sequence of snapshots of the memory as computed by all simulators is identical. As consensus cannot be implemented wait-free, the simulation coordinates snapshots using of a weaker form of agreement, the *safe agreement*.

The Safe-Agreement Object

Safe agreement is the most important building block of the simulation. First introduced as the *non-blocking busy-wait* agreement protocol [3], it has been further refined as safe agreement, with several blocking or non-blocking/wait-free implementations [2, 11, 14].

This weak form of agreement provides two methods to processes: *propose*(v) and *resolve*(). A participating process that proposes a value v first calls *propose*(v) once and is then allowed to make calls to *resolve*() that may return $\perp$ if safe agreement is not resolved yet or a value. In this later case, safe agreement is said to be resolved and the value returned is the decided value by the process. Formally, safe agreement is defined by three properties:

Termination: If no process crashes during the execution of *propose*(), then all processes decide, i.e., eventually all calls to *resolve*() return a non-$\perp$ value,

Validity: All processes that decide must decide a proposed value,

Agreement: All processes that decide must decide the same value.

The specification is almost identical to the one of consensus, apart from the weakened termination property. Safe agreement is wait-free solvable and thus solvable in t-resilient systems.

The crucial point of the BG simulation lies in the termination property of safe agreement: if a safe-agreement protocol cannot be resolved, i.e., if no process decides, then at least one process crashed during the call to *propose*(). Thus, a given safe-agreement instance can "capture" a calling process that crashed during the *propose* invocation.

Overview of the Simulation

The current state of the simulation and its history is thus represented by two twin data structures: (1) the shared memory matrix *MEM* that contains the consecutive memory status of all simulated threads, as seen by simulators, and (2) a matrix of safe-agreement objects SafeAgreement$[0\ldots][1\ldots n]$ with n columns, each column representing the execution advancement of one of the simulated processes, as shown in Fig. 1.

In this view, the entry at row ℓ and column i corresponds to the state of the ℓth snapshot for simulated process p_j. Hence, the "program counter" of a simulated thread p_i is the greatest row of column i that is either unresolved or resolved. In this example, simulations of threads p_2, p_4, and p_6 are stopped with unresolved safe agreement that are due to (at least) one simulator stuck in the associated *propose*() methods. The program counter of all other threads is 9.

Each simulator s_i is given the code of the n threads it has to simulate, as well as an input value of one of the threads. Conceptually, the algorithm run by simulator s_i is as follows:

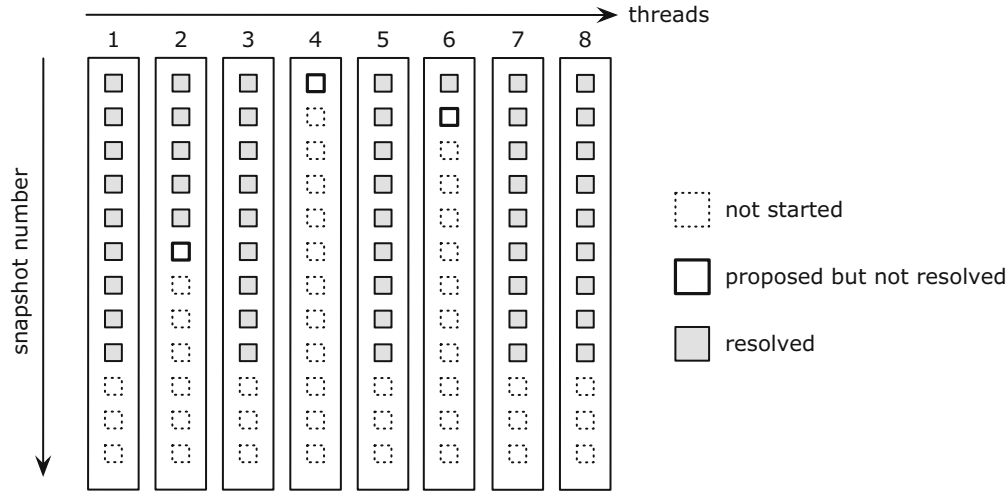
In the simulation, each *snapshot* invocation is mediated through a SafeAgreement object, lines 6 and 1. The only reason that could block the simulation of a given thread p_i is when the call to *resolve*, line 6, always returns $\perp$. By definition of the safe-agreement object, this situation can happen only when a simulator crashed during the call to *propose*() on the same safe-agreement instance: *the crash of a simulator can block the simulation of at most one simulated thread.*

Applications

The BG-simulation algorithm has been primarily used to reduce t-resilient solvability to wait-free

Algorithm 1 BG-simulation: code for a simulator s_j starting with input v

```
1:  procedure BG-SIMULATION(v)
2:      ∀i = 1...n, SafeAgreement[0][i].propose(v)
                                                          ▷ Initialization
3:      loop
4:          for i ← 1, n do                    ▷ Simulate threads in round-robin
5:              ℓ ← current program counter of p_i
6:              snap ← SafeAgreement[ℓ][i].resolve()
7:              if snap ≠ ⊥ then                          ▷ safe agreement is resolved
8:                  perform local computation using snap, write operations in local memory
9:                  execute write on behalf of p_i in MEM[ℓ][i]
10:                 if thread p_i is terminated then
11:                     return value and stop its simulation
12:                 else if at least (n − t) threads have program counter ≥ ℓ then
13:                     snap ← snapshot(MEM[ℓ])
14:                     SafeAgreement[ℓ + 1][i].propose(snap)
15:                 end if
16:             end if
17:         end for
18:     end loop
19: end procedure
```



BG Distributed Simulation Algorithm, Fig. 1 Conceptual view of advancement for snapshots of all simulated process with $n = 8$ and $t = 3$

solvability for colorless tasks, that is, tasks that are agnostic on process identities. The initial application has been made to the k-set agreement problem, in which all processes have to agree on a final set of values of size at most k. If k-set agreement was solvable in a k-resilient system of $n > k + 1$ processes, then the BG simulation of this algorithm with $k + 1$ simulators would produce a wait-free solution to k-set agreement. Since k-set agreement is not wait-free solvable for $k + 1$ processes [13, 19], it follows a contradiction.

The BG simulation presented here only applies to *colorless tasks*. Gafni [8] extended further to more general classes of tasks and provided the general characterization of t-resilient solvable tasks, similarly to the Herlihy-Shavit conditions for wait-free computability [13]. This extension has been also studied in [14, 16].

In order to study the relationship between wait-freedom and t-resilience, [5] uses objects of type $\mathcal{S}$ in addition to read/write registers and shows that for any $t < k$, t-resilient k-process consensus can be implemented with objects of type $\mathcal{S}$ and registers if and only if wait-free $(t + 1)$-process consensus can be implemented with objects of type $\mathcal{S}$ and registers. Imbs and Raynal [15] consider models equipped with registers and consensus objects and extend the results provided by BG simulation, showing equivalences between models based on the ratio between the maximum number of failures and the consensus number of consensus objects.

Chaudhuri and Reiners [7] use BG simulation to provide a characterization of the set consensus partial order, a refinement of Herlihy's consensus-based wait-free hierarchy [10]; a formal definition of *set consensus number* and a study of associated respective computing power have been later provided in [9].

Recommended Reading

1. Afek Y, Attiya H, Dolev D, Gafni E, Merritt M, Shavit N (1993) Atomic snapshots of shared memory. J ACM 40(4):873–890
2. Attiya H (2006) Adapting to point contention with long-lived safe agreement. In: Proceedings of the 13th international conference on structural information and communication complexity, SIROCCO'06, Chester. Springer, Berlin/Heidelberg, pp 10–23
3. Borowsky E, Gafni E (1993) Generalized FLP impossibility result for t-resilient asynchronous computations. In: STOC '93: proceedings of the twenty-fifth annual ACM symposium on theory

of computing, San Diego. ACM, New York, pp 91–100
4. Borowsky E, Gafni E, Lynch N, Rajsbaum S (2001) The BG distributed simulation algorithm. Distrib Comput 14(3):127–146
5. Chandra T, Hadzilacos V, Jayanti P, Toueg S (1994) Wait-freedom vs. t-resiliency and the robustness of wait-free hierarchies (extended abstract). In: PODC '94: proceedings of the thirteenth annual ACM symposium on principles of distributed computing, Los Angeles. ACM, New York, pp 334–343
6. Chaudhuri S (1993) More choices allow more faults: set consensus problems in totally asynchronous systems. Inf Comput 105(1):132–158
7. Chaudhuri S, Reiners P (1996) Understanding the set consensus partial order using the Borowsky-Gafni simulation (extended abstract). In: Proceedings of the 10th international workshop on distributed algorithms, Bologna. Springer, London, pp 362–379
8. Gafni E (2009) The extended BG-simulation and the characterization of t-resiliency. In: Proceedings of the 41st annual ACM symposium on theory of computing, STOC '09, Bethesda. ACM, New York, pp 85–92
9. Gafni E, Kuznetsov P (2011) On set consensus numbers. Distrib Comput 23(3–4):149–163
10. Herlihy M (1991) Wait-free synchronization. ACM Trans Program Lang Syst 13(1):124–149
11. Herlihy M, Kozlov D, Rajsbaum S (2013) Distributed Computing Through Combinatorial Topology. Morgan Kaufmann, Amsterdam
12. Herlihy M, Rajsbaum S (1997) The decidability of distributed decision tasks (extended abstract). In: Proceedings of the twenty-ninth annual ACM symposium on theory of computing, STOC '97, El Paso. ACM, New York, pp 589–598
13. Herlihy M, Shavit N (1999) The topological structure of asynchronous computability. J ACM 46(6):858–923
14. Imbs D, Raynal M (2009) Visiting gafni's reduction land: from the BG simulation to the extended BG simulation. In: SSS, pp 369–383
15. Imbs D, Raynal M (2010) The multiplicative power of consensus numbers. In: Proceedings of the 29th ACM SIGACT-SIGOPS symposium on principles of distributed computing, PODC '10, Zurich. ACM, New York, pp 26–35
16. Kuznetsov P (2013) Universal model simulation: BG and extended BG as examples. In: SSS, pp 17–31
17. Lynch NA (1996) Distributed algorithms. Morgan Kaufmann Publishers Inc., San Francisco
18. Lynch N, Rajsbaum S (1996) On the Borowsky-Gafni simulation algorithm. In: Proceedings of the fourth Israel symposium on theory of computing and systems, ISTCS '96, Jerusalem. IEEE Computer Society, pp 4–15
19. Saks M, Zaharoglou F (2000) Wait-free k-set agreement is impossible: the topology of public knowledge. SIAM J Comput 29(5):1449–1483

Bidimensionality

Fedor V. Fomin[1], Erik D. Demaine[2], Mohammad Taghi Hajiaghayi[3], and Dimitrios Thilikos[4,5]
[1]Department of Informatics, University of Bergen, Bergen, Norway
[2]MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA
[3]Department of Computer Science, University of Maryland, College Park, MD, USA
[4]AlGCo Project-Team, CNRS, LIRMM, France
[5]Department of Mathematics, National and Kapodistrian University of Athens, Athens, Greece

Keywords

EPTAS; Graph minors; Kernelization; Parametrized algorithms; Subexponential algorithms; Treewidth

Years and Authors of Summarized Original Work

2004; Demaine, Fomin, Hajiaghayi; Thilikos
2004; Demaine, Hajiaghayi
2005; Demaine, Fomin, Hajiaghayi, Thilikos
2005; Demaine, Hajiaghayi
2006; Demaine, Hajiaghayi, Thilikos
2008; Demaine, Hajiaghayi
2008; Dorn, Fomin, Thilikos
2009; Fomin, Golovach, Thilikos
2010; Demaine
2010; Fomin, Lokshtanov, Saurabh, Thilikos
2011; Fomin, Lokshtanov, Raman, Saurabh
2011; Fomin, Golovach, Thilikos
2012; Fomin, Lokshtanov, Saurabh
2013; Giannopoulou, Thilikos

2013; Demaine, Fomin, Hajiaghayi, Thilikos 2014; Grigoriev, Koutsonas, Thilikos

Problem Definition

The theory of bidimensionality provides general techniques for designing efficient fixed-parameter algorithms and approximation algorithms for a broad range of NP-hard graph problems in a broad range of graphs. This theory applies to graph problems that are "bidimensional" in the sense that (1) the solution value for the $k \times k$ grid graph and similar graphs grows with k, typically as $\Omega(k^2)$, and (2) the solution value goes down when contracting edges and optionally when deleting edges in the graph. Many problems are bidimensional; a few classic examples are vertex cover, dominating set, and feedback vertex set.

Graph Classes

Results about bidimensional problems have been developed for increasingly general families of graphs, all generalizing planar graphs.

The first two classes of graphs relate to embeddings on surfaces. A graph is *planar* if it can be drawn in the plane (or the sphere) without crossings. A graph has *(Euler) genus* at most g if it can be drawn in a surface of Euler characteristic g. A class of graphs has *bounded genus* if every graph in the class has genus at most g for a fixed g.

The next three classes of graphs relate to excluding minors. Given an edge $e = \{v, w\}$ in a graph G, the *contraction* of e in G is the result of identifying vertices v and w in G and removing all loops and duplicate edges. A graph H obtained by a sequence of such edge contractions starting from G is said to be a *contraction* of G. A graph H is a *minor* of G if H is a subgraph of some contraction of G. A graph class C is *minor closed* if any minor of any graph in C is also a member of C. A minor-closed graph class C is *H-minor-free* if $H \notin C$. More generally, the term "H-minor-free" refers to any minor-closed graph class that excludes some fixed graph H. A *single-crossing graph* is a minor of a graph that can be drawn in the plane with at most one pair of edges crossing. A minor-closed graph class is *single-crossing-minor-free* if it excludes a fixed single-crossing graph. An *apex graph* is a graph in which the removal of some vertex leaves a planar graph. A graph class is *apex-minor-free* if it excludes some fixed apex graph.

Bidimensional Parameters

Although implicitly hinted at in [2, 5, 10, 11], the first use of the term "bidimensional" was in [3].

First, "parameters" are an alternative view on optimization problems. A *parameter* P is a function mapping graphs to nonnegative integers. The *decision problem associated with* P asks, for a given graph G and nonnegative integer k, whether $P(G) \leq k$. Many optimization problems can be phrased as such a decision problem about a graph parameter P.

Now, a parameter is $g(r)$-*bidimensional* (or just *bidimensional*) if it is at least $g(r)$ in an $r \times r$ "grid-like graph" and if the parameter does not increase when taking either minors ($g(r)$-*minor-bidimensional*) or contractions ($g(r)$-*contraction-bidimensional*). The exact definition of "grid-like graph" depends on the class of graphs allowed and whether one considers minor or contraction bidimensionality. For minor bidimensionality and for any H-minor-free graph class, the notion of a "grid-like graph" is defined to be the $r \times rgrid$, i.e., the planar graph with r^2 vertices arranged on a square grid and with edges connecting horizontally and vertically adjacent vertices. For contraction bidimensionality, the notion of a "grid-like graph" is as follows:

1. For planar graphs and single-crossing-minor-free graphs, a "grid-like graph" is an $r \times r$ grid partially triangulated by additional edges that preserve planarity.
2. For bounded-genus graphs, a "grid-like graph" is such a partially triangulated $r \times r$ grid with up to genus (G) additional edges ("handles").
3. For apex-minor-free graphs, a "grid-like graph" is an $r \times r$ grid augmented with additional edges such that each vertex is incident to $O(1)$ edges to nonboundary vertices of the grid. (Here $O(1)$ depends on the excluded apex graph.)

Contraction bidimensionality is so far undefined for H-minor-free graphs (or general graphs).

Examples of bidimensional parameters include the number of vertices, the diameter, and the size of various structures such as feedback vertex set, vertex cover, minimum maximal matching, face cover, a series of vertex-removal parameters, dominating set, edge dominating set, R-dominating set, connected dominating set, connected edge dominating set, connected R-dominating set, unweighted TSP tour (a walk in the graph visiting all vertices), and chordal completion (fill-in). For example, feedback vertex set is $\Omega(r^2)$-minor-bidimensional (and thus also contraction-bidimensional) because (1) deleting or contracting an edge preserves existing feedback vertex sets and (2) any vertex in the feedback vertex set destroys at most four squares in the $r \times r$ grid, and there are $(r-1)^2$ squares, so any feedback vertex set must have $\Omega(r^2)$ vertices. See [1, 3] for arguments of either contraction or minor bidimensionality for the other parameters.

Key Results

Bidimensionality builds on the seminal graph minor theory of Robertson and Seymour, by extending some mathematical results and building new algorithmic tools. The foundation for several results in bidimensionality is the following two combinatorial results. The first relates any bidimensional parameter to treewidth, while the second relates treewidth to grid minors.

Theorem 1 ([1, 8]) *If the parameter P is $g(r)$-bidimensional, then for every graph G in the family associated with the parameter P, $\mathrm{tw}(G) = O(g^{-1}(P(G)))$. In particular, if $g(r) = \Theta(r^2)$, then the bound becomes $\mathrm{tw}(G) = O(\sqrt{P(G)})$.*

Theorem 2 ([8]) *For any fixed graph H, every H-minor-free graph of treewidth w has an $\Omega(w) \times \Omega(w)$ grid as a minor.*

The two major algorithmic results in bidimensionality are general subexponential fixed-parameter algorithm and general polynomial-time approximation scheme (PTASs).

Theorem 3 ([1, 8]) *Consider a $g(r)$-bidimensional parameter P that can be computed on a graph G in $h(w)n^{O(1)}$ time given a tree decomposition of G of width at most w. Then there is an algorithm computing P on any graph G in P's corresponding graph class, with running time $\left[h(O(g^{-1}(k))) + 2^{O(g-1(k))}\right] n^{O(1)}$. In particular, if $g(r) = \Theta(r^2)$ and $h(w) = 2^{O(w2)}$, then this running time is subexponential in k.*

Theorem 4 ([7]) *Consider a bidimensional problem satisfying the "separation property" defined in [4, 7].*

Suppose that the problem can be solved on a graph G with n vertices in $f(n, \mathrm{tw}(G))$ time. Suppose also that the problem can be approximated within a factor of α in $g(n)$ time. For contraction-bidimensional problems, suppose further that both of these algorithms also apply to the "generalized form" of the problem defined in [4, 7]. Then there is a $(1+^2)$-approximation algorithm whose running time is $O(nf(n, O(O^2/^2)) + n^3 g(n))$ for the corresponding graph class of the bidimensional problem.

Applications

The theorems above have many combinatorial and algorithmic applications.

Applying the parameter-treewidth bound of Theorem 1 to the parameter of the number of vertices in the graph proves that every H-minor-free graph on n vertices has treewidth $O(\sqrt{n})$, thus (re)proving the separator theorem for H-minor-free graphs. Applying the parameter-treewidth bound of Theorem 1 to the parameter of the diameter of the graph proves a stronger form of Eppstein's diameter-treewidth relation for apex-minor-free graphs. (Further work shows how to further strengthen the diameter-treewidth relation to linear [6].) The treewidth-grid relation of Theorem 2 can be used to bound the gap between half-integral multicommodity flow and fractional multicommodity flow in H-minor-free graphs. It also yields an $O(1)$-approximation for treewidth in H-minor-

free graphs. The subexponential fixed-parameter algorithms of Theorem 3 subsume or strengthen all previous such results. These results can also be generalized to obtain fixed-parameter algorithms in arbitrary graphs. The PTASs of Theorem 4 in particular establish the first PTASs for connected dominating set and feedback vertex set even for planar graphs. For details of all of these results, see [4].

Open Problems

Several combinatorial and algorithmic open problems remain in the theory of bidimensionality and related concepts.

Can the grid-minor theorem for H-minor-free graphs, Theorem 2, be generalized to arbitrary graphs with a polynomial relation between treewidth and the largest grid minor? (The best relation so far is exponential.) Such polynomial generalizations have been obtained for the cases of "map graphs" and "power graphs" [9]. Good grid-treewidth bounds have applications to minor-bidimensional problems.

Can the algorithmic results (Theorems 3 and 4) be generalized to solve contraction-bidimensional problems beyond apex-minor-free graphs? It is known that the basis for these results, Theorem 1, does not generalize [1]. Nonetheless, Theorem 3 has been generalized for one specific contraction-bidimensional problem, dominating set [3].

Can the polynomial-time approximation schemes of Theorem 4 be generalized to more general algorithmic problems that do not correspond directly to bidimensional parameters? One general family of such problems arises when adding weights to vertices and/or edges, and the goal is, e.g., to find the minimum-weight dominating set. Another family of such problems arises when placing constraints (e.g., on coverage or domination) only on subsets of vertices and/or edges. Examples of such problems include Steiner tree and subset feedback vertex set.

For additional open problems and details about the problems above, see [4].

Cross-References

- Approximation Schemes for Planar Graph Problems
- Branchwidth of Graphs
- Treewidth of Graphs

Recommended Reading

1. Demaine ED, Fomin FV, Hajiaghayi M, Thilikos DM (2004) Bidimensional parameters and local treewidth. SIAM J Discret Math 18(3):501–511
2. Demaine ED, Fomin FV, Hajiaghayi M, Thilikos DM (2005) Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. ACM Trans Algorithms 1(1):33–47
3. Demaine ED, Fomin FV, Hajiaghayi M, Thilikos DM (2005) Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs. J ACM 52(6):866–893
4. Demaine ED, Hajiaghayi M (to appear) The bidimensionality theory and its algorithmic applications. Comput J
5. Demaine ED, Hajiaghayi M (2004) Diameter and treewidth in minor-closed graph families, revisited. Algorithmica 40(3):211–215
6. Demaine ED, Hajiaghayi M (2004) Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In: Proceedings of the 15th ACM-SIAM symposium on discrete algorithms (SODA'04), New Orleans, Jan 2004, pp 833–842
7. Demaine ED, Hajiaghayi M (2005) Bidimensionality: new connections between FPT algorithms and PTASs. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms (SODA 2005), Vancouver, Jan 2005, pp 590–601
8. Demaine ED, Hajiaghayi M (2005) Graphs excluding a fixed minor have grids as large as treewidth, with combinatorial and algorithmic applications through bidimensionality. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms (SODA 2005), Vancouver, Jan 2005, pp 682–689
9. Demaine ED, Hajiaghayi M, Kawarabayashi K (2006) Algorithmic graph minor theory: improved grid minor bounds and Wagner's contraction. In: Proceedings of the 17th annual international symposium on algorithms and computation, Calcutta, Dec 2006. Lecture notes in computer science, vol 4288, pp 3–15
10. Demaine ED, Hajiaghayi M, Nishimura N, Ragde P, Thilikos DM (2004) Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. J Comput Syst Sci 69(2):166–195

11. Demaine ED, Hajiaghayi M, Thilikos DM (2005) Exponential speedup of fixed-parameter algorithms for classes of graphs excluding single-crossing graphs as minors. Algorithmica 41(4):245–267
12. Demaine ED, Hajiaghayi M, Thilikos DM (2006) The bidimensional theory of bounded-genus graphs. SIAM J Discret Math 20(2):357–371

Bin Packing

David S. Johnson
Department of Computer Science, Columbia University, New York, NJ, USA
AT&T Laboratories, Algorithms and Optimization Research Department, Florham Park, NJ, USA

Keywords

Cutting stock problem

Years and Authors of Summarized Original Work

1997; Coffman, Garay, Johnson

Problem Definition

In the one-dimensional bin packing problem, one is given a list $L = (a_1, a_2, \ldots, a_n)$ of items, each item a_i having a size $s(a_i) \in (0, 1]$. The goal is to pack the items into a minimum number of unit-capacity bins, that is, to partition the items into a minimum number of sets, each having total size of at most 1. This problem is NP-hard, and so much of the research on it has concerned the design and analysis of approximation algorithms, which will be the subject of this article.

Although bin packing has many applications, it is perhaps most important for the role it has played as a proving ground for new algorithmic and analytical techniques. Some of the first worst- and average-case results for approximation algorithms were proved in this domain, as well as the first lower bounds on the competitive ratios of online algorithms. Readers interested in a more detailed coverage than is possible here are directed to two relatively recent surveys [4, 11].

Key Results

Worst-Case Behavior

Asymptotic Worst-Case Ratios

For most minimization problems, the standard worst-case metric for an approximation algorithm A is the maximum, over all instances I, of the ratio $A(I)/OPT(I)$, where $A(I)$ is the value of the solution generated by A and $OPT(I)$ is the optimal solution value. In the case of bin packing, however, there are limitations to this "absolute worst-case ratio" metric. Here it is already NP-hard to determine whether $OPT(I) = 2$, and hence no polynomial-time approximation algorithm can have an absolute worst-case ratio better than 1.5 unless P = NP. To better understand the behavior of bin packing algorithms in the typical situation where the given list L requires a large number of bins, researchers thus use a more refined metric for bin packing, the *asymptotic worst-case ratio* R_A^∞. This is defined in two steps as follows.

$$R_A^n = \max\{A(L)/OPT(L) : L \text{ is a list with } OPT(L) = n\}$$

$$R_A^\infty = \limsup_{n\to\infty} R_A^n$$

The first algorithm whose behavior was analyzed in these terms was *First Fit* (FF). This algorithm envisions an infinite sequence of empty bins $B_1, B_2, \ldots$ and, starting with the first item in the input list L, places each item in turn into the first bin which still has room for it. In a technical report from 1971 which was one of the very first papers in which worst-case performance ratios were studied, Ullman [22] proved the following.

Theorem 1 ([22]) $R_{FF}^\infty = 17/10$.

In addition to FF, five other simple heuristics received early study and have served as the in-

spiration for later research. *Best Fit* (BF) is the variant of FF in which each item is placed in the bin into which it will fit with the least space left over, with ties broken in favor of the earliest such bin. Both FF and BF can be implemented to run in time $O(n \log n)$ [12]. *Next Fit* (NF) is a still simpler and linear-time algorithm in which the first item is placed in the first bin, and thereafter each item is placed in the last nonempty bin if it will fit, otherwise a new bin is started. *First Fit Decreasing* (FFD) and *Best Fit Decreasing* (BFD) are the variants of those algorithms in which the input list is first sorted into nonincreasing order by size and then the corresponding packing rule is applied. The results for these algorithms are as follows.

Theorem 2 ([12]) $R_{NF}^{\infty} = 2$.

Theorem 3 ([13]) $R_{BF}^{\infty} = 17/10$.

Theorem 4 ([12, 13]) $R_{FFD}^{\infty} = R_{BFD}^{\infty} = 11/9 = 1.222\ldots$

The above mentioned algorithms are relatively simple and intuitive. If one is willing to consider more complicated algorithms, one can do substantially better. The current best polynomial-time bin packing algorithm is very good indeed. This is the 1982 algorithm of Karmarkar and Karp [15], denoted here as "KK." It exploits the ellipsoid algorithm, approximation algorithms for the knapsack problem, and a clever rounding scheme to obtain the following guarantees.

Theorem 5 ([15]) $R_{KK}^{\infty} = 1$ *and there is a constant c such that for all lists L,*

$$KK(L) \leq OPT(L) + c \log^2(OPT(L)).$$

Unfortunately, the running time for KK appears to be worse than $O(n^8)$, and BFD and FFD remain much more practical alternatives.

Online Algorithms

Three of the abovementioned algorithms (FF, BF, and NF) are *online* algorithms, in that they pack items in the order given, without reference to the sizes or number of later items. As was subsequently observed in many contexts, the online restriction can seriously limit the ability of an algorithm to produce good solutions. Perhaps the first limitation of this type to be proved was Yao's theorem [24] that no online algorithm A for bin packing can have $R_A^{\infty} < 1.5$. The bound has since been improved to the following.

Theorem 6 ([23]) *If A is an online algorithm for bin packing, then* $R_A^{\infty} \geq 1.540\ldots$

Here the exact value of the lower bound is the solution to a complicated linear program.

Yao's paper also presented an online algorithm *Revised First Fit* (RFF) that had $R_{RFF}^{\infty} = 5/3 = 1.666\ldots$ and hence got closer to this lower bound than FF and BF. This algorithm worked by dividing the items into four classes based on size and index, and then using different packing rules (and packings) for each class. Subsequent algorithms improved on this by going to more and more classes. The current champion is the online *Harmonic++* algorithm (H++) of [21]:

Theorem 7 ([21]) $R_{H++}^{\infty} \leq 1.58889$.

Bounded-Space Algorithms

The NF algorithm, in addition to being online, has another property worth noting: no more than a constant number of partially filled bins remain open to receive additional items at any given time. In the case of NF, the constant is 1 – only the last partially filled bin can receive additional items. Bounding the number of open bins may be necessary in some applications, such as packing trucks on loading docks. The bounded-space constraint imposes additional limits on algorithmic behavior however.

Theorem 8 ([17]) *For any online bounded-space algorithm A,* $R_A^{\infty} \geq 1.691\ldots$.

The constant $1.691\ldots$ arises in many other bin packing contexts. It is commonly denoted by h_{∞} and equals $\sum_{i=1}^{\infty}(1/t_i)$, where $t_1 = 1$ and, for $i > 1$, $t_i = t_{i-1}(t_{i-1} + 1)$.

The lower bound in Theorem 8 is tight, owing to the existence of the *Harmonic*$_k$ algorithms (H_k) of [17]. H_k is a class-based algorithm in which the items are divided into classes C_h, $1 \le h \le k$, with C_k consisting of all items with size $1/k$ or smaller, and C_h, $1 \le h < k$, consisting of all a_i with $1/(h+1) < s(a_i) \le 1/h$. The items in each class are then packed by NF into a separate packing devoted just to that class. Thus, at most k bins are open at any time. In [17] it was shown that $\lim_{k\to\infty} R^{\infty}_{H_k} = h_\infty = 1.691\ldots$. This is even better than the asymptotic worst-case ratio of 1.7 for the unbounded-space algorithms FF and BF, although it should be noted that the bounded-space variant of BF in which all but the two most-full bins are closed also has $R^{\infty}_{A} = 1.7$ [8].

Average-Case Behavior

Continuous Distributions

Bin packing also served as an early test bed for studying the average-case behavior of approximation algorithms. Suppose F is a distribution on $(0, 1]$ and L_n is a list of n items with item sizes chosen independently according to F. For any list L, let $s(L)$ denote the lower bound on $OPT(L)$ obtained by summing the sizes of all the items in L. Then define

$$ER^n_A(F) = E\left[A(L_n)/OPT(L_n)\right],$$
$$ER^{\infty}_A(F) = \limsup_{n\to\infty} ER^n_A$$
$$EW^n_A(F) = E\left[A(L_n) - s(L_n)\right]$$

The last definition is included since $ER^{\infty}_A(F) = 1$ occurs frequently enough that finer distinctions are meaningful. For example, in the early 1980s, it was observed that for the distribution $F = U(0, 1]$ in which item sizes are uniformly distributed on the interval $(0, 1]$, $ER^{\infty}_{FFD}(F) = ER^{\infty}_{BFD}(F) = 1$, as a consequence of the following more-detailed results.

Theorem 9 ([16, 20]) *For* $A \in \{FFD, BFD, OPT\}$, $EW^n_A(U(0, 1]) = \Theta(\sqrt{n})$.

Somewhat surprisingly, it was later discovered that the online FF and BF algorithms also had sublinear expected waste, and hence $ER^{\infty}_A(U(0, 1]) = 1$.

Theorem 10 ([5, 19])

$$EW^n_{FF}(U(0, 1]) = \Theta(n^{2/3})$$
$$EW^n_{BF}(U(0, 1]) = \Theta(n^{1/2}\log^{3/4} n)$$

This good behavior does not, however, extend to the bounded-space algorithms NF and H_k:

Theorem 11 ([6, 18])

$$ER^{\infty}_{NF}(U(0, 1]) = 4/3 = 1.333\ldots$$
$$\lim_{k\to\infty} ER_{H_k}(U(0, 1]) = \pi^2/3 - 2 = 1.2899\ldots$$

All the above results except the last two exploit the fact that the distribution $U(0, 1]$ is symmetric about $1/2$, and hence an optimal packing consists primarily of two-item bins, with items of size $s > 1/2$ matched with smaller items of size very close to $1 - s$. The proofs essentially show that the algorithms in question do good jobs of constructing such matchings. In practice, however, there will clearly be situations where more than matching is required. To model such situations, researchers first turned to the distributions $U(0, b]$, $0 < b < 1$, where item sizes are chosen uniformly from the interval $(0, b]$. Simulations suggest that such distributions make things worse for the online algorithms FF and BF, which appear to have $ER^{\infty}_A(U(0, b]) > 1$ for all $b \in (0, 1)$. Surprisingly, they make things better for FFD and BFD (and the optimal packing).

Theorem 12 ([2, 14])

1. *For* $0 < b \le 1/2$ *and* $A \in \{FFD, BFD\}$, $EW^n_A(U(0, b]) = O(1)$.
2. *For* $1/2 < b < 1$ *and* $A \in \{FFD, BFD\}$, $EW^n_A(U(0, b]) = \Theta(n^{1/3})$.
3. *For* $0 < b < 1$, $EW^n_{OPT}(U(0, b]) = O(1)$.

Discrete Distributions

In many applications, the item sizes come from a finite set, rather than a continuous distribution like those discussed above. Thus, recently the study of average-case behavior for bin packing has turned to *discrete distributions*. Such a distribution is specified by a finite list $s_1, s_2, \ldots, s_d$ of rational sizes and for each s_i a corresponding rational probability p_i. A remarkable result of Courcoubetis and Weber [7] says the following.

Theorem 13 ([7]) *For any discrete distribution F, $EW^n_{OPT}(F)$ is either $\Theta(n)$, $\Theta(\sqrt{n})$, or $O(1)$.*

The discrete analogue of the continuous distribution $U(0, b]$ is the distribution $U\{j, k\}$, where the sizes are $1/k, 2/k, \ldots, j/k$ and all the probabilities equal $1/j$. Simulations suggest that the behavior of FF and BF in the discrete case are qualitatively similar to the behavior in the continuous case, whereas the behavior of FFD and BFD is considerably more bizarre [3]. Of particular note is the distribution $F = U\{6, 13\}$, for which $ER^{\infty}_{FFD}(F)$ is strictly greater than $ER^{\infty}_{FF}(F)$, in contrast to all the previously implied comparisons between the two algorithms.

For discrete distributions, however, the standard algorithms are all dominated by a new online algorithm called the *Sum-of-Squares* (SS) algorithm. Note that since the item sizes are all rational, they can be scaled so that they (and the bin size B) are all integral. Then at any given point in the operation of an online algorithm, the current packing can be summarized by giving, for each h, $1 \le h \le B$, the number n_h of bins containing items of total size h. In SS, one packs each item so as to minimize $\sum_{h=1}^{B-1} n_h^2$.

Theorem 14 ([9]) *For any discrete distribution F, the following hold.*

1. *If $EW^n_{OPT}(F) = \Theta(\sqrt{n})$, then $EW^n_{SS}(F) = \Theta(\sqrt{n})$.*
2. *If $EW^n_{OPT}(F) = O(1)$, then $EW^n_{SS}(F) \in \{O(1), \Theta(\log n)\}$.*

In addition, a simple modification to SS can eliminate the $\Theta(\log n)$ case of condition 2.

Applications

There are many potential applications of one-dimensional bin packing, from packing bandwidth requests into fixed-capacity channels to packing commercials into station breaks. In practice, simple heuristics like FFD and BFD are commonly used.

Open Problems

Perhaps the most fundamental open problem related to bin packing is the following. As observed above, there is a polynomial-time algorithm (KK) whose packings are within $O(\log^2(OPT))$ bins of optimal. Is it possible to do better? As far as is currently known, there could still be a polynomial-time algorithm that always gets within one bin of optimal, even if P $\neq$ NP.

Experimental Results

Bin packing has been a fertile ground for experimental analysis, and many of the theorems mentioned above were first conjectured on the basis of experimental results. For example, the experiments reported in [1] inspired Theorem 10 and 12, and the experiments in [10] inspired Theorem 14.

Cross-References

▸ Approximation Schemes for Bin Packing

Recommended Reading

1. Bentley JL, Johnson DS, Leighton FT, McGeoch CC (1983) An experimental study of bin packing. In: Proceedings of the 21st annual Allerton conference on communication, control, and computing, Urbana, University of Illinois, 1983, pp 51–60

2. Bentley JL, Johnson DS, Leighton FT, McGeoch CC, McGeoch LA (1984) Some unexpected expected behavior results for bin packing. In: Proceedings of the 16th annual ACM symposium on theory of computing. ACM, New York, pp 279–288
3. Coffman EG Jr, Courcoubetis C, Garey MR, Johnson DS, McGeoch LA, Shor PW, Weber RR, Yannakakis M (1991) Fundamental discrepancies between average-case analyses under discrete and continuous distributions. In: Proceedings of the 23rd annual ACM symposium on theory of computing, New York, 1991. ACM Press, New York, pp 230–240
4. Coffman EG Jr, Garey MR, Johnson DS (1997) Approximation algorithms for bin-packing: a survey. In: Hochbaum D (ed) Approximation algorithms for NP-hard problems. PWS Publishing, Boston, pp 46–93
5. Coffman EG Jr, Johnson DS, Shor PW, Weber RR (1997) Bin packing with discrete item sizes, part II: tight bounds on first fit. Random Struct Algorithms 10:69–101
6. Coffman EG Jr, So K, Hofri M, Yao AC (1980) A stochastic model of bin-packing. Inf Control 44:105–115
7. Courcoubetis C, Weber RR (1986) Necessary and sufficient conditions for stability of a bin packing system. J Appl Probab 23:989–999
8. Csirik J, Johnson DS (2001) Bounded space on-line bin packing: best is better than first. Algorithmica 31:115–138
9. Csirik J, Johnson DS, Kenyon C, Orlin JB, Shor PW, Weber RR (2006) On the sum-of-squares algorithm for bin packing. J ACM 53:1–65
10. Csirik J, Johnson DS, Kenyon C, Shor PW, Weber RR (1999) A self organizing bin packing heuristic. In: Proceedings of the 1999 workshop on algorithm engineering and experimentation. LNCS, vol 1619. Springer, Berlin, pp 246—265
11. Galambos G, Woeginger GJ (1995) Online bin packing – a restricted survey. ZOR Math Methods Oper Res 42:25–45
12. Johnson DS (1973) Near-optimal bin packing algorithms. PhD thesis, Massachusetts Institute of Technology, Department of Mathematics, Cambridge
13. Johnson DS, Demers A, Ullman JD, Garey MR, Graham RL (1974) Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM J Comput 3:299–325
14. Johnson DS, Leighton FT, Shor PW, Weber RR.: The expected behavior of FFD, BFD, and optimal bin packing under $U(0, \alpha])$ distributions (in preparation)
15. Karmarkar N, Karp RM (1982) An efficient approximation scheme for the one-dimensional bin packing problem. In: Proceedings of the 23rd annual symposium on foundations of computer science. IEEE Computer Society, Los Alamitos, pp 312–320
16. Knödel W (1981) A bin packing algorithm with complexity O(n log n) in the stochastic limit. In: Proceedings of the 10th symposium on mathematical foundations of computer science. LNCS, vol 118. Springer, Berlin, pp 369–378
17. Lee CC, Lee DT (1985) A simple on-line packing algorithm. J ACM 32:562–572
18. Lee CC, Lee DT (1987) Robust on-line bin packing algorithms. Technical report, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston
19. Leighton T, Shor P (1989) Tight bounds for minimax gridmatching with applications to the average case analysis of algorithms. Combinatorica 9:161–187
20. Lueker GS (1982) An average-case analysis of bin packing with uniformly distributed item sizes. Technical report, Report No 181, Department of Information and Computer Science, University of California, Irvine
21. Seiden SS (2002) On the online bin packing problem. J ACM 49:640–671
22. Ullman JD (1971) The performance of a memory allocation algorithm. Tech. Rep. 100, Princeton University, Princeton
23. van Vliet A (1992) An improved lower bound for online bin packing algorithms. Inf Process Lett 43:277–284
24. Yao AC (1980) New algorithms for bin packing. J ACM 27:207–227

Bin Packing with Cardinality Constraints

Hiroshi Fujiwara[1] and Koji M. Kobayashi[2]
[1]Shinshu University, Nagano, Japan
[2]National Institute of Informatics, Chiyoda-ku, Tokyo, Japan

Keywords

Approximation algorithm; Bin packing problem; Cardinality constraint; Competitive analysis; Online algorithm

Years and Authors of Summarized Original Work

1975; Krause, Shen, Schwetman
2003; Babel, Chen, Kellerer, Kotov
2006; Epstein
2010; Epstein, Levin
2013; Fujiwara, Kobayashi
2014; Dósa, Epstein

Problem Definition

In the *bin packing problem*, one is given a sequence of *items*, each of *size* in the range $(0, 1]$, and an infinite number of *bins*. The goal is to pack each item into some bin using as few bins as possible, under the constraint that the sum of sizes of items in each bin is at most one. In the *bin packing problem with cardinality constraints*, an additional constraint is imposed that each bin can contain at most k items.

This problem for $k = 2$ is solvable in polynomial time by reducing it to the *cardinality matching problem*. Nevertheless, this problem for $k \geq 3$ is NP-hard, since one can reduce *3-PARTITION* to it. Therefore, much work has been done on *approximation algorithms*. We remark that, in particular, it has also been of interest to design *online algorithms* that pack each item upon its arrival.

The standard performance measure of an approximation algorithm for this problem is the *asymptotic performance ratio*. For a sequence of items L and an approximation algorithm A, let $A(L)$ denote the value of the solution generated by A for L, and let $OPT(L)$ denote the value of the optimal solution for L. The asymptotic performance ratio of A is defined as

$$R_A^{\infty} = \limsup_{n \to \infty} \sup_{L} \left\{ \frac{A(L)}{OPT(L)} \;\middle|\; OPT(L) = n \right\}.$$

The bin packing problem with cardinality constraints is formally defined as follows:

Problem 1 (Bin Packing with Cardinality Constraints)
Input: *A sequence* $L = (a_1, a_2, \ldots, a_n) \in (0, 1]^n$ *and an integer* $k \geq 2$. Output: *An integer* $m \geq 1$ *and a partition of* $\{1, 2, \ldots, n\}$ *into disjoint subsets* $S_1, S_2, \ldots, S_m$ *such that (1)* m *is minimum, (2)* $\sum_{i \in S_j} a_i \leq 1$ *for all* $1 \leq j \leq m$, *and (3)* $|S_j| \leq k$ *for all* $1 \leq j \leq m$.

Key Results

Approximation Algorithms

Krause et al. [9, 10] gave approximation algorithms whose asymptotic performance ratios are all two. Kellerer and Pferschy [8] presented an improved approximation algorithm with asymptotic performance ratio $\frac{3}{2}$. Caprara et al. [3] provided an APTAS (asymptotic polynomial-time approximation scheme): a collection of approximation algorithms that, for any parameter $\varepsilon > 0$, guarantees an asymptotic performance ratio of $1 + \varepsilon$. Finally, a better polynomial-time scheme was developed:

Theorem 1 ([6]) *There exists an AFPTAS (asymptotic fully polynomial-time approximation scheme) for the bin packing problem with cardinality constraints, that is, an APTAS whose running time is polynomial in the input size and* $\frac{1}{\varepsilon}$.

Online Algorithms

An *online algorithm* is an approximation algorithm which, for each $i = 1, 2, \ldots, n$, decides into which bin to place the ith item without information on the sizes of later items or the value of n. The *First-Fit*, *Best-Fit*, and *Next-Fit* algorithms may be the most common online algorithms for the bin packing problem without cardinality constraints.

Krause et al. [9, 10] adapted the *First-Fit* algorithm to the problem with cardinality constraints and showed that its asymptotic performance ratio is at most $2.7 - \frac{12}{5k}$. The result was later improved. Some work was done for individual values of k. We thus summarize best known upper and lower bounds on the asymptotic performance ratio for each $2 \leq k \leq 6$ in Table 1. We say here that u is an *upper bound* on the asymptotic performance ratio if there exists an online algorithm A such that $R_A^{\infty} = u$. On the other hand, we say that l is a *lower bound* on the asymptotic performance ratio if $R_A^{\infty} \geq l$ holds for any online algorithm A.

Babel et al. [1] designed an online algorithm, denoted here by $BCKK$, which guarantees an

asymptotic performance ratio regardless of the value of k. For $k \geq 7$, $BCKK$ is the best so far.

Theorem 2 ([1]) *For any k, $R^{\infty}_{BCKK} = 2$.*

Recently, Dósa and Epstein [4] showed a lower bound on the asymptotic performance ratio for each $7 \leq k \leq 11$. Fujiwara and Kobayashi [7] established a lower bound for each $12 \leq k \leq 41$. Some results on the bin packing problem without cardinality constraints can be interpreted as lower bounds on the asymptotic performance ratio for large k: a lower bound of $\frac{217}{141}(\approx 1.53900)$ for $42 \leq k \leq 293$ [11] and $\frac{10,633}{6,903}(\approx 1.54034)$ for $294 \leq k \leq 2{,}057$ [2].

Bounded-Space Online Algorithms

A *bounded-space online algorithm* is an online algorithm which has only a constant number of bins available to accept given items at any time point. For example, the *Next-Fit* algorithm is a bounded-space online algorithm for the bin packing problem without cardinality constraints, since for the arrival of each new item, the algorithm always keeps a single bin which contains some item(s). All algorithms that appeared in the previous section, except *Next-Fit*, do not satisfy this property; such algorithms are sometimes called *unbounded-space online algorithms*.

For the bin packing problem with cardinality constraints, a bounded-space online algorithm called CCH_k [5] is known to be optimal, which is based on the Harmonic algorithm. Its asymptotic performance ratio is $\mathcal{R}_k = \sum_{i=1}^{k} \max\left\{\frac{1}{t_i - 1}, \frac{1}{k}\right\}$, where t_i is the sequence defined by $t_1 = 2$, $t_{i+1} = t_i(t_i - 1) + 1$ for $i \geq 1$. For example, we have $\mathcal{R}_2 = \frac{3}{2} = 1.5$, $\mathcal{R}_3 = \frac{11}{6} \approx 1.83333$, $\mathcal{R}_4 = 2$, $\mathcal{R}_5 = 2.1$, and $\mathcal{R}_6 = \frac{13}{6} \approx 2.16667$. The value of $\mathcal{R}_k$ increases as k grows and approaches $1 + \sum_{i=1}^{\infty} \frac{1}{t_i - 1} \approx 2.69103$.

Bin Packing with Cardinality Constraints, Table 1 Best known upper and lower bounds on the asymptotic performance ratio of online algorithms for $2 \leq k \leq 6$

k	Upper bound	Lower bound
2	$1 + \frac{\sqrt{5}}{5}(\approx 1.44721)$ [1]	1.42764 [7]
3	1.75 [5]	1.5 [1]
4	$\frac{71}{38}(\approx 1.86843)$ [5]	1.5 [7]
5	$\frac{771}{398}(\approx 1.93719)$ [5]	1.5 [4]
6	$\frac{287}{144}(\approx 1.99306)$ [5]	1.5 [12]

Theorem 3 ([5]) *For every k, $R^{\infty}_{CCH_k} = \mathcal{R}_k$. Besides, $R^{\infty}_{A} \geq \mathcal{R}_k$ holds for any bounded-space online algorithm A.*

Applications

In the paper by Krause et al. [9, 10], the aim was to analyze task scheduling algorithms for multiprocessor systems. Not only this but a constraint on the number of objects in a container is important in application, such as a limit to the number of files on a hard disk drive or a limit to the number of requests assigned to each node in a distributed system.

Open Problems

Many problems concerning (unbounded-space) online algorithms remain open. Even for small values of k, an optimal online algorithm has yet to be found. It is also interesting whether, for general k, there is an online algorithm whose asymptotic performance ratio is strictly smaller than two.

Cross-References

- Approximation Schemes for Bin Packing
- Bin Packing

Acknowledgments This work was supported by KAKENHI (23700014, 23500014, 26330010, and 26730008).

Recommended Reading

1. Babel L, Chen B, Kellerer H, Kotov V (2004) Algorithms for on-line bin-packing problems with cardinality constraints. Discret Appl Math 143(1-3):238–251

2. Balogh J, Békési J, Galambos G (2012) New lower bounds for certain classes of bin packing algorithms. Theor Comput Sci 440–441:1–13
3. Caprara A, Kellerer H, Pferschy U (2003) Approximation schemes for ordered vector packing problems. Naval Res Logist 50(1):58–69
4. Dósa G, Epstein L (2014) Online bin packing with cardinality constraints revisited. CoRR abs/1404.1056
5. Epstein L (2006) Online bin packing with cardinality constraints. SIAM J Discret Math 20(4):1015–1030
6. Epstein L, Levin A (2010) AFPTAS results for common variants of bin packing: a new method for handling the small items. SIAM J Optim 20(6):3121–3145
7. Fujiwara H, Kobayashi K (2013) Improved lower bounds for the online bin packing problem with cardinality constraints. J Comb Optim 1–21. e10.1007/s10878-013-9679-8
8. Kellerer H, Pferschy U (1999) Cardinality constrained bin-packing problems. Ann Oper Res 92(1):335–348
9. Krause KL, Shen VY, Schwetman HD (1975) Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. J ACM 22(4):522–550
10. Krause KL, Shen VY, Schwetman HD (1977) Errata: "Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems". J ACM 24(3):527
11. van Vliet A (1992) An improved lower bound for on-line bin packing algorithms. Inf Process Lett 43(5):277–284
12. Yao AC (1980) New algorithms for bin packing. J ACM 27(2):207–227

Bin Packing, Variants

Leah Epstein
Department of Mathematics, University of Haifa, Haifa, Israel

Keywords

Approximation schemes; Bin packing; Concave costs

Years and Authors of Summarized Original Work

1994; Anily, Bramel, Simchi-Levi
2012; Epstein, Levin

Problem Definition

The well-known bin packing problem [3, 8] has numerous variants [4]. Here, we consider one natural variant, called the bin packing problem with general cost structures (GCBP) [1, 2, 6]. In this problem, the action of an algorithm remains as in standard bin packing. We are given n items of rational sizes in $(0, 1]$. These items are to be assigned into unit size bins. Each bin may contain items of total size at most 1. While in the standard problem the goal is to minimize the number of used bins, the goal in GCBP is different; the cost of a bin is not 1, but it depends on the number of items actually packed into this bin. This last function is a concave function of the number of packed items, where the cost of an empty bin is zero. More precisely, the input consists of n items $I = \{1, 2, \ldots, n\}$ with sizes $1 \geq s_1 \geq s_2 \geq \cdots \geq s_n \geq 0$ and a function $f : \{0, 1, 2, \ldots, n\} \to \mathbb{R}_0^+$, where f is a monotonically nondecreasing concave function, satisfying $f(0) = 0$. The goal is to partition I into some number of sets $S_1, \ldots, S_\mu$, called bins, such that $\sum_{j \in S_i} s_j \leq 1$ for any $1 \leq i \leq \mu$, and so that $\sum_{i=1}^{\mu} f(|S_i|)$ is minimized (where $|S_i|$ denotes the cardinality of the set S_i). An instance of GCBP is defined not only by its input item sizes but also using the function f. It can be assumed that $f(1) = 1$ (by possible scaling of the cost function f). The problem is strongly NP-hard for multiple functions f, and as standard bin packing, it was studied using the asymptotic approximation ratio.

Key Results

There are two kinds of results for the problem. The first kind of results is algorithms that do not take f into account. The second kind is those that base their action on the values of f.

A class of (concave and monotonically nondecreasing) functions $\{f_q\}_{q \in \mathbb{N}}$, which was considered in [1], is the following. These are functions that grow linearly (with a slope of 1) up to an integer point q, and then, they are constant

(starting from that point). Specifically, $f_q(t) = t$ for $t \leq q$ and $f_q(t) = q$ for $t > q$. It was shown in [1] that focusing on such functions is sufficient when computing upper bounds on algorithms that act independently of the cost function. Note that $f_1 \equiv 1$, and thus GCBP with the cost function f_1 is equivalent to standard bin packing.

Before describing the results, we present a simple example showing the crucial differences between GCBP and standard bin packing. Consider the function $f = f_3$ (where $f(1) = 1$, $f(2) = 2$, and $f(k) = 3$ for $k \geq 3$). Given an integer $N \geq 1$, consider an input consisting of $3N$ items, each of size $\frac{2}{3}$, called large items, and $6N$ items, each of size $\frac{1}{6}$, called small items. An optimal solution for this input with respect to standard bin packing uses $3N$ bins, each containing one large item and two small items. This is the unique optimal solution (up to swapping positions of identical items). The cost of this solution for GCBP with the function $f = f_3$ is $9N$. Consider a solution that uses $4N$ bins, the first N bins receive six small items each, and each additional bin receives one large item. This solution is not optimal for standard bin packing, but its cost for GCBP with $f = f_3$ is $6N$.

Anily, Bramel, and Simchi-Levi [1] analyzed the worst-case performance of some natural bin packing heuristics [8], when they are applied to GCBP. They showed that many common heuristics for bin packing, such as First Fit, Best Fit, and Next Fit, do not have a finite asymptotic approximation ratio. Moreover, running the modifications of the first two heuristics after sorting the lists of items (in a nonincreasing order), i.e., applying the algorithms First Fit Decreasing and Best Fit Decreasing, leads to similar results. However, Next Fit Decreasing was shown to have an asymptotic approximation ratio of exactly 2. The algorithm Next Fit packs items into its last bin as long as this is possible and opens a new bin when necessary. Sorting the items in nondecreasing order gives a better asymptotic approximation ratio of approximately 1.691 (in this case, the three algorithms, First Fit Increasing, Best Fit Increasing, and Next Fit Increasing, are the same algorithm). It is stated in [1] that any heuristic that is independent of f has an asymptotic approximation ratio of at least $\frac{4}{3}$. An improved approximation algorithm, called MatchHalf (MH), was developed in [6]. The asymptotic approximation ratio of this algorithm does not exceed 1.5. The idea of MH is to create bins containing pairs of items. The candidate items to be packed into those bins are half of the items of size above $\frac{1}{2}$ (large items), but they can only be packed with smaller items. Naturally, the smallest large items are selected, and the algorithm tries to match them with smaller items. The remaining items and unmatched items are packed using Next Fit Increasing. Interestingly, it was shown [6] that matching a larger fraction of large items can harm the asymptotic approximation ratio.

A fully polynomial approximation scheme (asymptotic FPTAS or AFPTAS) for GCBP was given in [6]. This is a family of approximation algorithms that contains, for any $\varepsilon > 0$, an approximation algorithm whose asymptotic approximation ratio is at most $1 + \varepsilon$. The running time must be polynomial in the input and in $\frac{1}{\varepsilon}$. An AFPTAS for GCBP must use the function f in its calculations (this can be shown using the example above and similar examples and can also be deduced from the lower bound of $\frac{4}{3}$ on the asymptotic approximation ratio of an algorithm that is oblivious of f [1]). An AFPTAS for GCBP is presented in [6]. One difficulty in designing such a scheme is that the nature of packing of small items is important, unlike approximation schemes for standard bin packing, where small items can be added greedily [7, 9]. While in our problem we can impose cardinality constraints on bins (upper bounds on numbers of packed items) as in [5], still the cost function introduces major difficulties. Another ingredient of the scheme is preprocessing where some very small items are packed into relatively full bins. It is impossible to do this for all very small items as bins consisting of only such items will have a relatively large cost (as each such bin will contain a very large number of items). This AFPTAS and those of [5] require column generation as in [9] but require fairly complicated configuration linear programs.

Cross-References

▸ Harmonic Algorithm for Online Bin Packing

Recommended Reading

1. Anily S, Bramel J, Simchi-Levi D (1994) Worst-case analysis of heuristics for the bin packing problem with general cost structures. Oper Res 42(2):287–298
2. Bramel J, Rhee WT, Simchi-Levi D (1998) Average-case analysis of the bin packing problem with general cost structures. Naval Res Logist 44(7):673–686
3. Coffman E Jr, Csirik J (2007) Performance guarantees for one-dimensional bin packing. In: Gonzalez TF (ed) Handbook of approximation algorithms and metaheuristics, chap 32. Chapman & Hall/CRC, Boca Raton, pp (32–1)–(32–18)
4. Coffman E Jr, Csirik J (2007) Variants of classical one-dimensional bin packing. In: Gonzalez TF (ed) Handbook of approximation algorithms and metaheuristics, chap 33. Chapman & Hall/CRC, Boca Raton, pp (33–1)–(33–14)
5. Epstein L, Levin A (2010) AFPTAS results for common variants of bin packing: a new method for handling the small items. SIAM J Optim 20(6):3121–3145
6. Epstein L, Levin A (2012) Bin packing with general cost structures. Math Program 132(1–2): 355–391
7. Fernandez de la Vega W, Lueker GS (1981) Bin packing can be solved within $1 + \varepsilon$ in linear time. Combinatorica 1(4):349–355
8. Johnson DS, Demers AJ, Ullman JD, Garey MR, Graham RL (1974) Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM J Comput 3(4):299–325
9. Karmarkar N, Karp RM (1982) An efficient approximation scheme for the one-dimensional bin packing problem. In: Proceedings of the 23rd annual symposium on foundations of computer science (FOCS1982), Chicago, Illinois, USA, pp 312–320

Binary Decision Graph

Adnan Aziz[1] and Amit Prakash[2]
[1]Department of Electrical and Computer Engineering, University of Texas, Austin, TX, USA
[2]Microsoft, MSN, Redmond, WA, USA

Keywords

BDDs; Binary decision diagrams

Years and Authors of Summarized Original Work

1986; Bryant

Problem Definition

Boolean Functions

The concept of a *Boolean function* – a function whose domain is $\{0,1\}^n$ and range is $\{0,1\}$ – is central to computing. Boolean functions are used in foundational studies of complexity [7, 9] as well as the design and analysis of logic circuits [4, 13], A Boolean function can be represented using a *truth table* – an enumeration of the values taken by the function on each element of $\{0,1\}^n$. Since the truth table representation requires memory exponential in n, it is impractical for most applications. Consequently, there is a need for data structures and associated algorithms for efficiently representing and manipulating Boolean functions.

Boolean Circuits

Boolean functions can be represented in many ways. One natural representation is a *Boolean combinational circuit*, or circuit for short [6, Chapter 34]. A circuit consists of *Boolean combinational elements* connected by *wires*. The Boolean combinational elements are *gates* and *primary inputs*. Gates come in three types: NOT, AND, and OR. The NOT gate functions as follows: it takes a single Boolean-valued *input* and produces a single Boolean-valued *output* which takes value 0 if the input is 1, and 1 if the input is 0. The AND gate takes two Boolean-valued inputs and produces a single output; the output is 1 if both inputs are 1, and 0 otherwise. The OR gate is similar to AND, except that its output is 1 if one or both inputs are 1, and 0 otherwise.

Circuits are required to be acyclic. The absence of cycles implies that a Boolean assignment to the primary inputs can be unambiguously propagated through the gates in topological order. It follows that a circuit on n ordered primary inputs with a designated gate called the *primary output*

corresponds to a Boolean function on $\{0,1\}^n$. Every Boolean function can be represented by a circuit, e.g., by building a circuit that mimics the truth table.

The circuit representation is very general – any decision problem that is computable in polynomial time on a Turing machine can be computed by circuit polynomial in the instance size, and the circuits can be constructed efficiently from the Turing machine program [15]. However, the key analysis problems on circuits, namely, satisfiability and equivalence, are NP-hard [7].

Boolean Formulas

A *Boolean formula* is defined recursively: a *Boolean variable* x_i is a *Boolean formula*, and if φ and ψ are Boolean formulas, then so are $(\neg\phi)$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, and $(\phi \leftrightarrow \psi)$. The operators $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ are referred to as *connectives*; parentheses are often dropped for notational convenience. Boolean formulas also can be used to represent arbitrary Boolean functions; however, formula satisfiability and equivalence are also NP-hard. Boolean formulas are not as succinct as Boolean circuits: for example, the parity function has linear sized circuits, but formula representations of parity are super-polynomial. More precisely, $XOR_n : \{0,1\}^n \rightarrow \{0,1\}$ is defined to take the value 1 on exactly those elements of $\{0,1\}^n$ which contain an odd number of 1s. Define the *size* of a formula to be the number of connectives appearing in it. Then for any sequence of formulas $\theta_1, \theta_2, \ldots$ such that θ_k represents XOR_k, the size of θ_k is $\Omega(k^c)$ for all $c \in Z^+$ [14, Chapters 11, 12].

A *disjunct* is a Boolean formula in which $\wedge$ and $\neg$ are the only connectives, and $\neg$ is applied only to variables; for example, $x_1 \wedge \neg x_3 \wedge \neg x_5$ is a disjunct. A Boolean formula is said to be in *Disjunctive Normal Form* (DNF) if it is of the form $D_0 \vee D_1 \vee \cdots \vee D_{k-1}$, where each D_i is a disjunct. DNF formulas can represent arbitrary Boolean functions, e.g., by identifying each input on which the formula takes the value 1 with a disjunct. DNF formulas are useful in logic design, because it can be translated directly into a PLA implementation [4]. While satisfiability of DNF formulas is trivial, equivalence is NP-hard. In addition, given DNF formulas φ and ψ, the formulas $\neg\phi$ and $\phi \wedge \psi$ are not DNF formulas, and the translation of these formulas to DNF formulas representing the same function can lead to exponential growth in the size of the formula.

Shannon Trees

Let f be a Boolean function on domain $\{0,1\}^n$. Associate the n dimensions with variables $x_0, \ldots, x_{n-1}$. Then the *positive cofactor* of f with respect to x_i, denoted by f_{x_i}, is the function on domain $\{0,1\}^n$, which is defined by

$$f_{x_i}(\alpha_0, \ldots, \alpha_{i-1}, a_i, \alpha_{i+1}, \ldots, \alpha_{n-1}) = f(\alpha_0, \ldots, \alpha_{i-1}, 1, \alpha_{i+1}, \ldots, \alpha_{n-1}).$$

The *negative cofactor* of f with respect to x_i, denoted by $f_{x_i'}$, is defined similarly, with 0 taking the place of 1 in the right-hand side.

Every Boolean function can be decomposed using Shannon's expansion theorem:

$$f(x_1, \ldots, x_n) = x_i \cdot f_{x_i} + x_i' \cdot f_{x_i'}.$$

This observation can be used to represent f by a *Shannon tree* – a kill binary tree [6, Appendix B.5] of height n, where each path to a leaf node defines a complete assignment to the n variables that f is defined over, and the leaf node holds a 0 or a 1, based on the value f takes for the assignment.

The Shannon tree is not a particularly useful representation, since the height of the tree representing every Boolean function on $\{0,1\}^n$ is n, and the tree has 2^n leaves. The Shannon tree can be made smaller by merging isomorphic subtrees and bypassing nodes which have identical children. At first glance the reduced Shannon tree representation is not particularly useful, since it entails creating the full binary tree in the first place. Furthermore, it is not clear how to efficiently perform computations on the reduced Shannon tree representation, such as equivalence checking or computing the conjunction of functions presented as reduced Shannon trees.

Bryant [5] recognized that adding the restriction that variables appear in fixed order from root to leaves greatly reduced the complexity of manipulating reduced Shannon trees. He referred to this representation as a binary decision diagram (BDD).

Key Results

Definitions

Technically, a BDD is a directed acyclic graph (DAG), with a designated root, and at most two sinks – one labeled 0, the other labeled 1. Nonsink nodes are labeled with a variable. Each nonsink node has two outgoing edges – one labeled with a 1 leading to the *1-child*, the other is a 0, leading to the *0-child*. Variables must be ordered – that is, if the variable label x_i appears before the label x_j on some path from the root to a sink, then the label x_j is precluded from appearing before x_i on any path from the root to a sink. Two nodes are *isomorphic* if both are equilabeled sinks, or they are both nonsink nodes, with the same variable label, and their 0- and 1-children are isomorphic. For the DAG to be a valid BDD, it is required that there are no isomorphic nodes, and for no nodes are its 0- and 1-children the same.

A key result in the theory of BDDs is that given a fixed variable ordering, the representation is unique up to isomorphism, i.e., if F and G are both BDDs representing $f : \{0,1\}^n \to \{0,1\}$ under the variable ordering $x_1 \prec x_2 \prec \ldots x_n$, then F and G are isomorphic.

The definition of isomorphism directly yields a recursive algorithm for checking isomorphism. However, the resulting complexity is exponential in the number of nodes – this is illustrated, for example, by checking the isomorphism of the BDD for the parity function against itself on inspection, the exponential complexity arises from repeated checking of isomorphism between pairs of nodes – this naturally suggest dynamic programming. Caching isomorphism checks reduces the complexity of isomorphism checking to $O(|F| \cdot |G|)$, where $|B|$ denotes the number of nodes in the BDD B.

BDD Operations

Many logical operations can be implemented in polynomial time using BDDs: *bdd_and* which computes a BDD representing the logical AND of the functions represented by two BDDs, *bdd_or* and *bdd_not* which are defined similarly, and *bdd_compose* which takes a BDD representing a function f, a variable v, and a BDD representing a function g and returns the BDD for f where v is substituted by g are examples.

The example of *bdd_and* is instructive – it is based on the identity $f \cdot g = x \cdot (f_x \cdot g_x) + x' \cdot (f_{x'} \cdot g_{x'})$. The recursion can be implemented directly: the base cases are when either f or g are 0 and when one or both are 1. The recursion chooses the variable v labeling either the root of the BDD for f or g, depending on which is earlier in the variable ordering, and recursively computes BDDs for $f_v \cdot g_v$ and $f_{v'} \cdot g_{v'}$; these are merged if isomorphic. Given a BDD F for f, if v is the variable labeling the root of F, the BDDs for $f_{v'}$ and f_v, respectively, are simply the 0-child and 1-child of F's root.

The implementation of *bdd_and* as described has exponential complexity because of repeated subproblems arising. Dynamic programming again provides a solution – caching the intermediate results of *bdd_and* reduced the complexity to $O(|F| \cdot |G|)$.

Variable Ordering

All symmetric functions on $\{0,1\}^n$ have a BDD that is polynomial in n, independent of the variable ordering. Other useful functions such as comparators, multiplexers, adders, and subtracters can also be efficiently represented, if the variable ordering is selected correctly. Heuristics for ordering selection are presented in [1, 2, 11]. There are functions which do not have a polynomial-sized BDD under any variable ordering – the Unction representing the n-th bit of the output of a multiplier taking two n-bit unsigned integer inputs is an example [5]. Wegener [17] presents many more examples of the impact of variable ordering.

Applications

BDDs have been most commonly applied in the context of formal verification of digital hardware [8]. Digital hardware extends the notion of circuit described above by adding *state elements* which hold a Boolean value between updates and are updated on a *clock* signal.

The gates comprising a design are often updated based on performance requirements; these changes typically are not supposed to change the logical functionality of the design. BDD-based approaches have been used for checking the equivalence of digital hardware designs [10].

BDDs have also been used for checking properties of digital hardware. A typical formulation is that a set of "good" states and a set of "initial" states are specified using Boolean formulas over the state elements; the property holds iff there is no sequence of inputs which leads a state in the initial state to a state not in the set of good states. Given a design with n registers, a set of states A in the design can be characterized by a formula φ_A over n Boolean variables: φ_A evaluates to true on an assignment to the variables iff the corresponding state is in A. The formula φ_A represents a Boolean function, and so BDDs can be used to represent sets of states. The key operation of computing the *image* of a set of states A, i.e., the set of states that can be reached on application of a single input from states in A, can also be implemented using BDDs [12].

BDDs have been used for *test generation*. One approach to test generation is to specify legal inputs using constraints, in essence Boolean formulas over the primary input and state variables. Yuan et al. [18] have demonstrated that BDDs can be used to solve these constraints very efficiently.

Logic synthesis is the discipline of realizing hardware designs specified as logic equations using gates. Mapping equations to gates is straightforward; however, in practice a direct mapping leads to implementations that are not acceptable from a performance perspective, where performance is measured by gate area or timing delay. Manipulating logic equations in order to reduce area (e.g., through constant propagation, identifying common sub-expressions, etc.), and delay (e.g., through propagating late arriving signals closer to the outputs), is conveniently done using BDDs.

Experimental Results

Bryant reported results on verifying two qualitatively distinct circuits for addition. He was able to verify on a VAX 11/780 (a 1 MIP machine) that two 64-bit adders were equivalent in 95.8 min. He used an ordering that he derived manually.

Normalizing for technology, modern BDD packages are two orders of magnitude faster than Bryant's original implementation. A large source the improvement comes from the use of the *strong canonical form*, wherein a global database of BDD nodes is maintained, and no new node is added without checking to see if a node with the same label and 0- and 1-children exists in the database [3]. (For this approach to work, it is also required that the children of any node being added be in strong canonical form.) Other improvements stem from the use of complement pointers (if a pointer has its least-significant bit set, it refers to the complement of the function), better memory management (garbage collection based on reference counts, keeping nodes that are commonly accessed together close in memory), better hash functions, and better organization of the computed table (which keeps track of subproblems that have already been encountered) [16].

Data Sets

The SIS (http://embedded.eecs.berkeley.edu/pubs/downloads/sis/) system from UC Berkeley is used for logic synthesis. It comes with a number of combinational and sequential circuits that have been used for benchmarking BDD packages.

The VIS (http://embedded.eecs.berkeley.edu/pubs/downloads/vis) system from UC Berkeley and UC Boulder is used for design verification; it uses BDDs to perform checks. The distribution includes a large collection of verification problems, ranging from simple hardware circuits to complex multiprocessor cache systems.

URL to Code

A number of BDD packages exist today, but the package of choice is CUDD (http://vlsi.colorado.edu/~fabio/CUDD/). CUDD implements all the core features for manipulating BDDs, as well as variants. It is written in C++ and has extensive user and programmer documentation.

Cross-References

▸ Symbolic Model Checking

Recommended Reading

1. Aziz A, Tasiran S, Brayton R (1994) BDD variable ordering for interacting finite state machines. In: ACM design automation conference, San Diego, pp 283–288
2. Berman CL (1989) Ordered binary decision diagrams and circuit structure. In: IEEE international conference on computer design, Cambridge
3. Brace K, Rudell R, Bryant R (1990) Efficient implementation of a BDD package. In: ACM design automation conference, Orlando
4. Brayton R, Hachtel G, McMullen C, Sangiovanni-Vincentelli A (1984) Logic minimization algorithms for VLSI synthesis. Kluwer Academic, Boston
5. Bryant R (1986) Graph-based algorithms for Boolean function manipulation. IEEE Trans Comput C-35:677–691
6. Cormen TH, Leiserson CE, Rivest RH, Stein C (2001) Introduction to algorithms. MIT, Cambridge
7. Garey MR, Johnson DS (1979) Computers and intractability. W.H. Freeman and Co, New York
8. Gupta A (1993) Formal hardware verification methods: a survey. Formal Method Syst Des 1:151–238
9. Karchmer M (1989) Communication complexity: a new approach to circuit depth. MIT, Cambridge
10. Kuehlmann A, Krohm F (1997) Equivalence checking using cuts and heaps. In: ACM design automation conference, Anaheim
11. Malik S, Wang AR, Brayton RK, Sangiovanni-Vincentelli A (1988) Logic verification using binary decision diagrams in a logic synthesis environment. In: IEEE international conference on computer-aided design, Santa Clara, pp 6–9
12. McMillan KL (1993) Symbolic model checking. Kluwer Academic, Boston
13. De Micheli G (1994) Synthesis and optimization of digital circuits. McGraw Hill, New York
14. Schoning U, Pruim R (1998) Gems of theoretical computer science. Springer, Berlin/New York
15. Sipser M (2005) Introduction to the theory of computation, 2nd edn. Course Technology, Boston
16. Somenzi F (2012) Colorado University Decision Diagram package. http://vlsi.colorado.edu/~fabio/CUDD
17. Wegener I (2000) Branching programs and binary decision diagrams. SIAM, Philadelphia
18. Yuan J, Pixley C, Aziz A (2006) Constraint-based verification. Springer, New York

Binary Space Partitions

Adrian Dumitrescu[1] and Csaba D. Tóth[2,3]
[1]Computer Science, University of Wisconsin–Milwaukee, Milwaukee, WI, USA
[2]Department of Computer Science, Tufts University, Medford, MA, USA
[3]Department of Mathematics, California State University Northridge, Los Angeles, CA, USA

Keywords

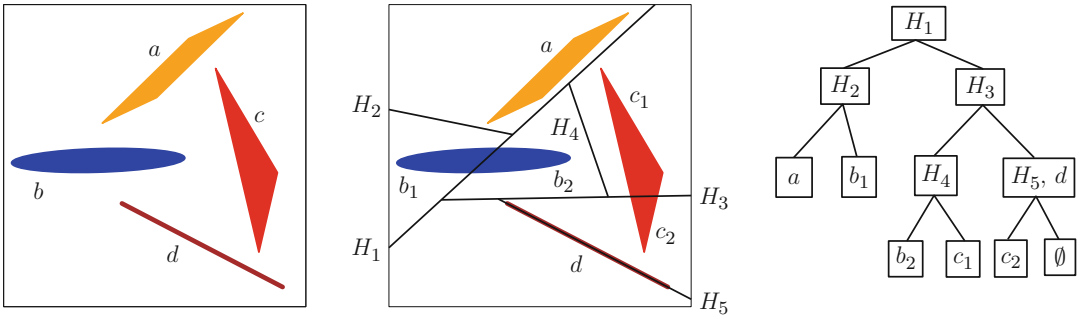
BSP tree; Computational geometry; Convex decomposition; Recursive partition

Years and Authors of Summarized Original Work

1990; Paterson, Yao
1992; D'Amore, Franciosa
1992; Paterson, Yao
2002; Berman, DasGupta, Muthukrishnan
2003; Tóth
2004; Dumitrescu, Mitchell, Sharir
2005; Hershberger, Suri, Tóth
2011; Tóth

Problem Definition

The *binary space partition* (for short, *BSP*) is a scheme for subdividing the ambient space $\mathbb{R}^d$



Binary Space Partitions, Fig. 1 Three 2-dimensional convex objects and a line segment (*left*), a binary space partition with five partition lines $H_1, \ldots, H_5$ (*center*), and the corresponding BSP tree (*right*)

into open convex sets (called *cells*) by hyperplanes in a recursive fashion. Each subdivision step for a cell results in two cells, in which the process may continue, independently of other cells, until a stopping criterion is met. The binary recursion tree, also called *BSP-tree*, is traditionally used as a data structure in computer graphics for efficient rendering of polyhedral scenes. Each node v of the BSP-tree, except for the leaves, corresponds to a cell $C_v \subseteq \mathbb{R}^d$ and a partitioning hyperplane H_v. The cell of the root r is $C_r = \mathbb{R}^d$, and the two children of a node v correspond to $C_v \cap H_v^-$ and $C_v \cap H_v^+$, where H_v^- and H_v^+ denote the open half-spaces bounded by H_v. Refer to Fig. 1.

A binary space partition *for* a set of n pairwise disjoint (typically polyhedral) objects in $\mathbb{R}^d$ is a BSP where the space is recursively partitioned until each cell intersects at most one object. When the BSP-tree is used as a data structure, every leaf v stores the fragment of at most one object clipped in the cell C_v, and every interior node v stores the fragments of any lower-dimensional objects that lie in $C_v \cap H_v$.

A BSP for a set of objects has two parameters of interest: the *size* and the *height* of the corresponding BSP-tree. Ideally, a BSP partitions space so that each object lies entirely in a single cell or in a cutting hyperplane, yielding a so-called *perfect* BSP [4]. However, in most cases this is impossible, and the hyperplanes H_v partition some of the input objects into *fragments*. Assuming that the input objects are k-dimensional, for some $k \leq d$, the BSP typically stores only k-dimensional fragments, i.e., object parts clipped in leaf cells C_v or in $C_v \cap H_v$ at interior nodes.

The size of the BSP-tree is typically proportional to the number of k-dimensional fragments that the input objects are partitioned into, or the number of nodes in the tree. Given a set S of objects in $\mathbb{R}^d$, one would like to find a BSP for S with small size and/or height. The *partition complexity* of a set of objects S is defined as the minimum size of a BSP for S.

Glossary

- **Autopartition**: a class of BSPs obtained by imposing the constraint that each cut is along a hyperplane containing a facet of one of the input objects.
- **Axis-aligned BSP**: a class of BSPs obtained by imposing the constraint that each cut is orthogonal to a coordinate axis.
- **Round-robin BSP:** An axis-aligned BSP in $\mathbb{R}^d$ where any d consecutive recursive cuts are along hyperplanes orthogonal to the d coordinate axes.
- **Tiling** in $\mathbb{R}^d$: a set of interior-disjoint polyhedra that partition $\mathbb{R}^d$.
- **Axis-aligned tiling**: a set of full-dimensional boxes that partition $\mathbb{R}^d$.
- **d-dimensional box**: the cross product of d real-valued intervals.

Key Results

The theoretical study of BSPs was initiated by Paterson and Yao [10, 11].

Line Segments in the Plane

A classical result of Paterson and Yao [10] is a simple and elegant randomized algorithm, which, given n disjoint segments, produces a BSP whose expected size is $O(n \log n)$; see also [3, Ch. 12]. It was widely believed for decades that every set of n disjoint line segments in the plane admits a BSP of size $O(n)$; see e.g., [10, p. 502]; this was until Tóth proved a tight super-linear bound for this problem, first by constructing a set of segments for which any BSP must have size $\Omega(n \log n / \log \log n)$ and later by matching this bound algorithmically:

Theorem 1 ([12, 15]) *Every set of n disjoint line segments in the plane admits a BSP of size $O(n \log n / \log \log n)$. This bound is the best possible, and a BSP of this size can be computed in $O(n \log^2 n)$ time.*

Simplices in $\mathbb{R}^d$

The randomized partition technique of Paterson and Yao generalizes to higher dimensions yielding the following.

Theorem 2 ([10]) *Every set of n $(d-1)$-dimensional simplices in $\mathbb{R}^d$, where $d \geq 3$ admits a BSP of size $O(n^{d-1})$.*

While there exist n disjoint triangles in $\mathbb{R}^3$ that require a BSP of size $\Omega(n^2)$, no super-quadratic lower bound is known in any dimension d. Near-linear upper bounds are known for "realistic" input models in $\mathbb{R}^3$ such as uncluttered scenes [5] or fat axis-aligned rectangles [14].

Axis-Parallel Segments, Rectangles, and Hyperrectangles

Theorem 3 ([1, 7, 10]) *Every set of n pairwise disjoint axis-parallel line segments in the plane admits an auto-partition of size at most $2n-1$. Such a BSP can be computed using $O(n \log n)$ time and space and has the additional property that no input segment is cut more than once. The upper bound on the size is the best possible apart from lower-order terms.*

Theorem 4 ([7, 11]) *Let Γ be a collection of n line segments in $\mathbb{R}^d$, where $d \geq 3$, consisting of n_i segments parallel to the x_i-axis, for $i = 1, \ldots, d$. Then Γ admits a BSP of size at most*

$$4^{1/(d-1)}(d-1)(n_1 n_2 \ldots n_d)^{1/(d-1)} + 2n.$$

Theorem 5 ([7]) *For constants $1 \leq k \leq d-1$, every set of n axis-parallel k-rectangles in d-space admits an axis-aligned BSP of size $O(n^{d/(d-k)})$. This bound is the best possible for $k < d/2$ apart from the constant factor.*

For $k \geq d/2$, the best known upper and lower bounds do not match. No super-quadratic lower bound is known in any dimension d. In $\mathbb{R}^4$, Dumitrescu et al. [7] constructed n 2-dimensional disjoint rectangles whose partition complexity is $\Omega(n^{5/3})$.

Tilings

Already in the plane, the worst-case partition complexity of axis-aligned tilings is smaller than that for disjoint boxes. Berman, DasGupta, and Muthukrishnan [6] showed that every axis-aligned tiling of size n admits an axis-aligned BSP of size at most $2n$; apart from lower-order terms, this bound is the best possible. For higher dimensions, Hershberger, Suri, and Tóth obtained the following result.

Theorem 6 ([9]) *Every axis-aligned tiling of size n in $\mathbb{R}^d$, where $d \geq 2$, admits a round-robin BSP of size $O(n^{(d+1)/3})$. On the other hand, there exist tilings of size n in $\mathbb{R}^d$ for which every BSP has size $\Omega(n^{\beta(d)})$, where $\beta(3) = 4/3$, and $\lim_{d\to\infty} \beta(d) = (1+\sqrt{5})/2 \approx 1.618$.*

In dimensions $d = 3$, the partition complexity of axis-aligned tilings of size n is $O(n^{4/3})$, which is tight by a construction of Hershberger and Suri [8].

Applications

The initial and most prominent applications are in computer graphics: BSPs support fast *hidden-surface removal* and *ray tracing* for moving viewpoints [10]. Rendering is used for visualizing spatial opaque surfaces on the screen. A common and efficient rendering technique is the so-called *painter's algorithm*. Every object is drawn sequentially according to the *back-to-front* order, starting with the deepest object and continuing with the objects closer to the viewpoint. When

all the objects have been drawn, every pixel represents the color of the object closest to the viewpoint. Further computer graphics applications include *constructive solid geometry* and *shadow generation*. Other applications of BSP trees include *range counting*, *point location*, *collision detection*, *robotics*, *graph drawing*, and *network design*; see, for instance, [13] and the references therein.

In the original setting, the input objects of the BSP were assumed to be static. Recent research on BSPs for moving objects can be seen in the context of *kinetic data structures* (KDS) of Basch, Guibas, and Hershberger [2]. In this model, objects move continuously along a given trajectory (flight plan), typically along a line or a low-degree algebraic curve. The splitting hyperplanes are defined by faces of the input objects, and so they move continuously, too. The BSP is updated only at discrete *events*, though, when the combinatorial structure of the BSP changes.

Open Problems

- What is the maximum partition complexity of n disjoint $(d-1)$-dimensional simplices in $\mathbb{R}^d$ for $d \geq 3$?
- What is the maximum partition complexity of n disjoint (axis-aligned) boxes in $\mathbb{R}^d$ for $d \geq 3$?
- What is the maximum (axis-aligned) partition complexity of a tiling of n axis-aligned boxes in $\mathbb{R}^d$ for $d \geq 4$?
- Are there families of n disjoint objects in $\mathbb{R}^d$ whose partition complexity is super-quadratic in n?
- How many combinatorial changes can occur in the kinetic BSP of n points moving with constant velocities in the plane?

In all five open problems, the dimension $d \in \mathbb{N}$ of the ambient space $\mathbb{R}^d$ is constant, and asymptotically tight bounds in terms of n are sought.

Recommended Reading

1. d'Amore F, Franciosa PG (1992) On the optimal binary plane partition for sets of isothetic rectangles. Inf Process Lett 44:255–259
2. Basch J, Guibas LJ, Hershberger J (1999) Data structures for mobile data. J Algorithms 31(1):1–28
3. de Berg M, Cheong O, van Kreveld M, Overmars M (2008) Computational geometry, 3rd edn. Springer, Berlin
4. de Berg M, de Groot M, Overmars MH (1997) Perfect binary space partitions. Comput Geom Theory Appl 7:81–91
5. de Berg M, Katz MJ, van der Stappen AF, Vleugels J (2002) Realistic input models for geometric algorithms. Algorithmica 34:81–97
6. Berman P, DasGupta B, Muthukrishnan S (2002) On the exact size of the binary space partitioning of sets of isothetic rectangles with applications. SIAM J Discret Math 15(2):252–267
7. Dumitrescu A, Mitchell JSB, Sharir M (2004) Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles. Discret Comput Geom 31(2):207–227
8. Hershberger J, Suri S (2003) Binary space partitions for 3D subdivisions. In: Proceedings of the 14th ACM-SIAM symposium on discrete algorithms, Baltimore. ACM, pp 100–108
9. Hershberger J, Suri S, Tóth CsD (2005) Binary space partitions of orthogonal subdivisions. SIAM J Comput 34(6):1380–1397
10. Paterson MS, Yao FF (1990) Efficient binary space partitions for hidden-surface removal and solid modeling. Discret Comput Geom 5:485–503
11. Paterson MS, Yao FF (1992) Optimal binary space partitions for orthogonal objects. J Algorithms 13:99–113
12. Tóth CsD (2003) A note on binary plane partitions. Discret Comput Geom 30:3–16
13. Tóth CsD (2005) Binary space partitions: recent developments. In: Goodman JE, Pach J, Welzl E (eds) Combinatorial and Computational Geometry. Volume 52 of MSRI Publications, Cambridge University Press, Cambridge, pp 529–556
14. Tóth CsD (2008) Binary space partition for axis-aligned fat rectangles. SIAM J Comput 38(1):429–447
15. Tóth CsD (2011) Binary plane partitions for disjoint line segments. Discret Comput Geom 45(4):617–646

Block Shaping in Floorplan

Chris Chu
Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA

Keywords

Block shaping; Fixed-outline floorplanning; Floorplanning; VLSI physical design

Years and Authors of Summarized Original Work

2012; Yan, Chu
2013; Yan, Chu

Problem Definition

Floorplanning is an early stage of the very-large-scale integration (VLSI) design process in which a coarse layout of a set of rectangular circuit blocks is determined. A floorplan enables designers to quickly estimate circuit performance, routing congestion, etc., of the circuit. In modern VLSI design flow, a fixed outline of the floorplanning region is given. On the other hand, many circuit blocks do not have a fixed shape during floorplanning as their internal circuitries have not yet been laid out. Those blocks are called soft blocks. Others blocks with predetermined shapes are called hard blocks. Given the geometric (i.e., left-right and above-below) relationship among the blocks, the block shaping problem is to determine the shapes of the soft blocks such that all blocks can be packed without overlap into the fixed-outline region.

To handle the block shaping problem in fixed-outline floorplanning, Yan and Chu [1,2] provide a problem formulation in which the floorplan height is minimized, while the width is upper bounded. The formulation is described below. Let W be the upper bound on the width of the floorplanning region. Given a set of n blocks, each block i has area A_i, width w_i, and height h_i. A_i is fixed, while w_i and h_i may vary as long as they satisfy $w_i \times h_i = A_i$, $W_i^{\min} \le w_i \le W_i^{\max}$, and $H_i^{\min} \le h_i \le H_i^{\max}$. If $W_i^{\min} = W_i^{\max}$ and $H_i^{\min} = H_i^{\max}$, then block i is a hard block. Two constraint graphs G_H and G_V [3, Chapter 10] are given to specify the geometric relationship among the blocks. G_H and G_V consist of $n+2$ vertices. Vertices 1 to n represent the n blocks. In addition, dummy vertices 0 (called source) and $n+1$ (called sink) are added. In G_H, vertices 0 and $n+1$ represent the leftmost and rightmost boundaries of the floorplanning region, respectively. In G_V, vertices 0 and $n+1$ represent the bottommost and topmost boundaries of the floorplanning region, respectively. A_0, w_0, h_0, A_{n+1}, w_{n+1}, and h_{n+1} are all set to 0. If block i is on the left of block j, $(i,j) \in G_H$. If block i is below block j, $(i,j) \in G_V$.

Let x_i and y_i be the x- and y-coordinates of the bottom-left corner of block i in the floorplan. Then, the block shaping problem formulation in [1,2] can be written as the following geometric program:

$$
\begin{aligned}
\text{Minimize } & y_{n+1} \\
\text{subject to } & x_{n+1} \le W \\
& x_i + w_i \le w_j && \forall (i,j) \in G_H \\
& y_i + h_i \le y_j && \forall (i,j) \in G_V \\
& w_i \times h_i = A_i && 1 \le i \le n \\
& W_i^{\min} \le w_i \le W_i^{\max} && 1 \le i \le n \\
& H_i^{\min} \le h_i \le H_i^{\max} && 1 \le i \le n \\
& x_0 = y_0 = 0
\end{aligned}
$$

To solve the original problem of packing all blocks into a fixed-outline region, we can take any feasible solution of the geometric program in which y_{n+1} is less than or equal to the height of the region.

Key Results

Almost all previous works target the classical floorplanning formulation, which minimizes the floorplan area. Such a formulation is not compatible with modern design methodologies [4], but those works may be modified to help fixed-outline floorplanning to various extents. For the special case of slicing floorplan [5], the block shaping problem can be solved by the elegant shape curve idea [6]. For a general floorplan which may not have a slicing structure, various heuristics have been proposed [7–9]. Moh et al. [10] formulated the shaping problem as a geometric program and optimally solved it using standard convex optimization. Young et al. [11] solved the geometric program formulation by Lagrangian relaxation. Lin et al. [12] minimized the floorplan area indirectly by minimizing its perimeter optimally using min-cost flow and trust region method. For previous works which directly tackled the block shaping problem in

fixed-outline floorplanning, Adya and Markov [13] proposed a simple greedy heuristic, and Lin and Hung [14] used second-order cone programming. Previous works are either non-optimal or time-consuming.

Yan and Chu [1, 2] presented a simple and optimal algorithm called slack-driven shaping (SDS). SDS iteratively shapes the soft blocks to reduce the floorplan height while not exceeding the floorplan width bound. We first present a simplified version called basic slack-driven shaping (basic SDS), which almost always produces an optimal solution. Then, we present its extension to the SDS algorithm.

Given some initial block shapes, the blocks can be packed to the four boundaries of the floorplanning region. For block i ($1 \le i \le n$), let Δ_{x_i} be the difference in x_i between the two layouts generated by packing all blocks to $x = W$ and to $x = 0$, respectively. Similarly, let Δ_{y_i} be the difference in y_i between the two layouts generated by packing all blocks to $y = y_{n+1}$ and to $y = 0$, respectively. The horizontal slack s_i^H and vertical slack s_i^V are defined as follows:

$$s_i^H = \max(0, \Delta_{x_i}),\ s_i^V = \max(0, \Delta_{y_i}).$$

Horizontal critical path (HCP) is defined as a path in G_H from source to sink such that all blocks along the path have zero horizontal slack. Vertical critical path (VCP) is similarly defined. We also define two subsets of blocks:

$$\begin{aligned} \text{SH} &= \{i \text{ is soft}\} \cap \{s_i^H > 0, s_i^V = 0\} \\ &\quad \cap \{w_i < W_i^{\max}\} \\ \text{SV} &= \{i \text{ is soft}\} \cap \{s_i^H = 0, s_i^V > 0\} \\ &\quad \cap \{h_i < H_i^{\max}\} \end{aligned}$$

Note that y_{n+1} can be reduced by decreasing the height (i.e., increasing the width) of the blocks in SH, and x_{n+1} can be reduced by decreasing the width (i.e., increasing the height) of the blocks in SV. We call the blocks in the sets SH and SV target soft blocks. In each iteration of basic SDS, we would like to increase the width w_i of each block $i \in$ SH by δ_i^H and the height h_i of each block $i \in$ SV by δ_i^V. The basic SDS algorithm is shown below:

Basic Slack-Driven Shaping Algorithm
Input: A set of n blocks, upper-bound width W, G_H and G_V.
Output: Optimized y_{n+1}, w_i and h_i for all i.
Begin
1. Set w_i to $W_i^{\min}$ for all i.
2. Pack blocks to $x = 0$ and compute x_{n+1}.
3. If $x_{n+1} > W$,
4. Return no feasible solution.
5. Else,
6. Repeat
7. Pack blocks to $y = 0$, $y = y_{n+1}$, $x = 0$, and $x = W$.
8. Calculate s_i^H and s_i^V for all i.
9. Identify target soft blocks in SH and SV.
10. $\forall i \in$ SH, increase w_i by $\delta_i^H = \dfrac{(W_i^{\max} - w_i)}{\text{MAX}_{p \in P_i^H}\left(\sum_{k \in p}(W_k^{\max} - w_k)\right)} s_i^H$,
where P_i^H is the set of paths in G_H passing through block i.
11. $\forall i \in$ SV, increase h_i by $\delta_i^V = \beta \times \dfrac{(H_i^{\max} - h_i)}{\text{MAX}_{p \in P_i^V}\left(\sum_{k \in p}(H_k^{\max} - h_k)\right)} s_i^V$,
where P_i^V is the set of paths in G_V passing through block i.
12. Until there is no target soft block.
End

Note that all δ_i^H and δ_i^V in Lines 10 and 11 can be computed using dynamic programming in linear time. Packing of blocks can also be done in linear time by longest path algorithm on a directed acyclic graph. Hence, each iteration of basic SDS takes linear time. The way δ_i^H and δ_i^V are set in Lines 10 and 11 is the key to the convergence of the algorithm.

Lemma 1 *For any path p from source to sink in G_H, we have $\sum_{i \in p} \delta_i^H \leq s^p_{\max\_H}$, where $s^p_{\max\_H}$ is the maximum horizontal slack over all blocks along p.*

Basically, $s^p_{\max\_H}$ gives us a budget on the total amount of increase in the block width along path p. Hence, Lemma 1 implies that the width of the floorplan will not be more than W after shaping of the blocks in SH at Line 10. The shaping of blocks in SV is done similarly, but a factor β is introduced in Line 11.

Lemma 2 *For any path p from source to sink in G_V, we have $\sum_{i \in p} \delta_i^V \leq \beta \times s^p_{\max\_V}$, where $s^p_{\max\_V}$ is the maximum vertical slack over all blocks along p.*

Lemma 2 guarantees that by setting $\beta \leq 1$, y_{n+1} will not increase after each iteration. In other words, the height of the floorplan will monotonically decrease during the whole shaping process. β is almost always set to 1. However, if $\beta = 1$, it is possible that the floorplan height may remain the same after one iteration even when the solution is not yet optimal. To avoid getting stuck at a local minimum, if the floorplan height does not decrease for two consecutive iterations, β is set to 0.9 for the next iteration.

Consider a shaping solution L generated by basic SDS (i.e., without any target soft block). Blocks at the intersection of some HCP and some VCP are called intersection blocks. The following optimality conditions were derived in [1, 2].

Lemma 3 *If L contains one VCP in which all intersection blocks are hard, then L is optimal.*

Lemma 4 *If L contains at most one HCP in which some intersection blocks are soft, then L is optimal.*

Lemma 5 *If L contains at most one VCP in which some intersection blocks are soft, then L is optimal.*

In practice, it is very rare for a shaping solution generated by basic SDS to satisfy none of the three optimality conditions. According to the experiments in [1, 2], all solutions by basic SDS satisfy at least one of the optimality conditions, i.e., are optimal. However, [1, 2] showed that it is possible for basic SDS to converge to non-optimal solutions. If a non-optimal solution is produced by basic SDS, it can be used as a starting solution to the geometric program above and then be improved by a single step of any descent-based optimization technique (e.g., deepest descent). This perturbed and improved solution can be fed to basic SDS again to be further improved. The resulting SDS algorithm, which is guaranteed optimal, is shown below:

Slack-Driven Shaping Algorithm
Input: A set of n blocks, upper-bound width W, G_H and G_V.
Output: Optimal y_{n+1}, w_i and h_i for all i.
Begin
1. Run basic SDS to generate shaping solution L.
2. If Lemma 3 or Lemma 4 or Lemma 5 is satisfied,
3. L is optimal. Exit.
4. Else,
5. Improve L by a single step of geometric programming.
6. Go to Line 1.
End

Applications

Floorplanning is a very important step in modern VLSI design. It enables designers to explore different alternatives in the design space and make critical decisions early in the design process. Typically, a huge number of alternatives need to be evaluated during the floorplanning stage. Hence, an efficient block shaping algorithm is a crucial component of a floorplanning tool. SDS is tens to hundreds of times faster than previous algorithms in practice. It also directly handles a fixed-outline floorplanning formulation, which is the standard in modern design methodologies. Hence, SDS should be able to improve the quality while also reduce the design time of VLSI circuits.

Open Problems

An interesting open problem is to derive a theoretical bound on the number of iterations for SDS to converge to an optimal solution. Although experimental results have shown that the number of iterations is small in practice, no theoretical bound is known.

Another interesting problem is to design an algorithm to achieve optimal block shaping entirely by simple slack-driven operations without resorting to geometric programming.

Besides, because of the similarity of the concept of slack in floorplanning and in circuit timing analysis, it would be interesting to see if a slack-driven approach similar to that in SDS can be applied to buffer and wire sizing for timing optimization.

Cross-References

- Floorplan and Placement
- Gate Sizing
- Slicing Floorplan Orientation
- Wire Sizing

Recommended Reading

1. Yan JZ, Chu C (2012) Optimal slack-driven block shaping algorithm in fixed-outline floorplanning. In: Proceedings of international symposium on physical design, Napa, pp 179–186
2. Yan JZ, Chu C (2013) SDS: an optimal slack-driven block shaping algorithm for fixed-outline floorplanning. IEEE Trans Comput Aided Design 32(2): 175–188
3. Wang L-T, Chang Y-W, Cheng K-T (eds) (2009) Electronic design automation: synthesis, verification, and test. Morgan Kaufmann, Burlington
4. Kahng A (2000) Classical floorplanning harmful? In: Proceedings of international symposium on physical design, San Diego, pp 207–213
5. Otten RHJM (1982) Automatic floorplan design. In: Proceedings of ACM/IEEE design automation conference, Las Vegas, pp 261–267
6. Stockmeyer L (1983) Optimal orientations of cells in slicing floorplan designs. Inf Control 57: 91–101
7. Wang TC, Wong DF (1992) Optimal floorplan area optimization. IEEE Trans Comput Aided Design 11(8):992–1001
8. Pan P, Liu CL (1995) Area minimization for floorplans. IEEE Trans Comput Aided Design 14(1): 129–132
9. Kang M, Dai WWM (1997) General floorplanning with L-shaped, T-shaped and soft blocks based on bounded slicing grid structure. In: Proceedings of ASP-DAC, Chiba, pp 265–270
10. Moh TS, Chang TS, Hakimi SL (1996) Globally optimal floorplanning for a layout problem. IEEE Trans Circuits Syst I 43:713–720
11. Young FY, Chu CCN, Luk WS, Wong YC (2001) Handling soft modules in general non-slicing floorplan using Lagrangian relaxation. IEEE Trans Comput Aided Design 20(5):687–692
12. Lin C, Zhou H, Chu C (2006) A revisit to floorplan optimization by Lagrangian relaxation. In: Proceedings of ICCAD, San Jose, pp 164–171
13. Adya SN, Markov IL (2003) Fixed-outline floorplanning: enabling hierarchical design. IEEE Trans VLSI Syst 11(6):1120–1135
14. Lin J-M, Hung Z-X (2011) UFO: unified convex optimization algorithms for fixed-outline floorplanning considering pre-placed modules. IEEE Trans Comput Aided Design 30(7):1034–1044

Boosting Textual Compression

Paolo Ferragina[1] and Giovanni Manzini[2,3]
[1]Department of Computer Science, University of Pisa, Pisa, Italy
[2]Department of Computer Science, University of Eastern Piedmont, Alessandria, Italy
[3]Department of Science and Technological Innovation, University of Piemonte Orientale, Alessandria, Italy

Keywords

Context-aware compression; High-order compression models

Years and Authors of Summarized Original Work

2005; Ferragina, Giancarlo, Manzini, Sciortino

Problem Definition

Informally, a boosting technique is a method that, when applied to a particular class of algorithms, yields improved algorithms. The improvement must be provable and well defined in terms of one or more of the parameters characterizing the algorithmic performance. Examples of boosters can be found in the context of randomized algorithms (here, a booster allows one to turn a *BPP* algorithm into an *RP* one [6]) and computational learning theory (here, a booster allows one to improve the prediction accuracy of a weak learning algorithm [10]). The problem of compression boosting consists of designing a technique that improves the compression performance of a wide class of algorithms. In particular, the results of Ferragina et al. provide a general technique for turning a compressor that uses no context information into one that always uses the best possible context.

The classic Huffman and arithmetic coding algorithms [1] are examples of *statistical* compressors which typically encode an input symbol according to its *overall* frequency in the data to be compressed. (In their dynamic versions these algorithms consider the frequency of a symbol in the already scanned portion of the input.) This approach is efficient and easy to implement but achieves poor compression. The compression performance of statistical compressors can be improved by adopting *higher*-order models that obtain better estimates for the frequencies of the input symbols. The PPM compressor [9] implements this idea by collecting (the frequency of) all symbols which follow *any* k-long context and by compressing them via arithmetic coding. The length k of the context is a parameter of the algorithm that depends on the data to be compressed: it is different if one is compressing English text, a DNA sequence, or an XML document. There exist other examples of sophisticated compressors that use context information in an *implicit* way, such as Lempel-Ziv and Burrows-Wheeler compressors [9]. All these context-aware algorithms are effective in terms of compression performance, but are usually rather complex to implement and difficult to analyze.

Applying the boosting technique of Ferragina et al. to Huffman or arithmetic coding yields a new compression algorithm with the following features: (i) the new algorithm uses the boosted compressor as a black box; (ii) the new algorithm compresses in a PPM-like style, automatically choosing the *optimal* value of k; and (iii) the new algorithm has essentially the same time/space asymptotic performance of the boosted compressor. The following sections give a precise and formal treatment of the three properties (i)–(iii) outlined above.

Key Results

Notation: The Empirical Entropy

Let s be a string over the alphabet $\Sigma = \{a_1, \ldots, a_h\}$, and for each $a_i \in \Sigma$, let n_i be the number of occurrences of a_i in s. The *0th order empirical entropy* of the string s is defined as $H_0(s) = -\sum_{i=1}^{h} (n_i/|s|) \log(n_i/|s|)$, where it is assumed that all logarithms are taken to the

base 2 and $0\log 0 = 0$. It is well known that H_0 is the maximum compression one can achieve using a uniquely decodable code in which a fixed codeword is assigned to each alphabet symbol. Greater compression is achievable if the codeword of a symbol depends on the k symbols following it (namely, its *context*). (In data compression it is customary to define the context looking at the symbols *preceding* the one to be encoded. The present entry uses the nonstandard "forward" contexts to simplify the notation of the following sections. Note that working with "forward" contexts is equivalent to working with the traditional "backward" contexts on the string s reversed (see [3] for details).) Let us define w_s as the string of single symbols immediately preceding the occurrences of w in s. For example, for $s = \texttt{bcabcabdca}$, it is $\texttt{ca}_s = \texttt{bbd}$. The value

$$H_k(s) = \frac{1}{|s|} \sum_{w \in \Sigma^k} |w_s| H_0(w_s) \qquad (1)$$

is the k-th order empirical entropy of s and is a lower bound to the compression one can achieve using codewords which only depend on the k symbols immediately following the one to be encoded.

Example 1 Let $s = \texttt{mississippi}$. For $k = 1$, it is $\texttt{i}_s = \texttt{mssp}$, $\texttt{s}_s = \texttt{isis}$, $\texttt{p}_s = \texttt{ip}$. Hence,

$$\begin{aligned} H_1(s) &= \frac{4}{11} H_0(\texttt{mssp}) + \frac{4}{11} H_0(\texttt{isis}) \\ &\quad + \frac{2}{11} H_0(\texttt{ip}) \\ &= \frac{6}{11} + \frac{4}{11} + \frac{2}{11} = \frac{12}{11}. \end{aligned}$$

Note that the empirical entropy is defined for any string and can be used to measure the performance of compression algorithms without any assumption on the input source. Unfortunately, for some (highly compressible) strings, the empirical entropy provides a lower bound that is too conservative. For example, for $s = \texttt{a}^n$, it is $|s| H_k(s) = 0$ for any $k \geq 0$. To better deal with highly compressible strings, [7] introduced the notion of *0th order modified empirical* entropy ${H_0}^*(s)$ whose property is that $|s| {H_0}^*(s)$ is at least equal to the number of bits needed to write down the length of s in binary. The *kth order modified empirical entropy* ${H_k}^*$ is then defined in terms of ${H_0}^*$ as the maximum compression one can achieve by looking at *no more than k* symbols following the one to be encoded.

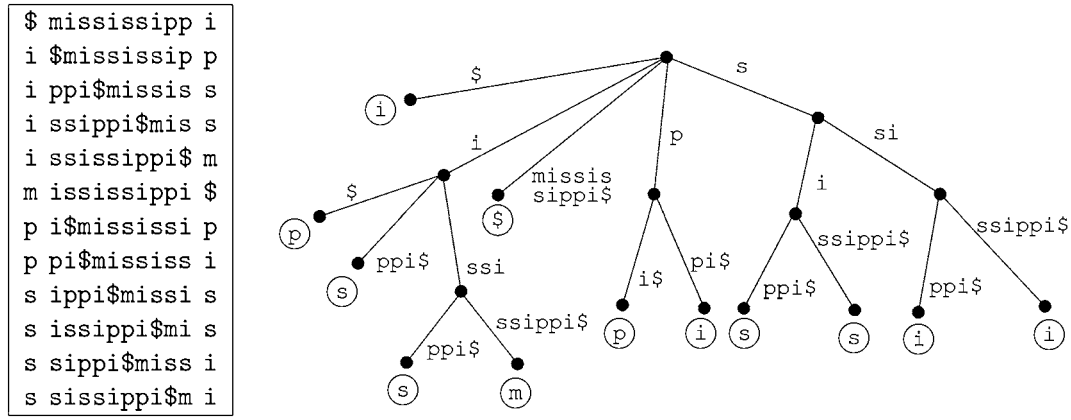
The Burrows-Wheeler Transform

Given a string s, the Burrows-Wheeler transform [2] (*bwt*) consists of three basic steps: (1) append to the end of s a special symbol \$ smaller than any other symbol in Σ; (2) form a *conceptual* matrix $\mathcal{M}$ whose rows are the cyclic shifts of the string $s\$$, sorted in lexicographic order; and (3) construct the transformed text $\hat{s} = \texttt{bwt}(s)$ by taking the last column of $\mathcal{M}$ (see Fig. 1). In [2] Burrows and Wheeler proved that $\hat{s}$ is a permutation of s, and that from $\hat{s}$ it is possible to recover s in $O(|s|)$ time.

To see the power of the *bwt*, the reader should reason in terms of empirical entropy. Fix a positive integer k. The first k columns of the *bwt* matrix contain, lexicographically ordered, all length-k substrings of s (and k substrings containing the symbol \$). For any length-k substring w of s, the symbols immediately preceding every occurrence of w in s are grouped together in a set of consecutive positions of $\hat{s}$ since they are the last symbols of the rows of $\mathcal{M}$ prefixed by w. Using the notation introduced for defining H_k, it is possible to rephrase this property by saying that the symbols of w_s are consecutive within $\hat{s}$ or, equivalently, that $\hat{s}$ contains, as a substring, a permutation $\pi_w(w_s)$ of the string w_s.

Example 2 Let $s = \texttt{mississippi}$ and $k = 1$. Figure 1 shows that $\hat{s}[1,4] = \texttt{pssm}$ is a permutation of $\texttt{i}_s = \texttt{mssp}$. In addition, $\hat{s}[6,7] = \texttt{pi}$ is a permutation of $\texttt{p}_s = \texttt{ip}$, and $\hat{s}[8,11] = \texttt{ssii}$ is a permutation of $\texttt{s}_s = \texttt{isis}$.

Since permuting a string does not change its (modified) 0th order empirical entropy (that is, $H_0(\pi_w(w_s)) = H_0(w_s)$), the Burrows-Wheeler transform can be seen as a tool for reducing the



Boosting Textual Compression, Fig. 1 The *bwt* matrix (*left*) and the suffix tree (*right*) for the string $s =$ `mississippi$`. The output of the *bwt* is the last column of the *bwt* matrix, i.e., $\hat{s} = \texttt{bwt}(s) =$ `ipssm$pissii`

problem of compressing s up to its kth order entropy to the problem of compressing *distinct portions* of $\hat{s}$ up to their 0*th order* entropy. To see this, assume partitioning of $\hat{s}$ into the substrings $\pi_w(w_s)$ by varying w over Σ^k. It follows that $\hat{s} = \bigsqcup_{w \in \Sigma^k} \pi_w(w_s)$ where $\bigsqcup$ denotes the concatenation operator among strings. (In addition to $\bigsqcup_{w \in \Sigma^k} \pi_w(w_s)$, the string $\hat{s}$ also contains the last k symbols of s (which do not belong to any w_s) and the special symbol \$. For simplicity these symbols will be ignored in the following part of the entry.) By (1) it follows that

$$\sum_{w \in \Sigma^k} |\pi_w(w_s)| H_0(\pi_w(w_s)) = \sum_{w \in \Sigma^k} |w_s| \, H_0(w_s) = |s| \, H_k(s).$$

Hence, to compress s up to $|s| \, H_k(s)$, it suffices to compress each substring $\pi_w(w_s)$ up to its 0th order empirical entropy. Note, however, that in the above scheme the parameter k must be chosen in advance. Moreover, a similar scheme cannot be applied to ${H_k}^*$ which is defined in terms of contexts of length *at most* k. As a result, no efficient procedure is known for computing the partition of $\hat{s}$ corresponding to ${H_k}^*(s)$. The compression booster [3] is a natural complement to the *bwt* and allows one to compress any string s up to $H_k(s)$ (or ${H_k}^*(s)$) simultaneously for all $k \geq 0$.

The Compression Boosting Algorithm

A crucial ingredient of compression boosting is the relationship between the *bwt* matrix and the suffix tree data structure. Let $\mathcal{T}$ denote the suffix tree of the string $s\$$. $\mathcal{T}$ has $|s| + 1$ leaves, one per suffix of $s\$$, and edges labeled with substrings of $s\$$ (see Fig. 1). Any node u of $\mathcal{T}$ has *implicitly associated* a substring of $s\$$, given by the concatenation of the edge labels on the downward path from the root of $\mathcal{T}$ to u. In that implicit association, the leaves of $\mathcal{T}$ correspond to the suffixes of $s\$$. Assume that the suffix tree edges are sorted lexicographically. Since each row of the *bwt* matrix is prefixed by one suffix of $s\$$ and rows are lexicographically sorted, the ith leaf (counting from the left) of the suffix tree corresponds to the ith row of the *bwt* matrix. Associate to the ith leaf of $\mathcal{T}$ the ith symbol of $\hat{s} = \texttt{bwt}(s)$. In Fig. 1 these symbols are represented inside circles.

For any suffix tree node u, let $\hat{s}\langle u \rangle$ denote the substring of $\hat{s}$ obtained by concatenating, from left to right, the symbols associated to the leaves descending from node u. Of course $\hat{s}\langle root(\mathcal{T}) \rangle = \hat{s}$. A subset $\mathcal{L}$ of $\mathcal{T}$'s nodes is called a *leaf cover* if every leaf of the suffix tree has a *unique* ancestor in $\mathcal{T}$. Any leaf cover $\mathcal{L} = \{u_1, \ldots, u_p\}$ naturally induces a partition of the leaves of $\mathcal{T}$.

Because of the relationship between $\mathcal{T}$ and the *bwt* matrix, this is also a partition of $\hat{s}$, namely, $\{\hat{s}\langle u_1\rangle, \ldots, \hat{s}\langle u_p\rangle\}$.

Example 3 Consider the suffix tree in Fig. 1. A leaf cover consists of all nodes of depth one. The partition of $\hat{s}$ induced by this leaf cover is {i,pssm,$,pi,ssii}.

Let C denote a function that associates to every string x over $\Sigma \cup \{\$\}$ a positive real value $C(x)$. For any leaf cover $\mathcal{L}$, define its cost as $C(\mathcal{L}) = \sum_{u \in \mathcal{L}} C(\hat{s}\langle u\rangle)$. In other words, the cost of the leaf cover $\mathcal{L}$ is equal to the sum of the costs of the strings in the partition induced by $\mathcal{L}$. A leaf cover $\mathcal{L}_{\min}$ is called *optimal* with respect to C if $C(\mathcal{L}_{\min}) \leq C(\mathcal{L})$, for any leaf cover $\mathcal{L}$.

Let A be a compressor such that, for any string x, its output size is bounded by $|x|H_0(x)+\eta|x|+\mu$ bits, where η and μ are constants. Define the cost function $C_{\mathtt{A}}(x) = |x|H_0(x) + \eta|x| + \mu$. In [3] Ferragina et al. exhibit a linear-time greedy algorithm that computes the optimal leaf cover $\mathcal{L}_{\min}$ with respect to $C_{\mathtt{A}}$. The authors of [3] also show that, for any $k \geq 0$, there exists a leaf cover $\mathcal{L}_k$ of cost $C_{\mathtt{A}}(\mathcal{L}_k) = |s|\, H_k(s)+\eta|s|+O(|\Sigma|^k)$. These two crucial observations show that, if one uses A to compress each substring in the partition induced by the optimal leaf cover $\mathcal{L}_{\min}$, the total output size is bounded in terms of $|s|\, H_k(s)$, for any $k \geq 0$. In fact,

$$\sum_{u \in \mathcal{L}_{\min}} C_{\mathtt{A}}(\hat{s}\langle u\rangle) = C_{\mathtt{A}}(\mathcal{L}_{\min}) \leq C_{\mathtt{A}}(\mathcal{L}_k) = |s|\, H_k(s) + \eta|s| + O(|\Sigma|^k)$$

In summary, boosting the compressor A over the string s consists of three main steps:

1. Compute $\hat{s} = \mathtt{bwt}(s)$.
2. Compute the optimal leaf cover $\mathcal{L}_{\min}$ with respect to $C_{\mathtt{A}}$ and partition $\hat{s}$ according to $\mathcal{L}_{\min}$.
3. Compress each substring of the partition using the algorithm A.

So the boosting paradigm reduces the design of effective compressors that use context information, to the (usually easier) design of 0th order compressors. The performance of this paradigm is summarized by the following theorem.

Theorem 1 ([3]) *Let A be a compressor that squeezes any string x in at most $|x|\, H_0(x) + \eta|x| + \mu$ bits. The compression booster applied to A produces an output whose size is bounded by $|s|\, H_k(s) + \log|s| + \eta|s| + O(|\Sigma|^k)$ bits simultaneously for all $k \geq 0$. With respect to A, the booster introduces a space overhead of $O(|s| \log |s|)$ bits and no asymptotic time overhead in the compression process.* □

A similar result holds for the modified entropy $H_k{}^*$ as well (but it is much harder to prove): given a compressor A that squeezes any string x in at most $\lambda|x|\, H_0^*(x) + \mu$ bits, the compression booster produces an output whose size is bounded by $\lambda|s|\, H_k^*(s) + \log|s| + O(|\Sigma|^k)$ bits, simultaneously for all $k \geq 0$. In [3] the authors also show that no compression algorithm, satisfying some mild assumptions on its inner working, can achieve a similar bound in which both the multiplicative factor λ and the additive logarithmic term are dropped simultaneously. Furthermore [3] proposes an instantiation of the booster which compresses any string s in at most 2.5 $|s|H_k{}^*(s) + \log|s| + O(|\Sigma|^k)$ bits. This bound is analytically superior to the bounds proven for the best existing compressors including Lempel-Ziv, Burrows-Wheeler, and PPM compressors.

Applications

Apart from the natural application in data compression, compressor boosting has been used also to design compressed full-text indexes [8].

Open Problems

The boosting paradigm may be generalized as follows: given a compressor A, find a permutation $\mathcal{P}$ for the symbols of the string s *and* a partitioning strategy such that the boosting approach, applied to them, minimizes the output size. These pages have provided convincing evidence that the Burrows-Wheeler transform is an

elegant and efficient permutation $\mathcal{P}$. Surprisingly enough, other classic data compression problems fall into this framework: shortest common superstring (which is MAX-SNP hard), run length encoding for a set of strings (which is polynomially solvable), *LZ77*, and minimum number of phrases (which is MAX-SNP hard). Therefore, the boosting approach is general enough to deserve further theoretical and practical attention [5].

Experimental Results

An investigation of several compression algorithms based on boosting and a comparison with other state-of-the-art compressors are presented in [4]. The experiments show that the boosting technique is more robust than other *bwt*-based approaches and works well even with less effective 0th order compressors. However, these positive features are achieved using more (time and space) resources.

Data Sets

The data sets used in [4] are available from http://people.unipmn.it/manzini/boosting. Other data sets for compression and indexing are available at the Pizza&Chili site http://pizzachili.di.unipi.it/.

URL to Code

The compression boosting page (http://people.unipmn.it/manzini/boosting) contains the source code of all the algorithms tested in [4]. The code is organized in a highly modular library that can be used to boost any compressor even without knowing the *bwt* or the boosting procedure.

Cross-References

- Arithmetic Coding for Data Compression
- Burrows-Wheeler Transform
- Compressing and Indexing Structured Text
- Suffix Array Construction
- Suffix Tree Construction
- Suffix Tree Construction in Hierarchical Memory
- Table Compression

Recommended Reading

1. Bell TC, Cleary JG, Witten IH (1990) Text compression. Prentice Hall, Englewood
2. Burrows M, Wheeler D (1994) A block sorting lossless data compression algorithm. Technical report 124, Digital Equipment Corporation
3. Ferragina P, Giancarlo R, Manzini G, Sciortino M (2005) Boosting textual compression in optimal linear time. J ACM 52:688–713
4. Ferragina P, Giancarlo R, Manzini G (2006) The engineering of a compression boosting library: theory vs practice in bwt compression. In: Proceedings of the 14th European symposium on algorithms (ESA), Zurich. LNCS, vol 4168. Springer, Berlin, pp 756–767
5. Giancarlo R, Restivo A, Sciortino M (2007) From first principles to the Burrows and Wheeler transform and beyond, via combinatorial optimization. Theor Comput Sci 387(3):236–248
6. Karp R, Pippenger N, Sipser M (1985) A time-randomness tradeoff. In: Proceedings of the conference on probabilistic computational complexity, Santa Barbara. AMS, pp 150–159
7. Manzini G (2001) An analysis of the Burrows-Wheeler transform. J ACM 48:407–430
8. Navarro G, Mäkinen V (2007) Compressed full text indexes. ACM Comput Surv 39(1):Article No. 2
9. Salomon D (2007) Data compression: the complete reference, 4th edn. Springer, New York
10. Schapire RE (1990) The strength of weak learnability. Mach Learn 2:197–227

Branchwidth of Graphs

Fedor V. Fomin[1] and Dimitrios Thilikos[2,3]
[1]Department of Informatics, University of Bergen, Bergen, Norway
[2]AlGCo Project Team, CNRS, LIRMM, France
[3]Department of Mathematics, National and Kapodistrian University of Athens, Athens, Greece

Keywords

Tangle Number

Years and Authors of Summarized Original Work

2003; Fomin, Thilikos

Problem Definition

Branchwidth, along with its better-known counterpart, treewidth, are measures of the "global connectivity" of a graph.

Definition

Let G be a graph on n vertices. A *branch decomposition* of G is a pair (T, τ), where T is a tree with vertices of degree 1 or 3 and τ is a bijection from the set of leaves of T to the edges of G. The *order*, we denote it as $\alpha(e)$, of an edge e in T is the number of vertices v of G such that there are leaves t_1, t_2 in T in different components of $T(V(T), E(T) - e)$ with $\tau(t_1)$ and $\tau(t_2)$ both containing v as an endpoint.

The *width* of (T, τ) is equal to $\max_{e \in E(T)} \{\alpha(e)\}$, i.e., is the maximum order over all edges of T. The *branchwidth* of G is the minimum width over all the branch decompositions of G (in the case where $|E(G)| \leq 1$, then we define the branchwidth to be 0; if $|E(G)| = 0$, then G has no branch decomposition; if $|E(G)| = 1$, then G has a branch decomposition consisting of a tree with one vertex – the width of this branch decomposition is considered to be 0).

The above definition can be directly extended to hypergraphs where τ is a bijection from the leaves of T to the hyperedges of G. The same definition can easily be extended to matroids.

Branchwidth was first defined by Robertson and Seymour in [25] and served as a main tool for their proof of Wagner's Conjecture in their Graph Minors series of papers. There, branchwidth was used as an alternative to the parameter of treewidth as it appeared easier to handle for the purposes of the proof. The relation between branchwidth and treewidth is given by the following result.

Theorem 1 ([25]) *If G is a graph, then* $\text{branchwidth}(G) \leq \text{treewidth}(G) + 1 \leq \lfloor 3/2\ \text{branchwidth}(G) \rfloor$.

The algorithmic problems related to branchwidth are of two kinds: first find fast algorithms computing its value and, second, use it in order to design fast dynamic programming algorithms for other problems.

Key Results

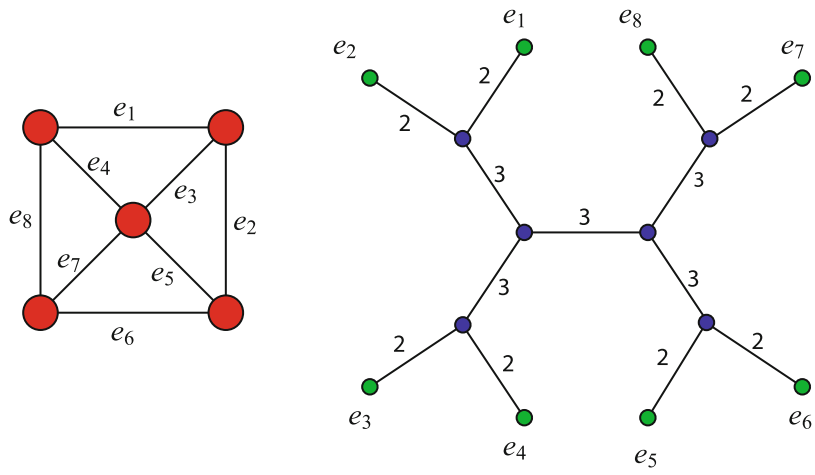
Algorithms for Branchwidth

Computing branchwidth is an NP-hard problem ([29]). Moreover, the problem remains NP-hard even if we restrict its input graphs to the class of split graphs or bipartite graphs [20].

On the positive side, branchwidth is computable in polynomial time on interval graphs [20, 24], and circular arc graphs [21]. Perhaps the most celebrated positive result on branchwidth is an $O(n^2)$ algorithm for the branchwidth of planar graphs, given by Seymour and Thomas in [29]. In the same paper they also give an $O(n^4)$ algorithm to compute an optimal branch decomposition. (The running time of this algorithm has been improved to $O(n^3)$ in [18].) The algorithm in [29] is basically an algorithm for a parameter called carving width, related to telephone routing and the result for branchwidth follows from the fact that the branch width of a planar graph is half of the carving-width of its medial graph.

The algorithm for planar graphs [29] can be used to construct an approximation algorithm for branchwidth of some non-planar graphs. On graph classes excluding a single crossing graph as a minor branchwidth can be approximated within a factor of 2.25 [7] (a graph H is a *minor* of a graph G if H can be obtained by a subgraph of G after applying edge contractions). Finally, it follows from [13] that for every minor closed graph class, branchwidth can be approximated by a constant factor.

Branchwidth cannot increase when applying edge contractions or removals. According to the Graph Minors theory, this implies that, for any fixed k, there is a finite number of minor minimal graphs of branchwidth more than k and we denote this set of graphs by $\mathcal{B}_k$. Checking whether a graph G contains a fixed graph as a minor can be done in polynomial time [27].



Branchwidth of Graphs, Fig. 1 Example of a graph and its branch decomposition of width 3

Therefore, the knowledge of $\mathcal{B}_k$ implies the construction of a polynomial time algorithm for deciding whether branchwidth$(G) \leq k$, for any fixed k. Unfortunately $\mathcal{B}_k$ is known only for small values of k. In particular, $\mathcal{B}_0 = \{P_2\}, B_1 = \{P_4, K_3\}, \mathcal{B}_2 = \{K_4\}$ and $\mathcal{B}_3 = \{K_5, V_8, M_6, Q_3\}$ (here K_r is a clique on r vertices, P_r is a path on r edges, V_8 is the graph obtained by a cycle on 8 vertices if we connect all pairs of vertices with cyclic distance 4, M_6 is the octahedron, and Q_3 is the 3-dimensional cube). However, for any fixed k, one can construct a linear, on $n = |V(G)|$, algorithm that decides whether an input graph G has branchwidth $\leq k$ and, if so, outputs the corresponding branch decomposition (see [3]). In technical terms, this implies that the problem of asking, for a given graph G, whether branchwidth$(G) \leq k$, parameterized by k is fixed parameter tractable (i.e., belongs in the parameterized complexity class FPT). (See [12] for further references on parameterized algorithms and complexity.) The algorithm in [3] is complicated and uses the technique of characteristic sequences, which was also used in order to prove the analogous result for treewidth. For the particular cases where $k \leq 3$, simpler algorithms exist that use the "reduction rule" technique (see [4]). We stress that $\mathcal{B}_4$ remains unknown while several elements of it have been detected so far (including the dodecahedron and the icosahedron graphs). There is a number of algorithms that for a given k in time $2^{O(k)} \cdot n^{O(1)}$ either decide that the branchwidth of a given graph is at least k, or construct a branch decomposition of width $O(k)$ (see [26]). These results can be generalized to compute the branchwidth of matroids and even more general parameters.

An exact algorithm for branchwidth appeared in [14]. Its complexity is $O((2 \cdot \sqrt{3})^n \cdot n^{O(1)})$. The algorithm exploits special properties of branchwidth (see also [24]).

In contrast to treewidth, edge maximal graphs of given branchwidth are not so easy to characterize (for treewidth there are just k-trees, i.e., chordal graphs with all maximal cliques of size $k + 1$). An algorithm for generating such graphs has been given in [23] and reveals several structural issues on this parameter.

It is known that a large number of graph theoretical problems can be solved in linear time when their inputs are restricted to graphs of small (i.e., fixed) treewidth or branchwidth (see [2]).

Branchwidth appeared to be a useful tool in the design of exact subexponential algorithms on planar graphs and their generalizations. The basic idea behind this approach is very simple: Let $\mathcal{P}$ be a problem on graphs and $\mathcal{G}$ be a class of graphs such that

- for every graph $G \in \mathcal{G}$ of branchwidth at most ℓ, the problem $\mathcal{P}$ can be solved in time $2^{c\cdot\ell} \cdot n^{\mathcal{O}(1)}$, where c is a constant, and;
- for every graph $G \in \mathcal{G}$ on n vertices a branch decomposition (not necessarily optimal) of G of width at most $h(n)$ can be constructed in polynomial time, where $h(n)$ is a function.

Then for every graph $G \in \mathcal{G}$, the problem $\mathcal{P}$ can be solved in time $2^{c\cdot h(n)} \cdot n^{\mathcal{O}(1)}$. Thus, everything boils down to computations of constants c and functions $h(n)$. These computations can be quite involved. For example, as was shown in [17], for every planar graph G on n vertices, the branchwidth of G is at most $\sqrt{4.5n} < 2.1214\sqrt{n}$. For extensions of this bound to graphs embeddable on a surface of genus g, see [15].

Dorn [9] used fast matrix multiplication in dynamic programming to estimate the constants c for a number of problems. For example, for the Maximum Independent Set problem, $c \leq \omega/2$, where $\omega < 2.376$ is the matrix product exponent over a ring, which implies that the Independent Set problem on planar graphs is solvable in time $O(2^{2.52\sqrt{n}})$. For the Minimum Dominating Set problem, $c \leq 4$, thus implying that the branch decomposition method runs in time $O(2^{3.99\sqrt{n}})$. It appears that algorithms of running time $2^{O(\sqrt{n})}$ can be designed even for some of the "non-local" problems, such as the Hamiltonian Cycle, Connected Dominating Set, and Steiner Tree, for which no time $2^{O(\ell)} \cdot n^{\mathcal{O}(1)}$ algorithm on general graphs of branchwidth ℓ is known [11]. Here one needs special properties of some optimal planar branch decompositions, roughly speaking that every edge of T corresponds to a disk on a plane such that all edges of G corresponding to one component of $T - e$ are inside the disk and all other edges are outside. Some of the subexponential algorithms on planar graphs can be generalized for graphs embedded on surfaces [10] and, more generally, to graph classes that are closed under taking of minors [8].

A similar approach can be used for parameterized problems on planar graphs. For example, a parameterized algorithm that finds a dominating set of size $\leq k$ (or reports that no such set exists) in time $2^{\mathcal{O}(\sqrt{k})}n^{\mathcal{O}(1)}$ can be obtained based on the following observations: there is a constant c such that every planar graph of branchwidth at least $c\sqrt{k}$ does not contain a dominating set of size at most k. Then for a given k the algorithm computes an optimal branch decomposition of a palanar graph G and if its width is more than $c\sqrt{k}$ concludes that G has no dominating set of size k. Otherwise, find an optimal dominating set by performing dynamic programming in time $2^{\mathcal{O}(\sqrt{k})}n^{\mathcal{O}(1)}$. There are several ways of bounding a parameter of a planar graph in terms of its branchwidth or treewidth including techniques similar to Baker's approach from approximation algorithms [1], the use of separators, or by some combinatorial arguments, as shown in [16]. Another general approach of bounding the branchwidth of a planar graph by parameters, is based on the results of Robertson et al. [28] regarding quickly excluding a planar graph. This brings us to the notion of *bidimensionality* [6]. Parameterized algorithms based on branch decompositions can be generalized from planar graphs to graphs embedded on surfaces and to graphs excluding a fixed graph as a minor.

Applications

See [5] for using branchwidth for solving TSP.

Open Problems

1. It is known that any planar graph G has branchwidth at most $\sqrt{4.5}\cdot\sqrt{|V(G)|}$ (or at most $\frac{3}{2}\cdot\sqrt{|E(G)|+2}$) [17]. Is it possible to improver this upper bound? Any possible improvement would accelerate many of the known exact or parameterized algorithms on planar graphs that use dynamic programming on branch decompositions.
2. In contrast to treewidth, very few graph classes are known where branchwidth is computable in polynomial time. Find graphs

classes where branchwidth can be computed or approximated in polynomial time.
3. Find $\mathcal{B}_k$ for values of k bigger than 3. The only structural result on $\mathcal{B}_k$ is that its planar elements will be either self-dual or pairwise-dual. This follows from the fact that dual planar graphs have the same branchwidth [29, 16].
4. Find an exact algorithm for branchwidth of complexity $O^*(2^n)$ (the notation $O^*()$ assumes that we drop the non-exponential terms in the classic $O()$ notation).
5. The dependence on k of the linear time algorithm for branchwidth in [3] is huge. Find an $2^{O(k)} \cdot n^{O(1)}$ step algorithm, deciding whether the branchwidth of an n-vertex input graph is at most k.

Cross-References

- ▸ Bidimensionality
- ▸ Treewidth of Graphs

Recommended Reading

1. Alber J, Bodlaender HL, Fernau H, Kloks T, Niedermeier R (2002) Fixed parameter algorithms for dominating set and related problems on planar graphs. Algorithmica 33:461–493
2. Arnborg S (1985) Efficient algorithms for combinatorial problems on graphs with bounded decomposability – a survey. BIT 25:2–23
3. Bodlaender HL, Thilikos DM (1997) Constructive linear time algorithms for branchwidth. In: Automata, languages and programming (Bologna, 1997). Lecture notes in computer science, vol 1256. Springer, Berlin, pp 627–637
4. Bodlaender HL, Thilikos DM (1999) Graphs with branchwidth at most three. J Algoritm 32:167–194
5. Cook W, Seymour PD (2003) Tour merging via branch-decomposition. INFORMS J Comput 15:233–248
6. Demaine ED, Fomin FV, Hajiaghayi M, Thilikos DM (2004) Bidimensional parameters and local treewidth. SIAM J Discret Math 18:501–511
7. Demaine ED, Hajiaghayi MT, Nishimura N, Ragde P, Thilikos DM (2004) Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. J Comput Syst Sci 69:166–195
8. Dorn F, Fomin FV, Thilikos DM (2006) Subexponential algorithms for non-local problems on H-minor-free graphs. In: Proceedings of the nineteenth annual ACM-SIAM symposium on discrete algorithms (SODA 2008). Society for Industrial and Applied Mathematics, Philadelphia, pp 631–640
9. Dorn F (2006) Dynamic programming and fast matrix multiplication. In: Proceedings of the 14th annual European symposium on algorithms (ESA 2006). Lecture notes in computer science, vol 4168. Springer, Berlin, pp 280–291
10. Dorn F, Fomin FV, Thilikos DM (2005) Fast subexponential algorithm for non-local problems on graphs of bounded genus. In: Proceedings of the 10th Scandinavian workshop on algorithm theory (SWAT 2006). Lecture notes in computer science. Springer, Berlin
11. Dorn F, Penninkx E, Bodlaender H, Fomin FV (2005) Efficient exact algorithms on planar graphs: exploiting sphere cut branch decompositions. In: Proceedings of the 13th annual European symposium on algorithms (ESA 2005). Lecture notes in computer science, vol 3669. Springer, Berlin, pp 95–106
12. Downey RG, Fellows MR (1999) Parameterized complexity. In: Monographs in computer science. Springer, New York
13. Feige U, Hajiaghayi M, Lee JR (2005) Improved approximation algorithms for minimum-weight vertex separators. In: Proceedings of the 37th annual ACM symposium on theory of computing (STOC 2005). ACM Press, New York, pp 563–572
14. Fomin FV, Mazoit F, Todinca I (2005) Computing branchwidth via efficient triangulations and blocks. In: Proceedings of the 31st workshop on graph theoretic concepts in computer science (WG 2005). Lecture notes computer science, vol 3787. Springer, Berlin, pp 374–384
15. Fomin FV, Thilikos DM (2004) Fast parameterized algorithms for graphs on surfaces: linear kernel and exponential speed-up. In: Proceedings of the 31st international colloquium on automata, languages and programming (ICALP 2004). Lecture notes computer science, vol 3142. Springer, Berlin, pp 581–592
16. Fomin FV, Thilikos DM (2006) Dominating sets in planar graphs: branch-width and exponential speed-up. SIAM J Comput 36:281–309
17. Fomin FV, Thilikos DM (2006) New upper bounds on the decomposability of planar graphs. J Graph Theory 51:53–81
18. Gu QP, Tamaki H (2005) Optimal branch-decomposition of planar graphs in O(n3) time. In: Proceedings of the 32nd international colloquium on automata, languages and programming (ICALP 2005). Lecture notes computer science, vol 3580. Springer, Berlin, pp 373–384
19. Gu QP, Tamaki H (2006) Branch-width, parse trees, and monadic second-order logic for matroids. J Comb Theory Ser B 96:325–351
20. Kloks T, Kratochvíl J, Müller H (2005) Computing the branchwidth of interval graphs. Discret Appl Math 145:266–275

21. Mazoit F (2006) The branch-width of circular-arc graphs. In: 7th Latin American symposium on theoretical informatics (LATIN 2006)
22. Oum SI, Seymour P (2006) Approximating clique-width and branch-width. J Comb Theory Ser B 96:514–528
23. Paul C, Proskurowski A, Telle JA (2006) Generating graphs of bounded branchwidth. In: Proceedings of the 32nd workshop on graph theoretic concepts in computer science (WG 2006). Lecture notes computer science, vol 4271. Springer, Berlin, pp 205–216
24. Paul C, Telle JA (2005) New tools and simpler algorithms for branchwidth. In: Proceedings of the 13th annual European symposium on algorithms (ESA 2005)
25. Robertson N, Seymour PD (1991) Graph minors. X. Obstructions to tree-decomposition. J Comb Theory Ser B 52:153–190
26. Robertson N, Seymour PD (1995) Graph minors. XII. Distance on a surface. J Comb Theory Ser B 64:240–272
27. Robertson N, Seymour PD (1995) Graph minors. XIII. The disjoint paths problem. J Comb Theory Ser B 63:65–110
28. Robertson N, Seymour PD, Thomas R (1994) Quickly excluding a planar graph. J Comb Theory Ser B 62:323–348
29. Seymour PD, Thomas R (1994) Call routing and the ratcatcher. Combinatorica 14:217–241

Broadcast Scheduling – Minimizing Average Response Time

Ravishankar Krishnaswamy
Computer Science Department, Princeton University, Princeton, NJ, USA

Keywords

Broadcast scheduling; Combinatorial optimization; Discrepancy; Linear programming; Rounding; Scheduling

Years and Authors of Summarized Original Work

2005; Bansal, Charikar, Khanna, Naor
2008; Bansal, Coppersmith, Sviridenko
2014; Bansal, Charikar, Krishnaswamy, Li

Problem Definition

In this entry, we consider the classical broadcast scheduling problem and discuss some recent advances on this problem. The problem is formalized as follows: there is a server which has a collection of unit-sized *pages* $P = \{1, \ldots, n\}$. The server can broadcast pages in integer time slots in response to *requests*, which are given as the following sequence: at time t, the server receives $w_p(t) \in \mathbb{Z}_{\geq 0}$ requests for each page $p \in P$. We say that a request ρ for page p that arrives at time t is satisfied at time $c_p(t)$ if $c_p(t)$ is the first time after t by which the server has completely transmitted page p. The response time of the request ρ is defined to be $c_p(t) - t$, i.e., the time that elapses from its arrival till the time it is satisfied. Notice that by definition, the response time for any request is at least 1. The goal is to find a schedule for broadcasting pages to minimize the average response time, i.e., $(\sum_{t,p} w_p(t)(c_p(t)-t))/\sum_{t,p} w_p(t)$. Recall that the problem we discuss here is an *offline problem*, where the entire request sequence is specified as part of the input. There has also been much research on the online version of the problem, and we briefly discuss this toward the end of the entry.

Key Results

Erlebach and Hall [5] were the first to show complexity theoretic hardness for this problem by showing that it is NP complete. The techniques we describe below were introduced in [1, 3]. By fine-tuning these ideas, [2] shows the following result on the approximability of the offline problem, which will be the main result we will build toward this entry.

Theorem 1 ([2]) *Let $\gamma > 0$ be any arbitrary parameter. There is a polynomial time algorithm that finds a schedule with average response time $(2+\gamma)\cdot OPT + O((\sqrt{\log_{1+\gamma} n \cdot \log\log n})\log n)$, where OPT denotes the value of the average response time in the optimum solution.*

By setting $\gamma = \Theta(\log n)$ above, we can get an approximation guarantee of $O(\log^{1.5} n)$. Also

note that the $O(\log^{1.5} n)$ term in Theorem 1 is additive. As a result, for instance, where OPT is large (say $\Omega(\log^{1.5+\epsilon} n)$ for some $\epsilon > 0$), we can set γ arbitrarily small to get an approximation ratio arbitrarily close to 2.

Linear Programming Formulation

All of the algorithmic results in [1–3] are based on rounding the following natural LP relaxation for the problem. For each page $p \in [n]$ and each time t, there is a variable y_{pt} which indicates whether page p was transmitted at time t. We have another set of variables $x_{ptt'}$ s.t $t' > t$, which indicates whether a request for page p which arrives at time t is satisfied at t'. Let $w_p(t)$ denote the total weight of requests for page p that arrive at time t.

$$\min \sum_{p,t,t'>t} (t'-t) \cdot \omega_p(t) \cdot x_{ptt'} \tag{1}$$

$$\text{s.t.} \sum_{p} y_{pt} \leq 1 \qquad \forall t \tag{2}$$

$$\sum_{t'>t} x_{ptt'} \geq 1 \qquad \forall p, t \tag{3}$$

$$x_{ptt'} \leq y_{pt'} \qquad \forall p, t, t' \geq t \tag{4}$$

$$x_{ptt'}, y_{pt'} \in [0,1] \qquad \forall p, t, t' \tag{5}$$

Constraint (2) ensures that only one page is transmitted in each time, (3) ensures that each request must be satisfied, and (4) ensures that a request for page p can be satisfied at time t only if p is transmitted at time t. Finally, a request arriving at time t that is satisfied at time t' contributes $(t'-t)$ to the objective. Now consider the linear program obtained by relaxing the integrality constraints on $x_{ptt'}$ and y_{pt}.

Rounding Techniques

The following points illustrate the main ideas that form the building blocks of the rounding algorithms in [1–3].

The Half-Integrality Assumption

In what follows, we discuss the techniques in the special case that the LP solution is *half-integral*, i.e., where all the $x_{ptt'} \in \{0, \frac{1}{2}\}$. The general case essentially builds upon this intuition, and all main technical ingredients are contained in this special case.

Viewing the LP Solution as a Convex Combination of Blocks

In half-integral solutions, note that every request is satisfied by the two earliest half broadcasts of the corresponding page. For any page p, let $\tau_p = \{t_{p,1}, t_{p,2}, \ldots\}$ denote the times when the fractional solution broadcasts $\frac{1}{2}$ units of page p. Notice that the fractional solution can be entirely characterized by these sets τ_ρ for all pages p. The main intuition now is to view the fractional broadcast of each page as a *convex combination* of two different solutions, one which broadcasts the page p integrally at the odd times $\{t_{p,1}, t_{p,3}, \ldots\}$ and another which broadcasts the page p integrally at the even times $\{t_{p,2}, t_{p,4}, \ldots\}$. We call these the *odd schedule* and even *schedule* for page p.

Rounding the Solution to Minimize Backlog: Attempt 1 [1]

Our first and most natural rounding idea is to round the convex combination for each page into one of the odd or even schedules, each with probability $1/2$. Let us call this the *tentative schedule*. Note that on average, the tentative schedule broadcasts one page per time slot, and moreover, the expected response time of any request is equal to its fractional response time. The only issue, however, is that different pages may broadcast at the same time slots. Indeed, there could some time interval $[t_1,t_2)$ where the tentative schedule makes many more than $t_2 - t_1$ broadcasts! A natural manner to resolve this issue is to broadcast conflicting pages in a first-come first-serve manner, breaking ties arbitrarily. Now, the typical request waits for at most its fractional cost (on average), plus the *backlog* due to conflicting broadcasts. Formally, the backlog is defined as $\max_{t_1,t_2>t_1} N_A(t_1,t_2) - (t_2 - t_1)$, where $N_A(t_1,t_2)$ is the number of broadcasts made in

the interval $[t_1, t_2)$ by the tentative schedule. For this simple randomized algorithm, note that the backlog of any interval $[t_1, t_2)$ is at most $\tilde{O}(\sqrt{n})$ w.h.p by a standard concentration bound. This can be formalized to give us the $\tilde{O}(\sqrt{n})$ approximation algorithm of [1].

Rounding the Solution to Minimize Backlog: Attempt 2 [3]

Our next attempt involves controlling the backlog by explicitly enforcing constraints which periodically reset the backlog to 0. For this, we write an auxiliary LP as follows: we divide the set τ_p for each page p into blocks of size $B = \Theta(\log n)$ units each and have a variable for choosing the odd schedule or even schedule within each block. (If the first block chooses an odd schedule and the second block chooses an even schedule, then the requests which arrived at the boundary may incur greater costs, but [3] argues that these can be bounded for $B \geq \Omega(\log n)$.) Since each block has $B/2$ units of fractional transmission (recall that the LP is half-integral), the total number of blocks is at most $2T/B$, where T is the time horizon of all broadcasts made by the LP solution. Therefore, the total number of variables is at most $4T/B$ (each block chooses either an odd schedule or even schedule). Now, instead of asking for the LP to choose schedules such that each time has at most one transmission, suppose we *group the time slots into intervals of size B and ask for the LP to choose schedules such that each interval has at most B transmissions.* Now, there are T/B such constraints, and there are $2T/B$ constraints which enforce that we pick one schedule for each block.

Therefore, in this relaxed LP, we start with a solution that has $4T/B$ variables each set to $1/2$ and a total of $3T/B$ constraints. But now, we can convert this into an *optimal basic feasible solution* (where the number of nonzero variables is at most the number of constraints). This implies that at least a constant fraction of the blocks chooses either the odd or even schedules integrally. It is easy to see that the backlog incurred in any time interval $[t_1, t_2)$ is at most $O(B)$ since we explicitly enforce 0 backlog for consecutive intervals of size B. Therefore, by repeating this process $O(\log T)$ time, we get a fully integral schedule with backlog $O(\log TB) = O(\log^2 n)$. This then gives us the $2 \cdot \text{OPT} + O(\log^2 n)$ approximation guarantee of [3].

Rounding the Solution to Minimize Backlog: Attempt 3 [2]

Our final attempt involves combining the ideas of attempts 1 and 2. Indeed, the main issue with approach 2 is that, when we solve for a basic feasible solution, we lose all control over how the solution looks! Therefore, we would ideally like for a rounding which enforces the constraints on time intervals of size B, but still *randomly selects the schedules within each block*. This way, we'll be able to argue that within each time interval of size B, the maximum backlog is $O(\sqrt{B})$. Moreover, if we look at a larger time interval I, we can decompose this into intervals of size B for which we have constraints in the LP, and a prefix and suffix of size at most B. Therefore, the backlog is constrained to be 0 by the LP for all intermediate intervals except the prefix and suffix which can have a backlog of $O(\sqrt{B})$. This will immediately give us the $O(\log^{1.5} n)$ approximation of [2].

Indeed, the main tool which lets us achieve this comes from a recent rounding technique of Lovett and Meka [7]. They prove the following result which they used as a subroutine for minimizing discrepancy of set systems, but it turns out to be a general result applicable in our setting as well [2].

Theorem 2 (Constructive partial coloring theorem [7]) *Let* $\mathbf{y} \in [0,1]^m$ *be any starting point,* $\delta > 0$ *be an arbitrary error parameter,* $v_1, \ldots, v_n \in \mathbf{R}^n$ *vectors, and* $\lambda_1, \ldots, \lambda_n \geq 0$ *parameters with*

$$\sum_{i=1}^{n} e^{-\lambda_i^2/16} \leq \frac{m}{16}. \tag{6}$$

Then, there is a randomized $\tilde{O}((m+n)^3/\delta^2)$*-time algorithm to compute a vector* $\mathbf{z} \in [0,1]^m$ *with*

(i) $z_j \in [0, \delta] \cup [1-\delta, 1]$ *for at least* $m/32$ *of the indices* $j \in [m]$.
(ii) $|v_i, \cdot \mathbf{z} - v_i \cdot \mathbf{y}| \leq \lambda_i \|v_i\|_2$, *for each* $i \in [n]$.

Hardness Results

The authors [2] also complement the above algorithmic result with the following negative results.

Theorem 3 *The natural LP relaxation for the broadcast problem has an integrality gap of* $\Omega(\log n)$.

Interestingly, Theorem 3 is based on establishing a new connection with the problem of minimizing the discrepancy of 3 permutations. In the 3-permutation problem, we are given 3 permutations π_1, π_2, π_3 of $[n]$. The goal of the problem is to find a coloring χ that minimizes the discrepancy. The discrepancy of $\Pi = (\pi_1, \pi_2, \pi_3)$ w.r.t a ± 1 coloring χ is the worst case discrepancy of all prefixes. That is, $\max_{i=1}^{3} \max_{k=1}^{n} \left| \sum_{j=1}^{k} \chi(\pi_{i,j}) \right|$, where $\pi_{i,j}$ is the jth element in π_i. Newman and Nikolov [8] showed a tight $\Omega(\log n)$ lower bound on the discrepancy of 3 permutations, resolving a long-standing conjecture. The authors [2] note that this can be used to give an integrality gap for the broadcast scheduling problem as well. Then, by generalizing the connection to the discrepancy of ℓ permutations, [2] shows the following hardness results (prior to this, only NP hardness was known).

Theorem 4 *There is no* $O(\log^{1/2-\epsilon} n)$ *approximation algorithm for the problem of minimizing average response time, for any* $\epsilon > 0$, *unless* $\mathsf{NP} \subseteq \cup_{t>0} \mathsf{BPTIME}(2^{\log^t n})$. *Moreover, for any sufficiently large* ℓ, *there is no* $O(\ell^{1/2})$ *approximation algorithm for the* ℓ*-permutation problem, unless* $\mathsf{NP} = \mathsf{RP}$.

Online Broadcast Scheduling

The broadcast scheduling problem has also been studied in the *online scheduling model*, where the algorithm is made aware of requests *only when they arrive*, and it has to make the broadcast choices without knowledge of the future requests. Naturally, the performance of our algorithms degrade when compared to the offline model, but remarkably, we can get nontrivial algorithms even in the online model! The only additional assumption we need in the online model is that our scheduling algorithm may broadcast two pages (instead of one) every $1/\epsilon$ time slots, in order to get approximation ratios that depend on $1/\epsilon$. In particular, several $(1+\epsilon)$-speed, $O(\text{poly}(1/\epsilon))$ are now known [4, 6], and it is also known that extra speed is necessary to obtain $n^{o(1)}$ competitive algorithms.

Recommended Reading

1. Bansal N, Charikar M, Khanna S, Naor J (2005) Approximating the average response time in broadcast scheduling. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms, Vancouver
2. Bansal N, Charikar M, Krishnaswamy R, Li S (2014) Better algorithms and hardness for broadcast scheduling via a discrepancy approach. In: Proceedings of the 25th annual ACM-SIAM symposium on discrete algorithms, Portland
3. Bansal N, Coppersmith D, Sviridenko M (2008) Improved approximation algorithms for broadcast scheduling. SIAM J Comput 38(3): 1157–1174
4. Bansal N, Krishnaswamy R, Nagarajan V (2010) Better scalable algorithms for broadcast scheduling. In: Automata, languages and programming (ICALP), Bordeaux, pp 324–335
5. Erlebach T, Hall A (2002) NP-hardness of broadcast scheduling and inapproximability of singlesource unsplittable min-cost flow. In: Proceedings of the 13th ACM-SIAM symposium on discrete algorithms, San Francisco, pp 194–202
6. Im S, Moseley B (2012) An online scalable algorithm for average flow time in broadcast scheduling. ACM Trans Algorithms (TALG) 8(4):39
7. Lovett S, Meka R (2012) Constructive discrepancy minimization by walking on the edges. In: 53rd annual IEEE symposium on foundations of computer science (FOCS), New Brunswick, pp 61–67
8. Newman A, Neiman O, Nikolov A (2012) Beck's three permutations conjecture: a counterexample and some consequences. In: 2012 IEEE 53rd annual symposium on foundations of computer science (FOCS), New Brunswick. IEEE, APA

Broadcasting in Geometric Radio Networks

Andrzej Pelc
Department of Computer Science, University of Québec-Ottawa, Gatineau, QC, Canada

Keywords

Broadcasting; Collision detection; Deterministic algorithm; Geometric; Knowledge radius; Radio network

Synonyms

Wireless information dissemination in geometric networks

Years and Authors of Summarized Original Work

2001; Dessmark, Pelc

Problem Definition

The Model Overview

Consider a set of stations (nodes) modeled as points in the plane, labeled by natural numbers, and equipped with transmitting and receiving capabilities. Every node u has a *range* r_u depending on the power of its transmitter, and it can reach all nodes at distance at most r_u from it. The collection of nodes equipped with ranges determines a directed graph on the set of nodes, called a *geometric radio network* (GRN), in which a directed edge (uv) exists if node v can be reached from u. In this case u is called a *neighbor* of v. If the power of all transmitters is the same, then all ranges are equal and the corresponding GRN is symmetric.

Research partially supported by NSERC discovery grant and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

Nodes send messages in synchronous *rounds*. In every round, every node acts either as a *transmitter* or as a *receiver*. A node gets a message in a given round, if and only if it acts as a receiver and exactly one of its neighbors transmits in this round. The message received in this case is the one that was transmitted. If at least two neighbors of a receiving node u transmit simultaneously in a given round, none of the messages is received by u in this round. In this case, it is said that a *collision* occurred at u.

The Problem

Broadcasting is one of the fundamental network communication primitives. One node of the network, called the *source*, has to transmit a message to all other nodes. Remote nodes are informed via intermediate nodes, along directed paths in the network. One of the basic performance measures of a broadcasting scheme is the total time, i.e., the number of rounds it uses to inform all the nodes of the network.

For a fixed real $s \geq 0$, called the *knowledge radius*, it is assumed that each node knows the part of the network within the circle of radius s centered at it, i.e., it knows the positions, labels, and ranges of all nodes at distance at most s. The following problem is considered:

How does the size of the knowledge radius influence deterministic broadcasting time in GRN?

Terminology and Notation

Fix a finite set $R = \{r_1, \ldots, r_\rho\}$ of positive reals such that $r_1 < \cdots < r_\rho$. Reals r_i are called *ranges*. A *node* v is a triple $[l, (x, y), r_i]$, where l is a binary sequence called the *label* of v; (x, y) are coordinates of a point in the plane, called the *position* of v; and $r_i \in R$ is called the *range* of v. It is assumed that labels are consecutive integers 1 to n, where n is the number of nodes, but all the results hold if labels are integers in the set $\{1, \ldots, M\}$, where $M \in O(n)$. Moreover, it is assumed that all nodes know an upper bound Γ on n, where Γ is polynomial in n. One of the nodes is distinguished and called the *source*. Any set of nodes C with a distinguished source,

such that positions and labels of distinct nodes are different, is called a *configuration*.

With any configuration C, the following directed graph $\mathcal{G}(C)$ is associated. Nodes of the graph are nodes of the configuration and a directed edge (uv) exists in the graph, if and only if the distance between u and v does not exceed the range of u. (The word "distance" always means the geometric distance in the plane and not the distance in a graph.) In this case u is called a neighbor of v. Graphs of the form $\mathcal{G}(C)$ for some configuration C are called *geometric radio networks* (GRN). In what follows, only configurations C such that in $\mathcal{G}(C)$ there exists a directed path from the source to any other node are considered. If the size of the set R of ranges is ρ, a resulting configuration and the corresponding GRN are called a ρ-configuration and ρ-GRN, respectively. Clearly, all 1-GRN are symmetric graphs. D denotes the *eccentricity* of the source in a GRN, i.e., the maximum length of all shortest paths in this graph from the source to all other nodes. D is of order of the diameter if the graph is symmetric but may be much smaller in general. $\Omega(D)$ is an obvious lower bound on broadcasting time.

Given any configuration, fix a nonnegative real s, called the *knowledge radius*, and assume that every node of C has initial input consisting of all nodes whose positions are at distance at most s from its own. Thus, it is assumed that every node knows a priori labels, positions, and ranges of all nodes within a circle of radius s centered at it. All nodes also know the set R of available ranges.

It is not assumed that nodes know any global parameters of the network, such as its size or diameter. The only global information that nodes have about the network is a polynomial upper bound on its size. Consequently, the broadcast process may be finished but no node needs to be aware of this fact. Hence, the adopted definition of broadcasting time is the same as in [3]. An algorithm accomplishes broadcasting in t rounds, if all nodes know the source message after round t, and no messages are sent after round t.

Only deterministic algorithms are considered. Nodes can transmit messages even before getting the source message, which enables preprocessing in some cases. The algorithms are *adaptive*, i.e., nodes can schedule their actions based on their local history. A node can obviously gain knowledge from previously obtained messages. There is, however, another potential way of acquiring information during the communication process. The availability of this method depends on what happens during a collision, i.e., when u acts as a receiver and two or more neighbors of u transmit simultaneously. As mentioned above, u does not get any of the messages in this case. However, two scenarios are possible. Node u may either hear nothing (except for the background noise), or it may receive *interference noise* different from any message received properly but also different from background noise. In the first case, it is said that there is no *collision detection*, and in the second case – that collision detection is available (cf., e.g., [1]). A discussion justifying both scenarios can be found in [1, 7].

Related Work

Broadcasting in geometric radio networks and some of their variations was considered, e.g., in [6, 8, 9, 11, 12]. In [12] the authors proved that scheduling optimal broadcasting is NP hard even when restricted to such graphs and gave an $O(n \log n)$ algorithm to schedule an optimal broadcast when nodes are situated on a line. In [11] broadcasting was considered in networks with nodes randomly placed on a line. In [9] the authors discussed fault-tolerant broadcasting in radio networks arising from regular locations of nodes on the line and in the plane, with reachability regions being squares and hexagons, rather than circles. Finally, in [6] broadcasting with restricted knowledge was considered but the authors studied only the special case of nodes situated on the line.

Key Results

The results summarized below are based on the paper [5], of which [4] is a preliminary version.

Arbitrary GRN in the Model Without Collision Detection

Clearly all upper bounds and algorithms are valid in the model with collision detection as well.

Large Knowledge Radius

Theorem 1 *The minimum time to perform broadcasting in an arbitrary GRN with source eccentricity D and knowledge radius $s > r_\rho$ (or with global knowledge of the network) is $\Theta(D)$.*

This result yields a centralized $O(D)$ broadcasting algorithm when global knowledge of the GRN is available. This is in sharp contrast with broadcasting in arbitrary graphs, as witnessed by the graph from [10] which has bounded diameter but requires time $\Omega(\log n)$ for broadcasting.

Knowledge Radius Zero

Next consider the case when knowledge radius $s = 0$, i.e., when every node knows only its own label, position, and range. In this case, it is possible to broadcast in time $O(n)$ for arbitrary GRN. It should be stressed that this upper bound is valid for arbitrary GRN, not only symmetric, unlike the algorithm from [3] designed for arbitrary symmetric graphs.

Theorem 2 *It is possible to broadcast in arbitrary n-node GRN with knowledge radius zero in time $O(n)$.*

The above upper bound for GRN should be contrasted with the lower bound from [2,3] showing that some graphs require broadcasting time $\Omega(n \log n)$. Indeed, the graphs constructed in [2, 3] and witnessing to this lower bound are not GRN. Surprisingly, this sharper lower bound does not require very unusual graphs. While counterexamples from [2, 3] are not GRN, it turns out that the reason for a longer broadcasting time is really not the topology of the graph but the difference in knowledge available to nodes. Recall that in GRN with knowledge radius 0, it is assumed that each node knows its own position (apart from its label and range): the upper bound $O(n)$ uses this geometric information extensively.

If this knowledge is not available to nodes (i.e., each node knows only its label and range), then there exists a family of GRN requiring broadcasting time $\Omega(n \log n)$. Moreover, it is possible to show such GRN resulting from configurations with only 2 distinct ranges. (Obviously for 1 configurations, this lower bound does not hold, as these configurations yield symmetric GRN, and in [3], the authors showed an $O(n)$ algorithm working for arbitrary symmetric graphs).

Theorem 3 *If every node knows only its own label and range (and does not know its position), then there exist n-node GRN requiring broadcasting time $\Omega(n \log n)$.*

Symmetric GRN

The Model with Collision Detection

In the model with collision detection and knowledge radius zero, optimal broadcast time is established by the following pair of results.

Theorem 4 *In the model with collision detection and knowledge radius zero, it is possible to broadcast in any n-node symmetric GRN of diameter D in time $O(D + \log n)$.*

The next result is the lower bound $\Omega(\log n)$ for broadcasting time, holding for some GRN of diameter 2. Together with the obvious bound, $\Omega(D)$ this matches the upper bound from Theorem 4.

Theorem 5 *For any broadcasting algorithm with collision detection and knowledge radius zero, there exist n-node symmetric GRN of diameter 2 for which this algorithm requires time $\Omega(\log n)$.*

The Model Without Collision Detection

For the model without collision detection, it is possible to maintain complexity $O(D + \log n)$ of broadcasting. However, we need a stronger assumption concerning knowledge radius: it is no longer 0, but positive, although arbitrarily small.

Theorem 6 *In the model without collision detection, it is possible to broadcast in any n-node symmetric GRN of diameter D in time $O(D + \log n)$, for any positive knowledge radius.*

Applications

The radio network model is applicable to wireless networks using a single frequency. The specific model of geometric radio networks described

in section "Problem Definition" is applicable to wireless networks where stations are located in a relatively flat region without large obstacles (natural or human made), e.g., in the sea or a desert, as opposed to a large city or a mountain region. In such a terrain, the signal of a transmitter reaches receivers at the same distance in all directions, i.e., the set of potential receivers of a transmitter is a disc.

Open Problems

1. Is it possible to broadcast in time $o(n)$ in arbitrary n-node GRN with eccentricity D sublinear in n for knowledge radius zero?
   Note: In view of Theorem 2, it is possible to broadcast in time $O(n)$.
2. Is it possible to broadcast in time $O(D+\log n)$ in all symmetric n-node GRN with eccentricity D, without collision detection, when knowledge radius is zero?
   Note: In view of Theorems 4 and 6, the answer is positive if either collision detection or a positive (even arbitrarily small) knowledge radius is assumed.

Cross-References

- ▸ Deterministic Broadcasting in Radio Networks
- ▸ Randomized Broadcasting in Radio Networks
- ▸ Randomized Gossiping in Radio Networks
- ▸ Routing in Geometric Networks

Recommended Reading

1. Bar-Yehuda R, Goldreich O, Itai A (1992) On the time complexity of broadcast in radio networks: an exponential gap between determinism and randomization. J Comput Syst Sci 45:104–126
2. Bruschi D, Del Pinto M (1997) Lower bounds for the broadcast problem in mobile radio networks. Distrib Comput 10:129–135
3. Chlebus BS, Gasieniec L, Gibbons A, Pelc A, Rytter W (2002) Deterministic broadcasting in ad hoc radio networks. Distrib Comput 15:27–38
4. Dessmark A, Pelc A (2001) Tradeoffs between knowledge and time of communication in geometric radio networks. In: Proceedings of 13th annual ACM symposium on parallel algorithms and architectures (SPAA 2001), Heraklion, pp 59–66
5. Dessmark A, Pelc A (2007) Broadcasting in geometric radio networks. J Discret Algorithms 5:187–201
6. Diks K, Kranakis E, Krizanc D, Pelc A (2002) The impact of knowledge on broadcasting time in linear radio networks. Theor Comput Sci 287:449–471
7. Gallager R (1985) A perspective on multiaccess channels. IEEE Trans Inf Theory 31:124–142
8. Gasieniec L, Kowalski DR, Lingas A, Wahlen M (2008) Efficient broadcasting in known geometric radio networks with non-uniform ranges. In: Proceedings of 22nd international symposium on distributed computing (DISC 2008), Arcachon, pp 274–288
9. Kranakis E, Krizanc D, Pelc A (2001) Fault-tolerant broadcasting in radio networks. J Algorithms 39:47–67
10. Pelc A, Peleg D (2007) Feasibility and complexity of broadcasting with random transmission failures. Theor Comput Sci 370:279–292
11. Ravishankar K, Singh S (1994) Broadcasting on $[0, L]$. Discret Appl Math 53:299–319
12. Sen A, Huson ML (1996) A new model for scheduling packet radio networks. In: Proceedings of 15th annual joint conference of the IEEE computer and communication societies (IEEE INFOCOM'96), San Francisco, pp 1116–1124

B-trees

Jan Vahrenhold
Department of Computer Science, Westfälische Wilhelms-Universität Münster, Münster, Germany

Keywords

Data structures; Dictionary; External memory; Indexing; Searching

Years and Authors of Summarized Original Work

1972; Bayer, McCreight

Problem Definition

This problem is concerned with storing a linearly ordered set of elements such that the DICTIO-

NARY operations FIND, INSERT, and DELETE can be performed efficiently.

In 1972, Bayer and McCreight introduced the class of B-trees as a possible way of implementing an "index for a dynamically changing random access file" [7, p. 173]. B-trees have received considerable attention both in the database and in the algorithms community ever since; a prominent witness to their immediate and widespread acceptance is the fact that the authoritative survey on B-trees authored by Comer [10] appeared as soon as 1979 and, already at that time, referred to the B-tree data structure as the "ubiquitous B-tree."

Notations

A *B-tree* is a multiway search tree defined as follows (the definition of Bayer and McCreight [7] is restated according to Knuth [19, Sec. 6.2.4] and Cormen et al. [11, Ch. 18.1]):

Definition 1 Let $m \geq 3$ be a positive integer. A tree T is a *B-tree* of degree m if it is either empty or fulfills the following properties:

1. All leaves of T appear on the same level of T.
2. Every node of T has at most m children.
3. Every node of T, except for the root and the leaves, has at least $m/2$ children.
4. The root of T is either a leaf or has at least two children.
5. An internal node with k children $c_1[\nu], \ldots, c_k[\nu]$ stores $k-1$ keys, and a leaf stores between $m/2-1$ and $m-1$ keys. The keys $\text{key}_i[\nu]$, $1 \leq i \leq k-1$, of a node $\nu \in T$ are maintained in sorted order, i.e., $\text{key}_1[\nu] \leq \ldots \leq \text{key}_{k-1}[\nu]$.
6. If ν is an internal node of T with k children $c_1[\nu], \ldots, c_k[\nu]$, the $k-1$ keys $\text{key}_1[\nu], \ldots, \text{key}_{k-1}[\nu]$ of ν separate the range of keys stored in the subtrees rooted at the children of ν. If x_i is any key stored in the subtree rooted at $c_i[\nu]$, the following holds:

$$x_1 \leq \text{key}_1[\nu] \leq x_2 \leq \text{key}_2[\nu] \leq \cdots \leq x_{k-1} \leq \text{key}_{k-1}[\nu] \leq x_k$$

To search a B-tree for a given key x, the algorithm starts with the root of the tree being the current node. If x matches one of the current node's keys, the search terminates successfully. Otherwise, if the current node is a leaf, the search terminates unsuccessfully. If the current node's key does not contain x and if the current node is not a leaf, the algorithm identifies the unique subtree rooted at the child of the current node that may contain x and recurses on this subtree. Since the keys of a node guide the search process, they are also referred to as *routing elements*.

Variants and Extensions

Knuth [19] defines a *B^*-tree* to be a B-tree where Property 3 in Definition 1 is modified such that every node (except for the root) contains at least $2m/3$ keys.

A *B^+-tree* is a leaf-oriented B-tree, i.e., a B-tree that stores the keys in the leaves only. Additionally, the leaves are linked in left-to-right order to allow for fast sequential traversal of the keys stored in the tree. In a leaf-oriented tree, the routing elements usually are copies of certain keys stored in the leaves ($\text{key}_i[\nu]$ can be set to be the largest key stored in the subtree rooted at $c_i[\nu]$), but any set of routing elements that fulfills Properties 5 and 6 of Definition 1 can do as well.

Huddleston and Mehlhorn [16] extended Definition 1 to describe a more general class of multiway search trees that includes the class of B-trees as a special case. Their class of so-called (a, b)-trees is parameterized by two integers a and b with $a \geq 2$ and $2a - 1 \leq b$. Property 2 of Definition 1 is modified to allow each node to have up to b children, and Property 3 is modified to require that, except for the root and the leaves, every node of an (a, b)-tree has at least a children. All other properties of Definition 1 remain unchanged for (a, b)-trees. Usually, (a, b)-trees are implemented as leaf-oriented trees.

By the above definitions, a B-tree is a $(b/2, b)$-tree (if b is even) or an $(a, 2a-1)$-tree (if b is odd). The subtle difference between even and odd maximum degree becomes relevant in an important amortization argument of Huddleston and Mehlhorn (see below) where the inequality

$b \geq 2a$ is required to hold. This amortization argument actually caused (a, b)-trees with $b \geq 2a$ to be given a special name: *weak* B-trees [16].

Update Operations

An INSERT operation on an (a, b)-tree first tries to locate the key x to be inserted. After an unsuccessful search that stops at some leaf ℓ, x is inserted into ℓ's set of keys. If ℓ becomes too full, i.e., contains more than b elements, two approaches are possible to resolve this *overflow* situation: (1) the node ℓ can be split around its median key into two nodes with at least a keys each, or (2) the node ℓ can have some of its keys be distributed to its left or right siblings (if this sibling has enough space to accommodate the new keys). In the first case, a new routing element separating the keys in the two new subtrees of ℓ's parent μ has to be inserted into the key set of μ, and in the second case, the routing element in μ separating the keys in the subtree rooted at ℓ from the keys rooted at ℓ's relevant sibling needs to be updated. If ℓ was split, the node μ needs to be checked for a potential overflow due to the insertion of a new routing element, and the split may propagate all the way up to the root.

A DELETE operation also first locates the key x to be deleted. If (in a non-leaf-oriented tree) x resides in an internal node, x is replaced by the largest key in the left subtree of x (or the smallest key in the right subtree of x) which resides in a leaf and is deleted from there. In a leaf-oriented tree, keys are deleted from leaves only (the correctness of a routing element on a higher levels is not affected by this deletion). In any case, a DELETE operation may result in a leaf node ℓ containing less than a elements. Again, there are two approaches to resolve this *underflow* situation: (1) the node ℓ is merged with its left or right sibling node or (2) keys from ℓ's left or right sibling node are moved to ℓ (unless the sibling node would underflow as a result of this). Both underflow handling strategies require updating the routing information stored in the parent of ℓ which (in the case of merging) may underflow itself. As with overflow handling, this process may propagate up to the root of the tree.

Note that the root of the tree can be split as a result of an INSERT operation and that it may disappear if the only two children of the root are merged to form the new root. This implies that B-trees grow and shrink at the top, and thus all leaves are guaranteed to appear on the same level of the tree (Property 1 of Definition 1).

Key Results

Since B-trees are a premier index structure for external storage, the results given in this section are stated not only in the RAM-model of computation but also in the I/O-model of computation introduced by Aggarwal and Vitter [2]. In the I/O-model, not only the number N of elements in the problem instance but also the number M of elements that simultaneously can be kept in main memory and the number B of elements that fit into one disk block are (nonconstant) parameters, and the complexity measure is the number of I/O-operations needed to solve a given problem instance. If B-trees are used in an external memory setting, the degree m of the B-tree is usually chosen such that one node fits into one disk block, i.e., $m \in \Theta(B)$, and this is assumed implicitly whenever the I/O-complexity of B-trees is discussed.

Theorem 1 *The height of an N-key B-tree of degree $m \geq 3$ is bounded by* $\log_{\lceil m/2 \rceil}(N+1)/2$.

Theorem 2 ([22]) *The storage utilization for B-trees of high order under random insertions and deletions is approximately* $\ln 2 \approx 69\,\%$.

Theorem 3 *A B-tree may be used to implement the abstract data type* DICTIONARY *such that the operations* FIND, INSERT, *and* DELETE *on a set of N elements from a linearly ordered domain can be performed in $\mathcal{O}(\log N)$ time (with $\mathcal{O}(\log_B N)$ I/O-operations) in the worst case.*

Remark 1 By threading the nodes of a B-tree, i.e., by linking the nodes according to their in-order traversal number, the operations PREV and

NEXT can be performed in constant time (with a constant number of I/O-operations).

A (one-dimensional) *range query* asks for all keys that fall within a given query range (interval).

Lemma 1 *A B-tree supports (one-dimensional) range queries with $\mathcal{O}(\log N + K)$ time complexity ($\mathcal{O}(\log_B N + K/B)$ I/O-complexity) in the worst case where K is the number of keys reported.*

Under the convention that each update to a B-tree results in a new "version" of the B-tree, a *multiversion* B-tree is a B-tree that allows for updates of the current version but also supports queries in earlier versions.

Theorem 4 ([9]) *A multiversion B-tree can be constructed from a B-tree such that it is optimal with respect to the worst-case complexity of the* FIND, INSERT, *and* DELETE *operations as well as to the worst-case complexity of answering range queries.*

Applications

Databases

One of the main reasons for the success of the B-tree lies in its close connection to databases: any implementation of Codd's relational data model (introduced incidentally in the same year as B-trees were invented) requires an efficient indexing mechanism to search and traverse relations that are kept on secondary storage. If this index is realized as a B^+-tree, all keys are stored in a linked list of leaves which is indexed by the top levels of the B^+-tree, and thus both efficient logarithmic searching and sequential scanning of the set of keys is possible.

Due to the importance of this indexing mechanism, a wide number of results on how to incorporate B-trees and their variants into database systems and how to formulate algorithms using these structures have been published in the database community. Comer [10] and Graefe [14] summarize early and recent results, but due to the bulk of results, even these summaries cannot be fully comprehensive. Also, B-trees have been shown to work well in the presence of concurrent operations [8], and Mehlhorn [20, p. 212] notes that they perform especially well if a top-down splitting approach is used. The details of this splitting approach may be found, e.g., in the textbook of Cormen et al. [11, Ch. 18.2].

Priority Queues

A B-tree may be used to serve as an implementation of the abstract data type PRIORITYQUEUE since the smallest key always resides in the first slot of the leftmost leaf.

Lemma 2 *An implementation of a priority queue that uses a B-tree supports the* MIN *operation in $\mathcal{O}(1)$ time (with $\mathcal{O}(1)$ I/O-operations). All other operations (including* DECREASEKEY*) have a time complexity of $\mathcal{O}(\log N)$ (an I/O-complexity of $\mathcal{O}(\log_B N)$) in the worst case.*

Mehlhorn [20, Sec. III, 5.3.1] examined B-trees (and, more general, (a, b)-trees with $a \geq 2$ and $b \geq 2a - 1$) in the context of *mergeable* priority queues. *Mergeable priority queues* are priority queues that additionally allow for concatenating and splitting priority queues. Concatenating priority queues for a set $S_1 \neq \emptyset$ and a set $S_2 \neq \emptyset$ is only defined if $\max\{x \mid x \in S_1\} < \min\{x \mid x \in S_2\}$ and results in a single priority queue for $S_1 \cup S_2$. Splitting a priority queue for a set $S_3 \neq \emptyset$ according to some $y \in \mathrm{dom}(S_3)$ results in a priority queue for the set $S_4 := \{x \in S_3 \mid x \leq y\}$ and a priority queue for the set $S_5 := \{x \in S_3 \mid x > y\}$ (one of these sets may be empty). Mehlhorn's result restated in the context of B-trees is as follows:

Theorem 5 (Theorem 6 in [20, Sec. III, 5.3.1]) *If sets $S_1 \neq \emptyset$ and $S_2 \neq \emptyset$ are represented by a B-tree each, then operation* CONCATENATE(S_1, S_2) *takes time $\mathcal{O}(\log \max\{|S_1|, |S_2|\})$ (has an I/O-complexity of $\mathcal{O}(\log_B \max\{|S_1|, |S_2|\})$) and operation* SPLIT$(S_1, y)$ *takes time $\mathcal{O}(\log |S_1|)$ (has an I/O-complexity of $\mathcal{O}(\log_B |S_1|)$). All bounds hold in the worst case.*

Buffered Data Structures

Many applications (including sorting) that involve massive data sets allow for batched data processing. A variant of B-trees that exploits this relaxed problem setting is the so-called *buffer tree* proposed by Arge [4]. A *buffer tree* is a B-trees of degree $m \in \Theta(M/B)$ (instead of $m \in \Theta(B)$) where each node is assigned a buffer of size $\Theta(M)$. These buffers are used to collect updates and query requests that are passed further down the tree only if the buffer gets full enough to allow for cost amortization.

Theorem 6 (Theorem 1 in [4]) *The total cost of an arbitrary sequence of N intermixed* INSERT *and* DELETE *operations on an initially empty buffer tree is $\mathcal{O}(N/B \log_{M/B} N/B)$ I/O operations, that is, the amortized I/O-cost of an operation is $\mathcal{O}(1/B \log_{M/B} N/B)$.*

As a consequence, N elements can be sorted spending an optimal number of $\mathcal{O}(N/B \log_{M/B} N/B)$ I/O-operations by inserting them into a (leaf-oriented) buffer tree in a batched manner and then traversing the leaves. By the preceding discussion, buffer trees can also be used to implement (batched) priority queues in the external memory setting. Arge [4] extended his analysis of buffer trees to show that they also support DELETEMIN operations with an amortized I/O-cost of $\mathcal{O}(1/B \log_{M/B} N/B)$.

Since the degree of a buffer tree is too large to allow for efficient Single shot, i.e., non-batched operations, Arge et al. [6] discussed how buffers can be attached to (and later detached from) a multiway tree while at the same time keeping the degree of the base structure in $\Theta(B)$. Their discussion uses the R-tree index structure as a running example; the techniques presented, however, carry over to the B-tree. The resulting data structure is accessed through standard methods and additionally allows for batched update operations, e.g., *bulk loading*, and queries. The amortized I/O-complexity of all operations is analogous to the complexity of the buffer tree operations.

Using this buffering technique along with weight balancing [5], Achakeev and Seeger [1] showed how to efficiently bulk load and bulk update partially persistent data structures such as the multiversion B-tree.

Variants of the B-tree base structure that support modern architectures such as many-core processors and that can be updated efficiently have also been proposed by Sewall et al. [21], Graefe et al. [15], and Erb et al. [12].

B-trees as Base Structures

Several external memory data structures are derived from B-trees or use a B-tree as their base structure – see the survey by Arge [3] for a detailed discussion. One of these structures, the so-called *weight-balanced* B-tree is particularly useful as a base tree for building dynamic external data structures that have secondary structures attached to all (or some) of their nodes. The weight-balanced B-tree, developed by Arge and Vitter [5], is a variant of the B-tree that requires all subtrees of a node to have approximately, i.e., up to a small constant factor, the same number of leaves. Weight-balanced B-trees can be shown to have the following property:

Theorem 7 ([5]) *In a weight-balanced B-tree, rebalancing after an update operation is performed by splitting or merging nodes. When a rebalancing operation involves a node v that is the root of a subtree with $w(v)$ leaves, at least $\Theta(w(v))$ update operations involving leaves below v have to be performed before v itself has to be rebalanced again.*

Using the above theorem, amortized bounds for maintaining secondary data structures attached to nodes of the base tree can be obtained – as long as each such structure can be updated with an I/O-complexity linear in the number of elements stored below the node it is attached to [3,5].

Amortized Analysis

Most of the amortization arguments used for (a,b)-trees, buffer trees, and their relatives are based upon a theorem due to Huddleston and Mehlhorn [16, Theorem 3]. This theorem states that the total number of rebalancing operations in any sequence of N intermixed insert and delete

operations performed on an initially empty *weak* B-tree, i.e., an (a, b)-tree with $b \geq 2a$, is at most linear in N. This result carries over to buffer trees since they are $(M/4B, M/B)$-trees. Since B-trees are (a, b)-trees with $b = 2a - 1$ (if b is odd), the result in its full generality is not valid for B-trees, and Huddleston and Mehlhorn present a simple counterexample for $(2, 3)$-trees.

A crucial fact used in the proof of the above amortization argument is that the sequence of operations to be analyzed is performed on an initially *empty* data structure. Jacobsen et al. [17] proved the existence of *non-extreme* (a, b)-trees, i.e., (a, b)-trees where only few nodes have a degree of a or b. Based upon this, they re-established the above result that the rebalancing cost in a sequence of operations is amortized constant (and thus the related result for buffer trees) also for operations on initially nonempty data structures.

In connection with concurrent operations in database systems, it should be noted that the analysis of Huddleston and Mehlhorn actually requires $b \geq 2a + 2$ if a top-down splitting approach is used. In can be shown, though, that even in the general case, few node splits (in an amortized sense) happen close to the root.

URLs to Code and Data Sets

There is a variety of (commercial and free) implementations of B-trees and (a, b)-trees available for download. Representatives are the `C++`-based implementations that are part of the LEDA-library (http://www.algorithmic-solutions.com), the `STXXL`-library (http://stxxl.sourceforge.net), and the TPIE-library (http://www.madalgo.au.dk/tpie/) as well as the `Java`-based implementation that is part of the `javaxxl`-library (http://xxl.googlecode.com). Furthermore, (pseudo-code) implementations can be found in almost every textbook on database systems or on algorithms and data structures – see, e.g., [11, 13]. Since textbooks almost always leave developing the implementation details of the DELETE operation as an exercise to the reader, the discussion by Jannink [18] is especially helpful.

Cross-References

- Cache-Oblivious B-Tree
- I/O-Model
- R-Trees

Recommended Reading

1. Achakeev D, Seeger B (2013) Efficient bulk updates on multiversion B-trees. PVLDB 14(6):1834–1845
2. Aggarwal A, Vitter JS (1988) The input/output complexity of sorting and related problems. Commun ACM 31(9):1116–1127
3. Arge LA (2002) External memory data structures. In: Abello J, Pardalos PM, Resende MGC (eds) Handbook of massive data sets. Kluwer, Dordrecht, pp 313–357
4. Arge LA (2003) The buffer tree: a technique for designing batched external data structures. Algorithmica 37(1):1–24
5. Arge LA, Vitter JS (2003) Optimal external interval management. SIAM J Comput 32(6):1488–1508
6. Arge LA, Hinrichs KH, Vahrenhold J, Vitter JS (2002) Efficient bulk operations on dynamic R-trees. Algorithmica 33(1):104–128
7. Bayer R, McCreight EM (1972) Organization and maintenance of large ordered indexes. Acta Inform 1(3):173–189
8. Bayer R, Schkolnick M (1977) Concurrency of operations on B-trees. Acta Inform 9(1):1–21
9. Becker B, Gschwind S, Ohler T, Seeger B, Widmayer P (1996) An asymptotically optimal multiversion B-tree. VLDB J 5(4):264–275
10. Comer DE (1979) The ubiquitous B-tree. ACM Comput Surv 11(2):121–137
11. Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms. The MIT electrical engineering and computer science series, 3rd edn. MIT, Cambridge
12. Erb S, Kobitzsch M, Sanders P (2014) Parallel bi-objective shortest paths using weight-balanced B-trees with bulk updates. In: Gudmundsson J, Katajainen J (eds) Proceedings of the 13th international symposium on experimental algorithms (SEA 2014). Lecture notes in computer science, vol 8504. Springer, Berlin, pp 111–122
13. Garcia-Molina H, Ullman JD, Widom J (2009) Database systems: the complete book, 2nd edn. Prentice Hall, Upper Saddle River
14. Graefe G (2006) B-tree indexes for high update rates. SIGMOD RECORD 35(1):39–44
15. Graefe G, Kimura H, Kuno H (2012) Foster B-trees. ACM Trans Database Syst 37(3):Article 17, 29 Pages
16. Huddleston S, Mehlhorn K (1982) A new data structure for representing sorted lists. Acta Inform 17(2):157–184

17. Jacobsen L, Larsen KS, Nielsen MN (2002) On the existence of non-extreme (a, b)-trees. Inf Process Lett 84(2):69–73
18. Jannink J (1995) Implementing deletions in B$^+$-trees. SIGMOD RECORD 24(1):33–38
19. Knuth DE (1998) Sorting and searching, the art of computer programming, vol 3, 2nd edn. Addison-Wesley, Reading
20. Mehlhorn K (1984) Data structures and algorithms 1: sorting and searching. EATCS monographs on theoretical computer science, vol 1. Springer, Berlin
21. Sewall J, Chhugani J, Kim C, Satish N, Dubey P (2011) PALM: parallel architecture-friendly latch-free modifications to B+-trees on many-core processors. PVLDB 11(4):795–806
22. Yao ACC (1978) On random 2–3 trees. Acta Inform 9:159–170

Burrows-Wheeler Transform

Paolo Ferragina[1] and Giovanni Manzini[2,3]
[1]Department of Computer Science, University of Pisa, Pisa, Italy
[2]Department of Computer Science, University of Eastern Piedmont, Alessandria, Italy
[3]Department of Science and Technological Innovation, University of Piemonte Orientale, Alessandria, Italy

Keywords

Block-sorting data compression

Years and Authors of Summarized Original Work

1994; Burrows, Wheeler

Problem Definition

The Burrows-Wheeler transform is a technique used for the lossless compression of data. It is the algorithmic core of the tool *bzip2* which has become a standard for the creation and distribution of compressed archives.

Before the introduction of the Burrows-Wheeler transform, the field of lossless data compression was dominated by two approaches (see [2, 21] for comprehensive surveys). The first approach dates back to the pioneering works of Shannon and Huffman, and it is based on the idea of using shorter codewords for the more frequent symbols. This idea has originated the techniques of Huffman and arithmetic coding and, more recently, the PPM (prediction by partial matching) family of compression algorithms. The second approach originated from the works of Lempel and Ziv and is based on the idea of adaptively building a dictionary and representing the input string as a concatenation of dictionary words. The best-known compressors based on this approach form the so-called ZIP-family; they have been the standard for several years and are available on essentially any computing platform (e.g., *gzip*, *zip*, *winzip*, just to cite a few).

The Burrows-Wheeler transform introduced a completely new approach to lossless data compression based on the idea of *transforming* the input to make it easier to compress. In the authors' words: "(this) technique [...] works by applying a reversible transformation to a block of text to make redundancy in the input more accessible to simple coding schemes" [5, Sect. 7]. Not only has this technique produced some state-of-the-art compressors, but it also originated the field of compressed indexes [20] and it has been successfully extended to compress (and index) structured data such as XML files [11] and tables [22].

Key Results

Notation

Let s be a string of length n drawn from an alphabet Σ. For $i = 0,\ldots, n-1$, $s[i]$ denotes the i-th character of s and $s[i, n-1]$ denotes the suffix of s starting at position i (i.e., starting with the character $s[i]$). Given two strings s and t, the notation $s \prec t$ is used to denote that s lexicographically precedes t.

The Burrows-Wheeler Transform

In [5] Burrows and Wheeler introduced a new compression algorithm based on a reversible

transformation, now called the *Burrows-Wheeler transform* (*bwt*). Given a string s, the computation of $bwt(s)$ consists of three basic steps (see Fig. 1):

1. Append to the end of s a special symbol \$ smaller than any other symbol in Σ.
2. Form a *conceptual* matrix $\mathcal{M}$ whose rows are the cyclic shifts of the string $s\$$ sorted in lexicographic order.
3. Construct the transformed text $\hat{s} = bwt(s)$ by taking the last column of $\mathcal{M}$.

Notice that every column of $\mathcal{M}$, hence also the transformed text $\hat{s}$, is a permutation of $s\$$. As an example F, the first column of the *bwt* matrix $\mathcal{M}$ consists of all characters of s alphabetically sorted. In Fig. 1 it is $F = \$iiiimppssss$.

Although it is not obvious from its definition, the *bwt* is an invertible transformation, and both the *bwt* and its inverse can be computed in $O(n)$ optimal time. To be consistent with the more recent literature, the following notation and proof techniques will be slightly different from the ones in [5].

Definition 1 For $1 \leq i \leq n$, let $s[k_i, n-1]$ denote the suffix of s prefixing row i of $\mathcal{M}$, and define $\Psi(i)$ as the index of the row prefixed by $s[k_i + 1, n-1]$.

For example, in Fig. 1 it is $\Psi(2) = 7$ since row 2 of $\mathcal{M}$ is prefixed by *ippi* and row 7 is prefixed by *ppi*. Note that $\Psi(i)$ is not defined for $i = 0$ since row 0 is not prefixed by a proper suffix of s. (In [5] instead of Ψ the authors make use of a map which is essentially the inverse of Ψ. The use of Ψ has been introduced in the literature of compressed indexes where Ψ and its inverse play an important role (see [20]).)

Lemma 1 *For* $i = 1, \ldots, n$, *it is* $F[i] = \hat{s}[\Psi(i)]$.

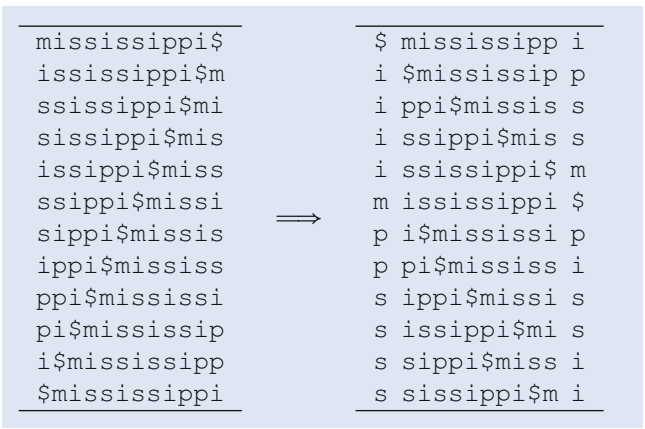
Proof Since each row contains a cyclic shift of $s\$$, the last character of the row prefixed by $s[k_i + 1, n-1]$ is $s[k_i]$. Definition 1 then implies $\hat{s}[\Psi(i)] = s[k_i] = F[i]$ as claimed. □

Lemma 2 *If* $1 \leq i < j \leq n$ *and* $F[i] = F[j]$, *then* $\Psi(i) < \Psi(j)$.

Proof Let $s[k_i, n-1]$ (resp. $s[k_j, n-1]$) denote the suffix of s prefixing row i (resp. row j). The hypothesis $i < j$ implies that $s[k_i, n-1] \prec s[k_j, n-1]$. The hypothesis $F[i] = F[j]$ implies $s[k_i] = s[k_j]$; hence, it must be $s[k_i + 1, n-1] \prec s[k_j + 1, n-1]$. The thesis follows since by construction $\Psi(i)$ (resp. $\Psi(j)$) is the lexicographic position of the row prefixed by $s[k_i + 1, n-1]$ (resp. $s[k_j + 1, n-1]$). □

Lemma 3 *For any character* $c \in \Sigma$, *if* $F[j]$ *is the* ℓ*-th occurrence of* c *in* F, *then* $\hat{s}[\Psi(j)]$ *is the* ℓ*-th occurrence of* c *in* $\hat{s}$.

Proof Take an index h such that $h < j$ and $F[h] = F[j] = c$(the case $h > j$ is symmetric).



Burrows-Wheeler Transform, Fig. 1 Example of Burrows-Wheeler transform for the string $s = mississippi$. The matrix on the right has the rows sorted in lexicographic order. The output of the *bwt* is the last column of the sorted matrix; in this example, the output is $\hat{s} = bwt(s) = ipssm\$pissii$

Lemma 2 implies $\Psi(h) < \Psi(j)$ and Lemma 1 implies $\hat{s}[\Psi(h)] = \hat{s}[\Psi(j)] = c$. Consequently, the number of c's preceding (resp. following) $F[j]$ in F coincides with the number of c's preceding (resp. following) $\hat{s}[\Psi(j)]$ in $\hat{s}$ and the lemma follows. □

In Fig. 1 it is $\Psi(2) = 7$ and both $F[2]$ and $\hat{s}[7]$ are the second i in their respective strings. This property is usually expressed by saying that corresponding characters maintain the *same relative order* in both strings F and $\hat{s}$.

Lemma 4 *For any i, $\Psi(i)$ can be computed from $\hat{s} = bwt(s)$.*

Proof Retrieve F simply by sorting alphabetically the symbols of $\hat{s}$. Then compute $\Psi(i)$ as follows: (1) set $c = F(i)$, (2) compute ℓ such that $F[i]$ is the ℓ-th occurrence of c in F, and (3) return the index of the ℓ-th occurrence of c in $\hat{s}$. □

Referring again to Fig. 1, to compute $\Psi(10)$ it suffices to set $c = F[10] = s$ and observe that $F[10]$ is the second s in F. Then it suffices to locate the index j of the second s in $\hat{s}$, namely, $j = 4$. Hence, $\Psi(10) = 4$, and in fact row 10 is prefixed by *sissippi* and row 4 is prefixed by *issippi*.

Theorem 1 *The original string s can be recovered from bwt(s).*

Proof Lemma 4 implies that the column F and the map Ψ can be retrieved from $bwt(s)$. Let j_0 denote the index of the special character \$ in $\hat{s}$. By construction, the row j_0 of the *bwt* matrix is prefixed by $s[0,n-1]$; hence, $s[0] = F[j_0]$. Let $j_1 = \Psi(j_0)$. By Definition 1 row j_1 is prefixed by $s[1, n-1]$; hence, $s[1] = F[j_1]$. Continuing in this way, it is straightforward to prove by induction that $s[i] = F[\Psi^i(j_0)]$, for $i = 1,\ldots, n-1$. □

Algorithmic Issues

A remarkable property of the *bwt* is that both the direct and the inverse transform admit efficient algorithms that are extremely simple and elegant.

Theorem 2 *Let s[1,n] be a string over a constant size alphabet Σ. String $\hat{s} = bwt(s)$ can be computed in $O(n)$ time using $O(n\log n)$ bits of working space.*

Proof The suffix array of s can be computed in $O(n)$ time and $O(n\log n)$ bits of working space by using, for example, the algorithm in [17]. The suffix array is an array of integers $sa[1, n]$ such that for $i = 1, \ldots, n, s[sa[i], n-1]$ is the i-th suffix of s in the lexicographic order. Since each row of $\mathcal{M}$ is prefixed by a unique suffix of s followed by the special symbol \$, the suffix array provides the ordering of the rows in $\mathcal{M}$. Consequently, $bwt(s)$ can be computed from sa in linear time using the procedure *sa2bwt* of Fig. 2. □

Theorem 3 *Let s[1,n] be a string over a constant size alphabet Σ. Given bwt(s), the string s can be retrieved in $O(n)$ time using $O(n\log n)$ bits of working space.*

Proof The algorithm for retrieving s follows almost verbatim the procedure outlined in the proof of Theorem 1. The only difference is that, for efficiency reasons, all the values of the map Ψ are computed in one shot. This is done by the procedure *bwt2psi* in Fig. 2. In *bwt2psi* instead of working with the column F, it uses the array *count* which is a "compact" representation of F. At the beginning of the procedure, for any character $c \in \Sigma$, $count[c]$ provides the index of the first row of $\mathcal{M}$ prefixed by c. For example, in Fig. 1 $count[i] = 1$, $count[m] = 5$, and so on. In the main *for* loop of *bwt2psi,* the array *bwt* is scanned and *count*[c] is increased every time an occurrence of character c is encountered (line 6). Line 6 also assigns to h the index of the ℓ-th occurrence of c in F. By Lemma 3, line 7 stores correctly in $psi[h]$ the value $i = \Psi(h)$. After the computation of array *psi*, s is retrieved by using the procedure *psi2text* of Fig. 2, whose correctness immediately follows from Theorem 1. Clearly, the procedures *bwt2psi* and *psi2text* in Fig. 2 run in $O(n)$ time. Their working space is dominated by the cost of storing the array *psi* which takes $O(n\log n)$ bits. □

Procedure sa2bwt

```
1. bwt[0]=s[n-1];
2. for(i=1;i<=n;i++)
3. if(sa[i] == 1)
4. bwt[i]='$';
5. else
6. bwt[i]=s[sa[i]-1];
```

Procedure bwt2psi

```
71. for(i=0;i<=n;i++)
2.  c = bwt[i];
3.  if(c == '$')
4.  j0 = i;
5.  else
6.  h = count[c]++;
7.  psi[h]=i;
```

Procedure psi2text

```
891. k = j0; i=0;
2.   do
3.   k = psi[k];
4.   s[i++] = bwt[k];
     while(i<n);
```

Burrows-Wheeler Transform, Fig. 2 Algorithms for computing and inverting the Burrows-Wheeler transform. Procedure *sa2bwt* computes $bwt(s)$ given s and its suffix array *sa*. Procedure *bwt2psi* takes $bwt(s)$ as input and computes the Ψ map storing it in the array *psi*. *bwt2psi* also stores in j_0 the index of the row prefixed by $s[0, n-1]$. *bwt2psi* uses the auxiliary array *count* $[1, |\Sigma|]$ which initially contains in *count[i]* the number of occurrences in $bwt(s)$ of the symbols $1,\ldots,i-1$. Finally, procedure *psi2text* recovers the string s given $bwt(s)$, the array *psi*, and the value j_0

The Burrows-Wheeler Compression Algorithm

The rationale for using the *bwt* for data compression is the following. Consider a string w that appears k times within s. In the *bwt* matrix of s, there will be k consecutive rows prefixed by w, say rows $r_w + 1, r_w + 2, \ldots, r_w + k$. Hence, the positions $r_w + 1, \ldots, r_w + k$ of $\hat{s} = bwt(s)s$ will contain precisely the symbols that immediately precede w in s. If in s certain patterns are more frequent than others, then for many substrings w, the corresponding positions $r_w + 1,\ldots, r_w + k$ of $\hat{s}$ will contain only a few distinct symbols. For example, if s is an English text and w is the string *his*, the corresponding portion of $\hat{s}$ will likely contain many *t*'s and blanks and only a few other symbols. Hence, $\hat{s}$ is a permutation of s that is usually *locally homogeneous*, in that its "short" substrings usually contain only a few distinct symbols. (Obviously this is true only if s has some regularity: if s is a random string $\hat{s}$ will be random as well!)

To take advantage of this property, Burrows and Wheeler proposed to process the string $\hat{s}$ using move-to-front encoding [4] (*mtf*). *mtf* encodes each symbol with the number of distinct symbols encountered since its previous occurrence. To this end, *mtf* maintains a list of the symbols ordered by recency of occurrence; when the next symbol arrives, the encoder outputs its current rank and moves it to the front of the list. Note that *mtf* produces a string which has the same length as $\hat{s}$ and, if $\hat{s}$ is locally homogeneous, the string $mtf(\hat{s})$ will mainly consist of small integers. (If s is an English text, $mtf(\hat{s})$ usually contains more that 50 % zeroes.) Given this skewed distribution, $mtf(\hat{s})$can be easily compressed: Burrows and Wheeler proposed to compress it using Huffman or Arithmetic coding, possibly preceded by the run-length encoding of runs of equal integers.

Burrows and Wheeler were mainly interested in proposing an algorithm with good practical performance. Indeed their simple implementation outperformed, in terms of compression ratio, the tool *gzip* that was the current standard for lossless compression. A few years after the introduction of the *bwt*, [14, 18] have shown that the compression ratio of the Burrows-Wheeler compression algorithm can be bounded in terms of the k-th order empirical entropy of the input string for any $k \geq 0$. For example, Kaplan et al. [14] showed that for any input string s and real $\mu > 1$, the length of the compressed string is bounded by $\mu n H_k(s) + n\log(\zeta(\mu)) + \mu g_k + O(\log n)$ bits, where $\zeta(\mu)$ is the standard Zeta function and g_k is a function depending only on k and the size of Σ. This bound holds *pointwise* for *any* string s, *simultaneously* for any $k \geq 0$ and $\mu > 1$, and it is remarkable since similar bounds have not been proven for any other known compressor. The theoretical study on the performance of *bwt*-based compressors is an active area of research. For more recent results, see [6, 12].

Applications

After the seminal paper of Burrows and Wheeler, many researchers have proposed compression algorithms based on the *bwt* (see [8, 9] and references therein). Of particular theoretical interest are the results in [10] showing that the *bwt* can be used to design a "compression booster," that is, a tool for improving the performance of other compressors in a well-defined and measurable way.

Today the main area of application of the *bwt* is the design of Compressed Full-text Indexes [20]. These indexes take advantage of the relationship between the *bwt* and the suffix array to provide a compressed representation of a string supporting the efficient search and retrieval of the occurrences of an arbitrary pattern.

Open Problems

In addition to the investigation on the performance of *bwt*-based compressors, an open problem of great practical significance is the space-efficient computation of the *bwt*. Given a string s of length n over an alphabet Σ, both s and $\hat{s} = bwt(s)$ take $O(n \log |\Sigma|)$ bits. Unfortunately, the linear time algorithms shown in Fig. 2 make use of auxiliary arrays (i.e., *sa* and Ψ) whose storage takes $\Theta(n \log n)$ bits. This poses a serious limitation to the size of the largest *bwt* that can be computed in main memory. The problem of space- and time-efficient computation of the *bwt* is still open, even if interesting results are reported in [1, 3, 7, 13, 15, 19]. The problem of designing space-efficient algorithms for inverting the *bwt* is also open; see [7, 16, 20] and references therein for further details.

Experimental Results

An experimental study of the performance of several compression algorithms based on the *bwt* and a comparison with other state-of-the-art compressors is presented in [8].

Data Sets

The data sets used in [8] are available from http://people.unipmn.it/manzini/boosting. Other data sets relevant for compression and compressed indexing are available at the Pizza&Chili site http://pizzachili.di.unipi.it/.

URL to Code

The compression boosting page (http://people.unipmn.it/manzini/boosting) contains the source code of the algorithms tested in [8]. An extremely efficient code for the computation of the suffix array and the *bwt* (without compression) is available at http://code.google.com/p/libdivsufsort. The code of *bzip2* is available at http://www.bzip.org.

Cross-References

- Arithmetic Coding for Data Compression
- Compressing and Indexing Structured Text
- Compressed Suffix Array
- Suffix Array Construction
- Table Compression

Recommended Reading

1. Bauer MJ, Cox AJ, Rosone G (2013) Lightweight algorithms for constructing and inverting the BWT of string collections. Theor Comput Sci 483: 134–148
2. Bell TC, Cleary JG, Witten IH (1990) Text compression. Prentice Hall, Englewood Cliffs
3. Beller T, Zwerger M, Gog S, Ohlebusch E (2013) Space-efficient construction of the Burrows-Wheeler transform. In: Proceedings of the 20th international symposium on string processing and information retrieval (SPIRE), Jerusalem. LNCS, vol 8214. Springer, Berlin, pp 5–16

4. Bentley J, Sleator D, Tarjan R, Wei V (1986) A locally adaptive compression scheme. Commun ACM 29:320–330
5. Burrows M, Wheeler D (1994) A block sorting lossless data compression algorithm. Technical report 124, Digital Equipment Corporation
6. Culpepper JS, Petri M, Puglisi SJ (2012) Revisiting bounded context block-sorting transformations. Softw Pract Exp 42:1037–1054
7. Ferragina P, Gagie T, Manzini G (2012) Lightweight data indexing and compression in external memory. Algorithmica 63:707–730
8. Ferragina P, Giancarlo R, Manzini G (2006) The engineering of a compression boosting library: theory vs practice in BWT compression. In: Proceedings of the 14th European symposium on algorithms (ESA), Zurich. LNCS, vol 4168. Springer, Berlin, pp 756–767
9. Ferragina P, Giancarlo R, Manzini G (2009) The myriad virtues of wavelet trees. Inf Comput 207:849–866
10. Ferragina P, Giancarlo R, Manzini G, Sciortino M (2005) Boosting textual compression in optimal linear time. J ACM 52:688–713
11. Ferragina P, Luccio F, Manzini G, Muthukrishnan S (2009) Compressing and indexing labeled trees, with applications. J ACM 57
12. Gagie T, Manzini G (2010) Move-to-front, distance coding, and inversion frequencies revisited. Theor Comput Sci 411:2925–2944
13. Hon W, Sadakane K, Sung W (2009) Breaking a time-and-space barrier in constructing full-text indices. SIAM J Comput 38:2162–2178
14. Kaplan, H., Landau, S., Verbin, E.: A simpler analysis of Burrows-Wheeler-based compression. Theoretical Computer Science **387**, 220–235 (2007)
15. Kärkkäinen J (2007) Fast BWT in small space by blockwise suffix sorting. Theor Comput Sci 387:249–257
16. Kärkkäinen J, Kempa D, Puglisi SJ (2012) Slashing the time for BWT inversion. In: Proceedings of the IEEE data compression conference (DCC), Snowbird. IEEE Computer Society, pp 99–108
17. Kärkkäinen J, Sanders P, Burkhardt S (2006) Linear work suffix array construction. J ACM 53(6):918–936
18. Manzini G (2001) An analysis of the Burrows-Wheeler transform. J ACM 48:407–430
19. Na JC, Park K (2007) Alphabet-independent linear-time construction of compressed suffix arrays using o(n logn)-bit working space. Theor Comput Sci 385:127–136
20. Navarro G, Mäkinen V (2007) Compressed full text indexes. ACM Comput Surv 39(1):2
21. Salomon D (2007) Data compression: the complete reference, 4th edn. Springer, New York
22. Vo BD, Vo KP (2007) Compressing table data with column dependency. Theor Comput Sci 387(3):273–283

Byzantine Agreement

Michael Okun
Weizmann Institute of Science, Rehovot, Israel

B

Keywords

Byzantine generals; Consensus; Interactive consistency

Years and Authors of Summarized Original Work

1980; Pease, Shostak, Lamport

Problem Definition

The study of Pease, Shostak and Lamport was among the first to consider the problem of achieving a coordinated behavior between processors of a distributed system in the presence of failures [21]. Since the paper was published, this subject has grown into an extensive research area. Below is a presentation of the main findings regarding the specific questions addressed in their paper. In some cases this entry uses the currently accepted terminology in this subject, rather than the original terminology used by the authors.

System Model

A distributed system is considered to have n independent processors, p_1, ... ,p_n, each modeled as a (possibly infinite) state machine. The processors are linked by a communication network that supports direct communication between every pair of processors. The processors can communicate only by exchanging messages, where the sender of every message can be identified by the receiver. While the processors may fail, it is assumed that the communication subsystem is fail-safe. It is not known in advance which processors will not fail (remain correct) and which ones will

fail. The types of processor failures are classified according to the following hierarchy.

Crash failure A crash failure means that the processor no longer operates (ad infinitum, starting from the failure point). In particular, other processors will not receive messages from a faulty processor after it crashes.

Omission failure A processor fails to send and receive an arbitrary subset of its messages.

Byzantine failure A faulty processor behaves arbitrarily.

The Byzantine failure is further subdivided into two cases, according to the ability of the processors to create unfalsifiable signatures for their messages. In the *authenticated Byzantine failure* model it is assumed that each message is *signed* by its sender and that no other processor can fake a signature of a correct processor. Thus, even if such a message is forwarded by other processors, its authenticity can be verified. If the processors represent malevolent (human) users of a distributed system, a Public Key Infrastructure (PKI) is typically used to sign the messages (which involves cryptography related issues [17], not discussed here). Practically, in systems where processors are just "processors", a simple signature, such as CRC (cyclic redundancy check), might be sufficient [13]. In the *unauthenticated Byzantine failure* model there are no message signatures.

Definition of the Byzantine Agreement Problem

In the beginning, each processor p_i has an externally provided input value v_i, from some set V (of at least size 2). In the *Byzantine Agreement* (BA) problem, every correct processor p_i is required to decide on an output value $d_i \in V$ such that the following conditions hold:

Termination Eventually, p_i decides, i.e., the algorithm cannot run indefinitely.

Validity If the input value of all the processors is v, then the correct processors decide v.

Agreement All the correct processors decide on the same value.

For crash failures and omission failures there exists a stronger agreement condition:

Uniform Agreement No two processors (either correct or faulty) decide differently.

The termination condition has the following stronger version.

Simultaneous Termination All the correct processors decide in the *same round* (see definition below).

Timing Model

The BA problem was originally defined for *synchronous* distributed systems [18, 21]. In this timing model the processors are assumed to operate in lockstep, which allows to partition the execution of a protocol to rounds. Each round consists of a send phase, during which a processor can send a (different) message to each processor directly connected to it, followed by a receive phase, in which it receives messages sent by these processors in the current round. Unlimited local computations (state transitions) are allowed in both phases, which models the typical situation in real distributed systems, where computation steps are faster than the communication steps by several orders of magnitude.

Overview

This entry deals only with *deterministic* algorithms for the BA problem in the synchronous model. For algorithms involving randomization see the ▸Optimal Probabilistic Synchronous Byzantine Agreement entry in this volume. For results on BA in other models of synchrony, see ▸Asynchronous Consensus Impossibility, ▸Failure Detectors, ▸Consensus with Partial Synchrony entries in this volume.

Key Results

The maximum possible number of faulty processors is assumed to be bounded by an a priori specified number t (e.g., estimated from the failure probability of individual processor and the requirements on the failure probability of the

system as a whole). The number of processors that actually become faulty in a given execution is denoted by f, where $f \leq t$.

The complexity of synchronous distributed algorithms is measured by three complementary parameters. The first is the *round complexity,* which measures the number of rounds required by the algorithm. The second is the *message complexity*, i.e., the total number of messages (and sometimes also their size in bits) sent by all the processors (in case of Byzantine failures, only messages sent by correct processors are counted). The third complexity parameter measures the number of local operations, as in sequential algorithms.

All the algorithms presented bellow are efficient, i.e., the number of rounds, the number of messages and their size, and the local operations performed by each processor are polynomial in n. In most of the algorithms, both the exchanged messages and the local computations involve only the basic data structures (e.g., arrays, lists, queues). Thus, the discussion is restricted only to the round and the message complexities of the algorithms.

The network is assumed to be fully connected, unless explicitly stated otherwise.

Crash Failures

A simple BA algorithm which runs in $t + 1$ rounds and sends $O(n^2)$ messages, together with a proof that this number of rounds is optimal, can be found in textbooks on distributed computing [19]. Algorithms for deciding in $f + 1$ rounds, which is the best possible, are presented in [7, 23] (one additional round is necessary before the processors can stop [11]). Simultaneous termination requires $t + 1$ rounds, even if no failures actually occur [11], however there exists an algorithm that in any given execution stops in the earliest possible round [14]. For uniform agreement, decision can be made in $\min(f + 2, t + 1)$ rounds, which is tight [7].

In case of crash failures it is possible to solve the BA problem with $O(n)$ messages, which is also the lower bound. However, all known message-optimal BA algorithms require a superlinear time. An algorithm that runs in $O(f + 1)$ rounds and uses only $O(n \text{ polylog } n)$ messages, is presented in [8], along with an overview of other results on BA message complexity.

Omission Failures

The basic algorithm used to solve the crash failure BA problem works for omission failures as well, which allows to solve the problem in $t + 1$ rounds [23]. An algorithm which terminates in $\min(f + 2, t + 1)$ rounds was presented in [22]. Uniform agreement is impossible for $t \geq n/2$ [23]. For $t < n/2$, there is an algorithm that achieves uniform agreement in $\min(f + 2, t + 1)$ rounds (and $O(n^2 f)$ message complexity) [20].

Byzantine Failures with Authentication

A $(t + 1)$-round BA algorithm is presented in [12]. An algorithm which terminates in $\min(f + 2, t + 1)$ rounds can be found in [24]. The message complexity of the problem is analyzed in [10], where it is shown that the number of signatures and the number of messages in any authenticated BA algorithm are $\Omega(nt)$ and $\Omega(n + t^2)$, respectively. In addition, it is shown that $\Omega(nt)$ is the bound on the number of messages for the unauthenticated BA.

Byzantine Failures Without Authentication

In the unauthenticated case, the BA problem can be solved if and only if $n > 3t$. The proof can be found in [1, 19]. An algorithm that decides in $\min(f + 3, t + 1)$ rounds (it might require two additional rounds to stop) is presented in [16]. Unfortunately, this algorithm is complicated. Simpler algorithms, that run in $\min(2f + 4, 2t + 1)$ and $3\min(f + 2, t + 1)$ rounds, are presented in [24] and [5], respectively. In these algorithms the number of sent messages is $O(n^3)$, moreover, in the latter algorithm the messages are of constant size (2 bits). Both algorithms assume $V = \{0, 1\}$. To solve the BA problem for a larger V, several instances of a binary algorithm can be run in parallel. Alternatively, there exists a simple 2-round protocol that reduces a BA problem with

arbitrary initial values to the binary case, e.g., see Sect. 6.3.3 in [19]. For algorithms with optimal $O(nt)$ message complexity and $t + o(t)$ round complexity see [4, 9].

Arbitrary Network Topologies

When the network is not fully connected, BA can be solved for crash, omission and authenticated Byzantine failures if and only if it is $(t + 1)$-connected [12]. In case of Byzantine failures without authentication, BA has a solution if and only if the network is $(2t + 1)$-connected and $n > 3t$ [19]. In both cases the BA problem can be solved by simulating the algorithms for the fully connected network, using the fact that the number of disjoint communication paths between any pair of non-adjacent processors exceeds the number of faulty nodes by an amount that is sufficient for reliable communication.

Interactive Consistency and Byzantine Generals

The BA (consensus) problem can be stated in several similar ways. Two widely used variants are the *Byzantine Generals* (BG) problem and the *Interactive Consistency (*IC) problem. In the BG case there is a designated processor, say p_1, which is the only one to have an input value. The termination and agreement requirements of the BG problem are exactly as in BA, while the validity condition requires that if the input value of p_1 is v and p_1 is correct, then the correct processors decide v. The IC problem is an extension of BG, where every processor is "designated", so that each processor has to decide on a vector of n values, where the conditions for the i-th entry are as in BG, with p_i as the designated processor. For deterministic synchronous algorithms BA, BG and IC problems are essentially equivalent, e.g., see the discussion in [15].

Firing Squad

The above algorithms assume that the processors share a "global time", i.e., all the processors start in the same (first) round, so that their round counters are equal throughout the execution of the algorithm. However, there are cases in which the processors run in a synchronous network, yet each processor has its own notion of time (e.g., when each processor starts on its own, the round counter values are distinct among the processors). In these cases, it is desirable to have a protocol that allows the processors to agree on some specific round, thus creating a common round which synchronizes all the correct processors. This synchronization task, known as the *Byzantine firing squad* problem [6], is tightly realted to BA.

General Translation Techniques

One particular direction that was pursued as part of the research on the BA problem is the development of methods that automatically translate any protocol that tolerates a more benign failure type into one which tolerates more severe failures [24]. Efficient translations spanning the entire failure hierarchy, starting from crash failures all the way to unauthenticated Byzantine failures, can be found in [3] and in Ch. 12 of [1].

Applications

Due to the very tight synchronization assumptions made in the algorithms presented above, they are used mainly in real-time, safety-critical systems, e.g., aircraft control [13]. In fact, the original interest of Pease, Shostak and Lamport in this problem was raised by such an application [21]. In addition, BA protocols for the Byzantine failure case serve as a basic building block in many cryptographic protocols, e.g., secure multi-party computation [17], by providing a broadcast channel on top of pairwise communication channels.

Cross-References

- ▸ Asynchronous Consensus Impossibility
- ▸ Atomic Broadcast
- ▸ Consensus with Partial Synchrony
- ▸ Failure Detectors
- ▸ Optimal Probabilistic Synchronous Byzantine Agreement
- ▸ Randomization in Distributed Computing

- ▸ Renaming
- ▸ Set Agreement

Recommended Reading

1. Attiya H, Welch JL (1998) Distributed computing: fundamentals, simulations and advanced topics. McGraw-Hill, London
2. Barborak M, Dahbura A, Malek M (1993) The consensus problem in fault-tolerant computing. ACM Comput Surv 25(2):171–220
3. Bazzi RA, Neiger G (2001) Simplifying fault-tolerance: providing the abstraction of crash failures. J ACM 48(3):499–554
4. Berman P, Garay JA, Perry KJ (1992) Bit optimal distributed consensus. In: Yaeza-Bates R, Manber U (eds) Computer science research. Plenum Publishing, New York, pp 313–322
5. Berman P, Garay JA, Perry KJ (1992) Optimal early stopping in distributed consensus. In: Proceedings of the 6th international workshop on distributed algorithms (WDAG), Haifa, Nov 1992, pp 221–237
6. Burns JE, Lynch NA (1987) The Byzantine firing squad problem. Adv Comput Res 4:147–161
7. Charron-Bost B, Schiper A (2004) Uniform consensus is harder than consensus. J Algorithms 51(1):15–37
8. Chlebus BS, Kowalski DR (2006) Time and communication efficient consensus for crash failures. In: Proceedings of the 20th international international symposium on distributed computing (DISC), Stockholm, Sept 2006, pp 314–328
9. Coan BA, Welch JL (1992) Modular construction of a Byzantine agreement protocol with optimal message bit complexity. Inf Comput 97(1):61–85
10. Dolev D, Reischuk R (1985) Bounds on information exchange for Byzantine agreement. J ACM 32(1):191–204
11. Dolev D, Reischuk R, Strong HR (1990) Early stopping in Byzantine agreement. J ACM 37(4): 720–741
12. Dolev D, Strong HR (1983) Authenticated algorithms for Byzantine agreement. SIAM J Comput 12(4):656–666
13. Driscoll K, Hall B, Sivencrona H, Zumsteg P (2003) Byzantine fault tolerance, from theory to reality. In: Proceedings of the 22nd international conference on computer safety, reliability, and security (SAFECOMP), Edinburgh, Sept 2003, pp 235–248
14. Dwork C, Moses Y (1990) Knowledge and common knowledge in a Byzantine environment: crash failures. Inf Comput 88(2):156–186
15. Fischer MJ (1983) The consensus problem in unreliable distributed systems (a brief survey). Research report, YALEU/DCS/RR-273, Yale University, New Heaven
16. Garay JA, Moses Y (1998) Fully polynomial Byzantine agreement for $n > 3t$ processors in $t + 1$ rounds. SIAM J Comput 27(1):247–290
17. Goldreich O (2004) Foundations of cryptography, vols. 1-2. Cambridge University Press, Cambridge (2001)
18. Lamport L, Shostak RE, Pease MC (1982) The Byzantine generals problem. ACM Trans Program Lang Syst 4(3):382–401
19. Lynch NA (1996) Distributed algorithms. Morgan Kaufmann, San Francisco
20. Parvédy PR, Raynal M (2004) Optimal early stopping uniform consensus in synchronous systems with process omission failures. In: Proceedings of the 16th annual ACM symposium on parallel algorithms (SPAA), Barcelona, June 2004, pp 302–310
21. Pease MC, Shostak RE, Lamport L (1980) Reaching agreement in the presence of faults. J ACM 27(2):228–234
22. Perry KJ, Toueg S (1986) Distributed agreement in the presence of processor and communication faults. IEEE Trans Softw Eng 12(3):477–482
23. Raynal M (2002) Consensus in synchronous systems: a concise guided tour. In: Proceedings of the 9th Pacific rim international symposium on dependable computing (PRDC), Tsukuba-City, Dec 2002, pp 221–228
24. Toueg S, Perry KJ, Srikanth TK (1987) Fast distributed agreement. SIAM J Comput 16(3):445–457

C

Cache-Oblivious B-Tree

Rolf Fagerberg
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

Keywords

Cache-oblivious; Dictionary; External memory; Search tree

Years and Authors of Summarized Original Work

2005; Bender, Demaine, Farach-Colton

Problem Definition

Computers contain a hierarchy of memory levels, with vastly differing access times. Hence, the time for a memory access depends strongly on what is the innermost level containing the data accessed. In analysis of algorithms, the standard RAM (or von Neumann) model cannot capture this effect, and external memory models were introduced to better model the situation. The most widely used of these models is the two-level I/O-model [4], also called the external memory model or the disk access model. The I/O-model approximates the memory hierarchy by modeling two levels, with the inner level having size M, the outer level having infinite size, and transfers between the levels taking place in blocks of B consecutive elements. The cost of an algorithm is the number of such transfers it makes.

The cache-oblivious model, introduced by Frigo et al. [26], elegantly generalizes the I/O-model to a multilevel memory model by a simple measure: the algorithm is not allowed to know the value of B and M. More precisely, a cache-oblivious algorithm is an algorithm formulated in the RAM model, but analyzed in the I/O-model, with an analysis valid for *any* value of B and M. Cache replacement is assumed to take place automatically by an optimal off-line cache replacement strategy. Since the analysis holds for any B and M, it holds for all levels simultaneously (for a detailed version of this statement, see [26]).

The subject of the present chapter is that of efficient dictionary structures for the cache-oblivious model.

Key Results

The first cache-oblivious dictionary was given by Prokop [32], who showed how to lay out a static binary tree in memory such that searches take $O(\log_B n)$ memory transfers. This layout, often called the *van Emde Boas*

Research supported by Danish Council for Independent Research, Natural Sciences.

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

layout because it is reminiscent of the classic van Emde Boas data structure, also ensures that range searches take $O(\log_B n + k/B)$ memory transfers [8], where k is the size of the output. Both bounds are optimal for comparison-based searching.

The first dynamic, cache-oblivious dictionary was given by Bender et al. [13]. Making use of a variant of the van Emde Boas layout, a density maintenance algorithm of the type invented by Itai et al. [28], and weight-balanced B-trees [5], they arrived at the following results:

Theorem 1 ([13]) *There is a cache-oblivious dictionary structure supporting searches in $O(\log_B n)$ memory transfers and insertions and deletions in amortized $O(\log_B n)$ memory transfers.*

Theorem 2 ([13]) *There is a cache-oblivious dictionary structure supporting searches in $O(\log_B n)$ memory transfers, insertions and deletions in amortized $O(\log_B n + (\log^2 n)/B)$ memory transfers, and range searches in $O(\log_B n + k/B)$ memory transfers, where k is the size of the output.*

Later, Bender et al. [10] developed a cache-oblivious structure for maintaining linked lists which supports insertion and deletion of elements in $O(1)$ memory transfers and scanning of k consecutive elements in amortized $O(k/B)$ memory transfers. Combining this structure with the structure of the first theorem above, the following result can be achieved.

Theorem 3 ([10,13]) *There is a cache-oblivious dictionary structure supporting searches in $O(\log_B n)$ memory transfers, insertions and deletions in amortized $O(\log_B n)$ memory transfers, and range searches in amortized $O(\log_B n + k/B)$ memory transfers, where k is the size of the output.*

A long list of extensions of these basic cache-oblivious dictionary results has been given. We now survey these.

Bender et al. [12] and Brodal et al. [20] gave very similar proposals for reproducing the result of Theorem 2 but with simpler structures (avoiding the use of weight-balanced B-trees). Based on exponential trees, Bender et al. [11] gave a proposal with $O(\log_B n)$ worst-case queries and updates. They also gave a solution with partial persistence, where searches (in all versions of the structure) and updates (in latest version of the structure) require amortized $O(\log_B(m+n))$ memory transfers, where m is the number of versions and n is the number of elements in the version operated on. Bender et al. [14] extended the cache-oblivious model to a concurrent setting and gave three proposals for cache-oblivious B-trees in this setting. Bender et al. [16] presented cache-oblivious dictionary structures exploring trade-offs between faster insertion costs and slower search cost, and Brodal et al. [21] later gave improved structures meeting lower bounds. Franceschini and Grossi [25] showed how to achieve $O(\log_B n)$ worst-case queries and updates while using $O(1)$ space besides the space for the n elements stored. Brodal and Kejlberg-Rasmussen [19] extended this to structures adaptive to the working set bound and allowing predecessor queries. Cache-oblivious dictionaries for other data types such as strings [15, 18, 22–24, 27] and geometric data [1, 2, 6, 7, 9] have been given. The expected number of I/Os for hashing was studied in the cache-oblivious model in [31].

It has been shown [17] that the best possible multiplicative constant in the $\Theta(\log_B n)$ search bound for comparison-based searching is different in the I/O-model and in the cache-oblivious model. It has also been shown [1,3] that for three-sided range reporting in 2D, the best possible space bound for structures with worst-case optimal query times is different in the two models. The latter result implies that linear space cache-oblivious persistent B-trees with optimal worst-case bounds for (1D) range reporting are not possible.

Applications

Dictionaries solve a fundamental data structuring problem which is part of solutions for a very high number of computational problems. Dictionaries for external memory are useful in settings

where memory accesses are dominating the running time, and cache-oblivious dictionaries in particular stand out by their ability to optimize the access to all levels of an unknown memory hierarchy. This is an asset, e.g., when developing programs to be run on diverse or unknown architectures (such as software libraries or programs for heterogeneous distributed computing like grid computing and projects such as SETI@home). Even on a single, known architecture, the memory parameters available to a computational process may be nonconstant if several processes compete for the same memory resources. Since cache-oblivious algorithms are optimized for all parameter values, they have the potential to adapt more graceful to these changes and also to varying input sizes forcing different memory levels to be in use.

Open Problems

It is an open problem to find a data structure achieving worst-case versions of all of the bounds in Theorem 3.

Experimental Results

Cache-oblivious dictionaries have been studied empirically in [12, 15, 20, 29, 30, 33]. The overall conclusion of these investigations is that cache-oblivious methods easily can outperform RAM algorithms, although sometimes not as much as algorithms tuned to the specific memory hierarchy and problem size at hand. On the other hand, cache-oblivious algorithms seem to perform well on all levels of the memory hierarchy and to be more robust to changing problem sizes.

Cross-References

- B-trees
- Cache-Oblivious Model
- Cache-Oblivious Sorting

Recommended Reading

1. Afshani P, Zeh N (2011) Improved space bounds for cache-oblivious range reporting. In: Proceedings of the 22nd annual ACM-SIAM symposium on discrete algorithms, San Francisco, pp 1745–1758
2. Afshani P, Hamilton CH, Zeh N (2010) A general approach for cache-oblivious range reporting and approximate range counting. Comput Geom 43(8):700–712. Conference version appeared at SoCG 2009
3. Afshani P, Hamilton CH, Zeh N (2011) Cache-oblivious range reporting with optimal queries requires superlinear space. Discret Comput Geom 45(4):824–850. Conference version appeared at SoCG 2009
4. Aggarwal A, Vitter JS (1988) The input/output complexity of sorting and related problems. Commun ACM 31(9):1116–1127
5. Arge L, Vitter JS (2003) Optimal external memory interval management. SIAM J Comput 32(6):1488–1508
6. Arge L, Zeh N (2006) Simple and semi-dynamic structures for cache-oblivious planar orthogonal range searching. In: Proceedings of the 22nd ACM symposium on computational geometry, Sedona, pp 158–166
7. Arge L, de Berg M, Haverkort HJ (2005) Cache-oblivious R-trees. In: Proceedings of the 21st ACM symposium on computational geometry, Pisa, pp 170–179
8. Arge L, Brodal GS, Fagerberg R (2005) Cache-oblivious data structures. In: Mehta D, Sahni S (eds) Handbook on data structures and applications. CRC, Boca Raton
9. Arge L, Brodal GS, Fagerberg R, Laustsen M (2005) Cache-oblivious planar orthogonal range searching and counting. In: Proceedings of the 21st ACM symposium on computational geometry, Pisa, pp 160–169
10. Bender M, Cole R, Demaine E, Farach-Colton M (2002) Scanning and traversing: maintaining data for traversals in a memory hierarchy. In: Proceedings of the 10th annual European symposium on algorithms, Rome. LNCS, vol 2461, pp 139–151
11. Bender M, Cole R, Raman R (2002) Exponential structures for cache-oblivious algorithms. In: Proceedings of the 29th international colloquium on automata, languages, and programming, Málaga. LNCS, vol 2380, pp 195–207
12. Bender MA, Duan Z, Iacono J, Wu J (2004) A locality-preserving cache-oblivious dynamic dictionary. J Algorithms 53(2):115–136. Conference version appeared at SODA 2002
13. Bender MA, Demaine ED, Farach-Colton M (2005) Cache-oblivious B-trees. SIAM J Comput 35(2):341–358. Conference version appeared at FOCS 2000
14. Bender MA, Fineman JT, Gilbert S, Kuszmaul BC (2005) Concurrent cache-oblivious B-trees. In: Proceedings of the 17th annual ACM symposium

on parallelism in algorithms and architectures, Las Vegas, pp 228–237
15. Bender MA, Farach-Colton M, Kuszmaul BC (2006) Cache-oblivious string B-trees. In: Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, Chicago, pp 233–242
16. Bender MA, Farach-Colton M, Fineman JT, Fogel YR, Kuszmaul BC, Nelson J (2007) Cache-oblivious streaming B-trees. In: Proceedings of the 19th annual ACM symposium on parallelism in algorithms and architectures, San Diego, pp 81–92
17. Bender MA, Brodal GS, Fagerberg R, Ge D, He S, Hu H, Iacono J, López-Ortiz A (2011) The cost of cache-oblivious searching. Algorithmica 61(2):463–505. Conference version appeared at FOCS 2003
18. Brodal GS, Fagerberg R (2006) Cache-oblivious string dictionaries. In: Proceedings of the 17th annual ACM-SIAM symposium on discrete algorithms, Miami, pp 581–590
19. Brodal GS, Kejlberg-Rasmussen C (2012) Cache-oblivious implicit predecessor dictionaries with the working set property. In: Proceedings of the 29th annual symposium on theoretical aspects of computer science, Paris. Leibniz international proceedings in informatics, vol 14, pp 112–123
20. Brodal GS, Fagerberg R, Jacob R (2002) Cache-oblivious search trees via binary trees of small height. In: Proceedings of the 13th annual ACM-SIAM symposium on discrete algorithms, San Francisco, pp 39–48
21. Brodal GS, Demaine ED, Fineman JT, Iacono J, Langerman S, Munro JI (2010) Cache-oblivious dynamic dictionaries with update/query tradeoffs. In: Proceedings of the 21st annual ACM-SIAM symposium on discrete algorithms, Austin, pp 1448–1456
22. Ferragina P (2013) On the weak prefix-search problem. Theor Comput Sci 483:75–84. Conference version appeared at CPM 2011
23. Ferragina P, Venturini R (2013) Compressed cache-oblivious string B-tree. In: Proceedings of the algorithms – ESA 2013 – 21st annual European symposium, Sophia Antipolis. LNCS, vol 8125, pp 469–480
24. Ferragina P, Grossi R, Gupta A, Shah R, Vitter JS (2008) On searching compressed string collections cache-obliviously. In: Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, Vancouver, pp 181–190
25. Franceschini G, Grossi R (2003) Optimal worst-case operations for implicit cache-oblivious search trees. In: Proceedings of the 8th international workshop on algorithms and data structures (WADS), Ottawa. LNCS, vol 2748, pp 114–126
26. Frigo M, Leiserson CE, Prokop H, Ramachandran S (2012) Cache-oblivious algorithms. ACM Trans Algorithms 8(1):4. Conference version appeared at FOCS 1999
27. Hon W, Lam TW, Shah R, Tam S, Vitter JS (2011) Cache-oblivious index for approximate string matching. Theor Comput Sci 412(29):3579–3588. Conference version appeared at CPM 2007
28. Itai A, Konheim AG, Rodeh M (1981) A sparse table implementation of priority queues. In: 8th International colloquium on Automata, languages and programming, Acre (Akko). LNCS, vol 115, pp 417–431
29. Ladner RE, Fortna R, Nguyen BH (2002) A comparison of cache aware and cache oblivious static search trees using program instrumentation. In: Experimental algorithmics. LNCS, Springer-Verlag, Berlin/Heidelberg, vol 2547, pp 78–92
30. Lindstrom P, Rajan D (2014) Optimal hierarchical layouts for cache-oblivious search trees. In: IEEE 30th international conference on data engineering, Chicago, pp 616–627
31. Pagh R, Wei Z, Yi K, Zhang Q (2014) Cache-oblivious hashing. Algorithmica 69(4):864–883. Conference version appeared at PODS 2010
32. Prokop H (1999) Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology
33. Rahman N, Cole R, Raman R (2001) Optimised predecessor data structures for internal memory. In: Proceedings of the algorithm engineering, 5th international workshop (WAE), Aarhus. LNCS, vol 2141, pp 67–78

Cache-Oblivious Model

Rolf Fagerberg
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

Keywords

Cache-oblivious; Computational models; External memory

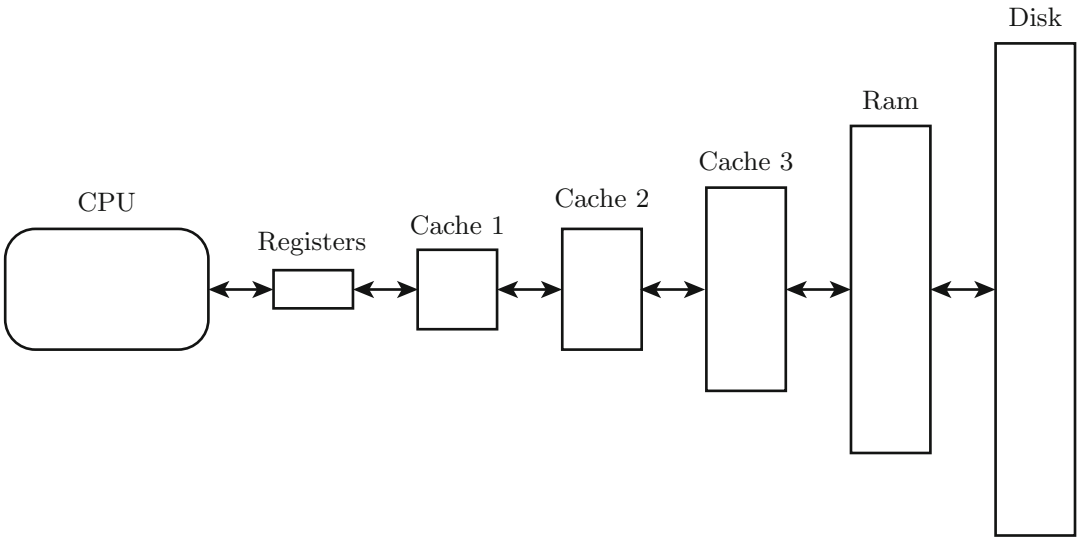
Years and Authors of Summarized Original Work

1999; Frigo, Leiserson, Prokop, Ramachandran

Problem Definition

The memory system of contemporary computers consists of a hierarchy of memory levels, with

Research supported by Danish Council for Independent Research, Natural Sciences.

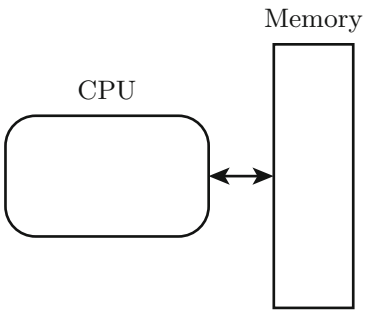


Cache-Oblivious Model, Fig. 1 The memory hierarchy

each level acting as a cache for the next; a typical hierarchy consists of registers, level 1 cache, level 2 cache, level 3 cache, main memory, and disk (Fig. 1). One characteristics of the hierarchy is that the memory levels get larger and slower the further they get from the processor, with the access time increasing most dramatically between RAM memory and disk. Another characteristics is that data is moved between levels in blocks of consecutive elements.

As a consequence of the differences in access time between the levels, the cost of a memory access depends highly on what is the current lowest memory level holding the element accessed. Hence, the memory access pattern of an algorithm has a major influence on its practical running time. Unfortunately, the RAM model (Fig. 2) traditionally used to design and analyze algorithms is not capable of capturing this, as it assumes that all memory accesses take equal time.

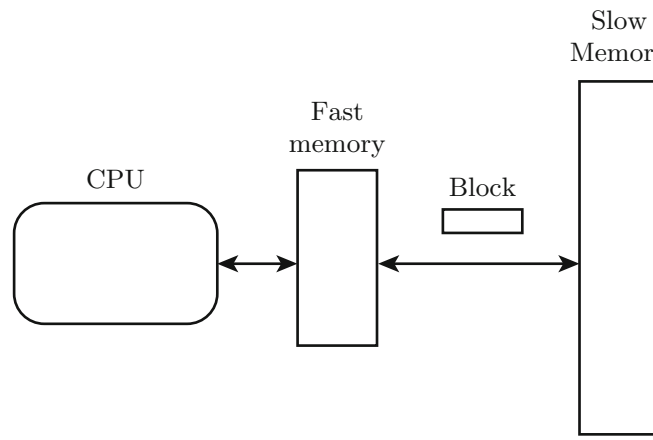
To better account for the effects of the memory hierarchy, a number of computational models have been proposed. The simplest and most successful is the two-level I/O-model introduced by Aggarwal and Vitter [3] (Fig. 3). In this model a two-level memory hierarchy is assumed, consisting of a fast memory of size M and a slower memory of infinite size, with



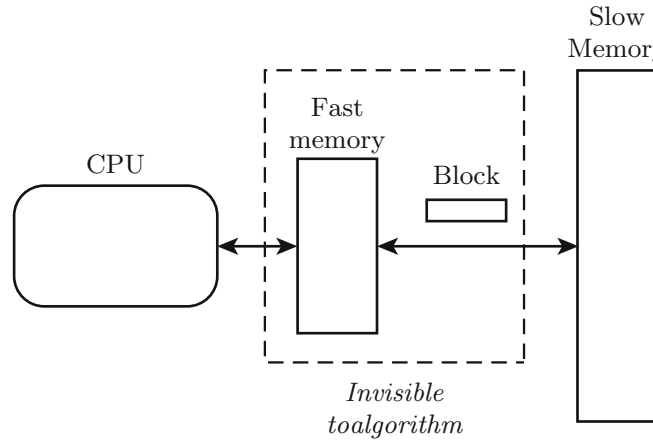
Cache-Oblivious Model, Fig. 2 The RAM model

data transferred between the levels in blocks of B consecutive elements. Computation can only be performed on data in the fast memory, and algorithms are assumed to have complete control over transfers of blocks between the two levels. Such a block transfer is denoted a *memory transfer*. The complexity measure is the number of memory transfers performed. The strength of the I/O-model is that it captures part of the memory hierarchy while being sufficiently simple to make design and analysis of algorithms feasible. Over the last two decades, a large body of results for the I/O-model has been produced, covering most areas of algorithmics. For an overview, see the surveys [5, 32, 34–36].

Cache-Oblivious Model, Fig. 3 The I/O-model



Cache-Oblivious Model, Fig. 4 The cache-oblivious model



More elaborate models of multilevel memory have been proposed (see e.g., [34] for an overview) but these models have been less successful than the I/O-model, mainly because of their complexity which makes analysis of algorithms harder. All these models, including the I/O-model, assume that the characteristics of the memory hierarchy (the level and block sizes) are known.

In 1999 the *cache-oblivious model* (Fig. 4) was introduced by Frigo et al. [30]. In short, a cache-oblivious algorithm is an algorithm formulated in the RAM model but analyzed in the I/O-model, with the analysis required to hold for *any* block size B and memory size M. Memory transfers are assumed to take place automatically by an optimal off-line cache replacement strategy.

The crux of the cache-oblivious model is that because the I/O-model analysis holds for any block and memory size, it holds for all levels of a multilevel memory hierarchy (see [30] for a detailed version of this statement). Put differently, by optimizing an algorithm to one unknown level of the memory hierarchy, it is optimized to all levels simultaneously. Thus, the cache-oblivious model elegantly generalizes the I/O-model to a multilevel memory model by one simple measure: the algorithm is not allowed to know the value of B and M. The challenge, of course, is to develop algorithms having good memory transfer analyses under these conditions.

Besides capturing the entire memory hierarchy in a conceptually simple way, the cache-oblivious model has other benefits: Algorithms developed in the model do not rely on knowing the parameters of the memory hierarchy, which is an asset when developing programs to be run on diverse or unknown architectures

(e.g., software libraries or programs for heterogeneous distributed computing such as grid computing and projects like SETI@home). Even on a single, known architecture, the memory parameters available to a computational process may be nonconstant if several processes compete for the same memory resources. Since cache-oblivious algorithms are optimized for all parameter values, they have the potential to adapt more graceful to these changes. Also, the same code will adapt to varying input sizes forcing different memory levels to be in use. Finally, cache-oblivious algorithms automatically are optimizing the use of translation lookaside buffers (a cache holding recently accessed parts of the page table used for virtual memory) of the CPU, which may be seen as second memory hierarchy parallel to the one mentioned in the introduction.

Possible weak points of the cache-oblivious model are the assumption of optimal off-line cache replacement and the lack of modeling of the limited associativity of many of the levels of the hierarchy. The first point is mitigated by the fact that normally, the provided analysis of a proposed cache-oblivious algorithm will work just as well assuming a least recently used cache replacement policy, which is closer to actual replacement strategies of computers. The second point is also a weak point of most other memory models.

Key Results

This section surveys a number of the known results in the cache-oblivious model. Other surveys available include [6, 15, 26, 32].

First of all, note that scanning an array of N elements takes $O(N/B)$ memory transfers for any values of B and M and hence is an optimal cache-oblivious algorithm. Thus, standard RAM algorithms based on scanning may already posses good analysis in the cache-oblivious model – for instance, the classic deterministic linear time selection algorithm has complexity $O(N/B)$ [26].

For sorting, a fundamental fact in the I/O-model is that comparison-based sorting of N elements takes $\Theta(\mathrm{Sort}(N))$ memory transfers [3], where $\mathrm{Sort}(N) = \frac{N}{B}\log_{M/B}\frac{N}{M}$. Also in the cache-oblivious model, sorting can be carried out in $\Theta(\mathrm{Sort}(N))$ memory transfer, if one makes the so-called *tall cache* assumption $M \geq B^{1+\varepsilon}$ [16, 30]. Such an assumption has been shown to be necessary [18], which proves a separation in power between cache-oblivious algorithms and algorithms in the I/O-model (where this assumption is not needed for the sorting bound).

For searching, B-trees have cost $O(\log_B N)$, which is optimal in the I/O-model for comparison-based searching. This cost is also attainable in the cache-oblivious model, as shown for the static case in [33] and for the dynamic case in [12]. Also for searching, a separation between cache-oblivious algorithms and algorithms in the I/O-model has been shown [13] in the sense that the constants attainable in the $O(\log_B N)$ bound are provably different.

By now, a large number of cache-oblivious algorithms and data structures in many areas have been given. These include priority queues [7, 17]; many dictionaries for standard data, string data, and geometric data (see survey in section on cache-oblivious B-trees); and algorithms for other problems in computational geometry [8, 16, 22], for graph problems [4, 7, 19, 23, 31], for scanning dynamic sets [9], for layout of static trees [11], for search problems on multi-sets [28], for dynamic programming [14, 24], for adaptive sorting [20], for inplace sorting [29], for sorting of strings [27], for partial persistence [10], for matrix operations [30], and for the fast Fourier transform [30].

In the negative direction, a few further separations in power between cache-oblivious algorithms and algorithms in the I/O-model are known. Permuting in the I/O-model has complexity $\Theta(\min\{\mathrm{Sort}(N), N\})$, assuming that elements are indivisible [3]. It has been proven [18] that this asymptotic complexity cannot be attained in the cache-oblivious model. A separation with respect to space complexity has been proven [2] for three-sided range reporting in 2D where the best possible space bound for structures with worst-case optimal query times is different in the two models. This result also implies that linear space cache-oblivious persistent B-trees with optimal

worst-case bounds for (1D) range reporting are not possible.

Applications

The cache-oblivious model is a means for design and analysis of algorithms that use the memory hierarchy of computers efficiently.

Experimental Results

Cache-oblivious algorithms have been evaluated empirically in a number of areas, including sorting [21], searching (see survey in section on cache-oblivious B-trees), matrix algorithms [1, 30, 37], and dynamic programming [24, 25].

The overall conclusion of these investigations is that cache-oblivious methods often outperform RAM algorithms but not always exactly as much as do algorithms tuned to the specific memory hierarchy and problem size. On the other hand, cache-oblivious algorithms seem to perform well on all levels of the memory hierarchy and to be more robust to changing problem sizes than cache-aware algorithms.

Cross-References

- ▸ Cache-Oblivious B-Tree
- ▸ Cache-Oblivious Sorting
- ▸ I/O-Model

Recommended Reading

1. Adams MD, Wise DS (2006) Seven at one stroke: results from a cache-oblivious paradigm for scalable matrix algorithms. In: Proceedings of the 2006 workshop on memory system performance and correctness, San Jose, pp 41–50
2. Afshani P, Zeh N (2011) Improved space bounds for cache-oblivious range reporting. In: Proceedings of the 22nd annual ACM-SIAM symposium on discrete algorithms, San Francisco, pp 1745–1758
3. Aggarwal A, Vitter JS (1988) The Input/Output complexity of sorting and related problems. Commun ACM 31(9):1116–1127
4. Allulli L, Lichodzijewski P, Zeh N (2007) A faster cache-oblivious shortest-path algorithm for undirected graphs with bounded edge lengths. In: Proceedings of the 18th annual ACM-SIAM symposium on discrete algorithms, New Orleans, pp 910–919
5. Arge L (2002) External memory data structures. In: Abello J, Pardalos PM, Resende MGC (eds) Handbook of massive data sets. Kluwer Academic, Dordrecht/Boston/London, pp 313–358
6. Arge L, Brodal GS, Fagerberg R (2005) Cache-oblivious data structures. In: Mehta D, Sahni S (eds) Handbook on data structures and applications. Chapman & Hall/CRC, Boca Raton/London/New York/Washington, D.C
7. Arge L, Bender MA, Demaine ED, Holland-Minkley B, Munro JI (2007) An optimal cache-oblivious priority queue and its application to graph algorithms. SIAM J Comput 36(6):1672–1695. Conference version appeared at STOC 2002
8. Arge L, Mølhave T, Zeh N (2008) Cache-oblivious red-blue line segment intersection. In: Proceedings of the 16th annual European symposium on algorithms, Karlsruhe. LNCS, vol 5193, pp 88–99
9. Bender M, Cole R, Demaine E, Farach-Colton M (2002) Scanning and traversing: maintaining data for traversals in a memory hierarchy. In: Proceedings of the 10th annual European symposium on algorithms, Rome. LNCS, vol 2461, pp 139–151
10. Bender M, Cole R, Raman R (2002) Exponential structures for cache-oblivious algorithms. In: Proceedings of the 29th international colloquium on automata, languages, and programming, Málaga. LNCS, vol 2380, pp 195–207
11. Bender M, Demaine E, Farach-Colton M (2002) Efficient tree layout in a multilevel memory hierarchy. In: Proceedings of the 10th annual European symposium on algorithms, Rome. LNCS, vol 2461, pp 165–173, corrected full version at http://arxiv.org/abs/cs/0211010
12. Bender MA, Demaine ED, Farach-Colton M (2005) Cache-oblivious B-trees. SIAM J Comput 35(2):341–358. Conference version appeared at FOCS 2000
13. Bender MA, Brodal GS, Fagerberg R, Ge D, He S, Hu H, Iacono J, López-Ortiz A (2011) The cost of cache-oblivious searching. Algorithmica 61(2):463–505. Conference version appeared at FOCS 2003
14. Bille P, Stöckel M (2012) Fast and cache-oblivious dynamic programming with local dependencies. In: Proceedings of the 6th international conference on language and automata theory and applications, A Coruña. LNCS, vol 7183, pp 131–142
15. Brodal GS (2004) Cache-oblivious algorithms and data structures. In: Proceedings of the 9th Scandinavian workshop on algorithm theory, Humlebæk. LNCS, vol 3111, pp 3–13
16. Brodal GS, Fagerberg R (2002) Cache oblivious distribution sweeping. In: Proceedings of the 29th international colloquium on automata, languages, and programming, Málaga. LNCS, vol 2380, pp 426–438

17. Brodal GS, Fagerberg R (2002) Funnel heap – a cache oblivious priority queue. In: Proceedings of the 13th international symposium on algorithms and computation, Vancouver. LNCS, vol 2518, pp 219–228
18. Brodal GS, Fagerberg R (2003) On the limits of cache-obliviousness. In: Proceedings of the 35th annual ACM symposium on theory of computing, San Diego, pp 307–315
19. Brodal GS, Fagerberg R, Meyer U, Zeh N (2004) Cache-oblivious data structures and algorithms for undirected breadth-first search and shortest paths. In: Proceedings of the 9th Scandinavian workshop on algorithm theory, Humlebæk. LNCS, vol 3111, pp 480–492
20. Brodal GS, Fagerberg R, Moruz G (2005) Cache-aware and cache-oblivious adaptive sorting. In: Proceedings of the 32nd international colloquium on automata, languages and programming, Lisbon. LNCS, vol 3580, pp 576–588
21. Brodal GS, Fagerberg R, Vinther K (2007) Engineering a cache-oblivious sorting algorithm. ACM J Exp Algorithmics 12:Article 2.2. Conference version appeared at ALENEX 2004
22. Chan TM, Chen EY (2010) Optimal in-place and cache-oblivious algorithms for 3-D convex hulls and 2-D segment intersection. Comput Geom 43(8):636–646
23. Chowdhury RA, Ramachandran V (2004) Cache-oblivious shortest paths in graphs using buffer heap. In: Proceedings of the 16th annual ACM symposium on parallelism in algorithms and architectures, Barcelona
24. Chowdhury RA, Ramachandran V (2006) Cache-oblivious dynamic programming. In: Proceedings of the 17th annual ACM-SIAM symposium on discrete algorithms, Miami, pp 591–600
25. Chowdhury RA, Le HS, Ramachandran V (2010) Cache-oblivious dynamic programming for bioinformatics. IEEE/ACM Trans Comput Biol Bioinf 7(3):495–510
26. Demaine ED (2002) Cache-oblivious algorithms and data structures. Lecture notes from the EEF summer school on massive data sets. Online version at http://theory.csail.mit.edu/~edemaine/papers/BRICS2002/
27. Fagerberg R, Pagh A, Pagh R (2006) External string sorting: faster and cache-oblivious. In: Proceedings of the 23rd annual symposium on theoretical aspects of computer science, Marseille. LNCS, vol 3884, pp 68–79
28. Farzan A, Ferragina P, Franceschini G, Munro JI (2005) Cache-oblivious comparison-based algorithms on multisets. In: Proceedings of the 13th annual European symposium on algorithms, Palma de Mallorca. LNCS, vol 3669, pp 305–316
29. Franceschini G (2004) Proximity mergesort: optimal in-place sorting in the cache-oblivious model. In: Proceedings of the 15th annual ACM-SIAM symposium on discrete algorithms, New Orleans, pp 291–299
30. Frigo M, Leiserson CE, Prokop H, Ramachandran S (2012) Cache-oblivious algorithms. ACM Trans Algorithms 8(1):4. Conference version appeared at FOCS 1999
31. Jampala H, Zeh N (2005) Cache-oblivious planar shortest paths. In: Proceedings of the 32nd international colloquium on automata, languages, and programming, Lisbon. LNCS, vol 3580, pp 563–575
32. Meyer U, Sanders P, Sibeyn JF (eds) (2003) Algorithms for memory hierarchies. LNCS, vol 2625. Springer, Berlin/Heidelberg/New York
33. Prokop H (1999) Cache-oblivious algorithms. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology
34. Vitter JS (2001) External memory algorithms and data structures: dealing with MASSIVE data. ACM Comput Surv 33(2):209–271
35. Vitter JS (2005) Geometric and spatial data structures in external memory. In: Mehta D, Sahni S (eds) Handbook on data structures and applications. Chapman & Hall/CRC, Boca Raton/London/New York/Washington, D.C
36. Vitter JS (2008) Algorithms and data structures for external memory. Found Trends Theor Comput Sci 2(4):305–474
37. Yotov K, Roeder T, Pingali K, Gunnels JA, Gustavson FG (2007) An experimental comparison of cache-oblivious and cache-conscious programs. In: Proceedings of the 19th annual ACM symposium on parallelism in algorithms and architectures, San Diego, pp 93–104

Cache-Oblivious Sorting

Gerth Stølting
Department of Computer Science, University of Aarhus, Århus, Denmark

Keywords

Funnelsort

Problem Definition

Sorting a set of elements is one of the most well-studied computational problems. In the cache-oblivious setting, the first study of sorting was presented in 1999 in the seminal paper by Frigo et al. [8] that introduced the cache-oblivious framework for developing algorithms

aimed at machines with (unknown) hierarchical memory.

Model

In the cache-oblivious setting, the computational model is a machine with two levels of memory: a cache of limited capacity and a secondary memory of infinite capacity. The capacity of the cache is assumed to be M elements, and data is moved between the two levels of memory in blocks of B consecutive elements. Computations can only be performed on elements stored in cache, i.e., elements from secondary memory need to be moved to the cache before operations can access the elements. Programs are written as acting directly on one unbounded memory, i.e., programs are like standard RAM programs. The necessary block transfers between cache and secondary memory are handled automatically by the model, assuming an optimal offline cache replacement strategy. The core assumption of the cache-oblivious model is that M and B are *unknown to the algorithm*, whereas in the related I/O model introduced by Aggarwal and Vitter [1], the algorithms know M and B, and the algorithms perform the block transfers explicitly. A thorough discussion of the cache-oblivious model and its relation to multilevel memory hierarchies is given in [8].

Sorting

For the sorting problem, the input is an array of N elements residing in secondary memory, and the output is required to be an array in secondary memory, storing the input elements in sorted order.

Key Results

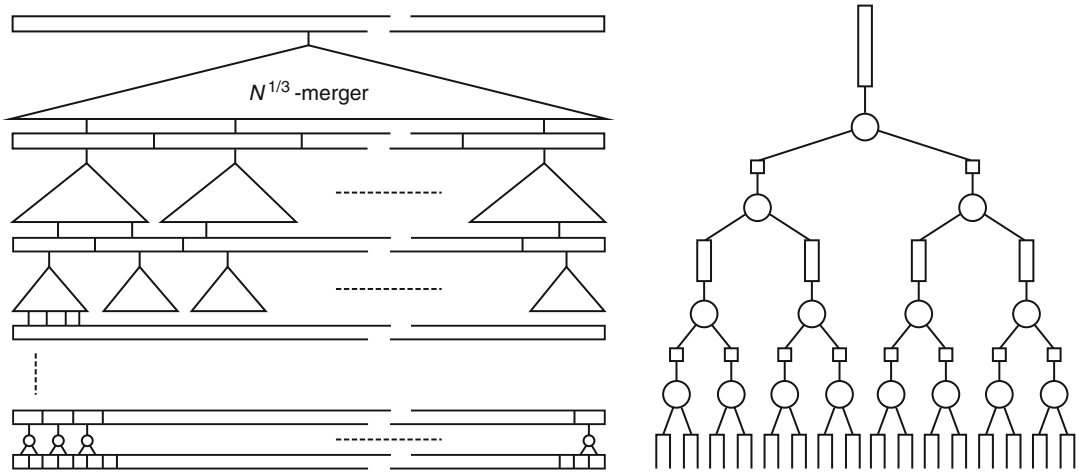
In the I/O model, tight upper and lower bounds were proved for the sorting problem and the problem of permuting an array [1]. In particular it was proved that sorting requires $\Omega\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$ block transfers and permuting an array requires $\Omega\left(\min\left\{N, \frac{N}{B}\log_{M/B}\frac{N}{B}\right\}\right)$ block transfers. Since lower bounds for the I/O model also hold for the cache-oblivious model, the lower bounds from [1] immediately give a lower bound of $\Omega\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$ block transfers for cache-oblivious sorting and $\Omega\left(\min\left\{N, \frac{N}{B}\log_{M/B}\frac{N}{B}\right\}\right)$ block transfers for cache-oblivious permuting. The upper bounds from [1] cannot be applied to the cache-oblivious setting since these algorithms make explicit use of B and M.

Binary mergesort performs $O(N\log_2 N)$ comparisons, but analyzed in the cache-oblivious model, it performs $O\left(\frac{N}{B}\log_2\frac{N}{M}\right)$ block transfers which is a factor $\Theta\left(\log\frac{M}{B}\right)$ from the lower bound (assuming a recursive implementation of binary mergesort, in order to get M in the denominator in the $\log N/M$ part of the bound on the block transfers). Another comparison-based sorting algorithm is the classical quicksort sorting algorithm from 1962 by Hoare [9] that performs expected $O(N\log_2 N)$ comparisons and expected $O\left(\frac{N}{B}\log_2\frac{N}{M}\right)$ block transfers. Both these algorithms achieve their relatively good performance for the number of block transfers from the fact that they are based on repeated scanning of arrays – a property not shared with, e.g., heapsort [10] that has a very poor performance of $\Theta\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$ block transfers. In the I/O model, the optimal performance of $O\left(\frac{M}{B}\log_{M/B}\frac{N}{B}\right)$ is achieved by generalizing binary mergesort to $\Theta\left(\frac{M}{B}\right)$-way mergesort [1].

Frigo et al. in [8] presented two cache-oblivious sorting algorithms (which can also be used to permute an array of elements). The first algorithm [8, Section 4] is denoted as *funnelsort* and is a reminiscent of classical binary mergesort, whereas the second algorithm [8, Section 5] is a distribution-based sorting algorithm. Both algorithms perform *optimal* $O\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$ block transfers – provided a *tall cache assumption* $M = \Omega(B^2)$ is satisfied.

Funnelsort

The basic idea of funnelsort is to rearrange the sorting process performed by binary mergesort, such that the processed data is stored "locally." This is achieved by two basic ideas: (1) a



Cache-Oblivious Sorting, Fig. 1 The overall recursion of funnelsort (*left*) and a 16-merger (*right*)

top-level recursion that partitions the input into $N^{1/3}$ sequences of size $N^{2/3}$ funnelsorts these sequences recursively and merges the resulting sorted subsequences using an $N^{1/3}$-*merger*. (2) A k-merger is recursively defined to perform binary merging of k input sequences in a clever schedule with an appropriate recursive layout of data in memory using buffers to hold suspended merging processes (see Fig. 1). Subsequently two simplifications were made, without sacrificing the asymptotic number of block transfers performed. In [3], it was proved that the binary merging could be performed lazily, simplifying the scheduling of merging. In [5], it was further observed that the recursive layout of k-mergers is not necessary. It is sufficient that a k-merger is stored in a consecutive array, i.e., the buffers can be laid out in an arbitrary order which simplifies the construction algorithm for the k-merger.

Implicit Cache-Oblivious Sorting

Franceschini in [7] showed how to perform optimal cache-oblivious sorting implicitly using only $O(1)$ space, i.e., all data is stored in the input array except for $O(1)$ additional words of information. In particular the output array is just a permutation of the input array.

The Role of the Tall Cache Assumption

The role of the tall cache assumption on cache-oblivious sorting was studied by Brodal and Fagerberg in [4]. If no tall cache assumption is made, they proved the following theorem:

Theorem 1 ([4], Corollary 3) *Let* $B_1 = 1$ *and* $B_2 = M/2$. *For any cache-oblivious comparison-based sorting algorithm, let* t_1 *and* t_2 *be upper bounds on the number of I/Os performed for block sizes* B_1 *and* B_2. *If for a real number* $d \geq 0$, *it is satisfied that* $t_2 = d \cdot \frac{N}{B_2} \log_{M/B_2} \frac{N}{B_2}$ *then* $t_1 > 1/8 \cdot N \log_2 N/M$.

The theorem shows that cache-oblivious comparison-based sorting without a tall cache assumption cannot match the performance of algorithms in the I/O model where M and B are known to the algorithm. It also has the natural interpretation that if a cache-oblivious algorithm is required to be I/O optimal for the case $B = M/2$, then binary mergesort is best possible – any other algorithm will be the same factor of $\Theta(\log M)$ worse than the optimal block transfer bound for the case $M >> D$.

For the related problem of permuting an array, the following theorem states that for all possible tall cache assumptions $B \leq M^{\delta}$, no cache-oblivious permuting algorithm exists with a block transfer bound (even only in the average case sense), matching the worst case bound in the I/O model.

Theorem 2 ([4], Theorem 2) *For all* $\delta > 0$, *there exists no cache-oblivious algorithm for*

permuting that for all $M \geq 2B$ *and* $1 \leq B \leq M^{\delta}$ *achieves* $O\left(\min\left\{N, \frac{N}{B}\log_{M/B}\frac{N}{B}\right\}\right)$ *I/Os averaged over all possible permutations of size* N.

Applications

Many problems can be reduced to cache-oblivious sorting. In particular Arge et al. [2] developed a cache-oblivious priority queue based on a reduction to sorting. They furthermore showed how a cache-oblivious priority queue can be applied to solve a sequence of graph problems, including list ranking, BFS, DFS, and minimum spanning trees.

Brodal and Fagerberg in [3] showed how to modify the cache-oblivious lazy funnelsort algorithm to solve several problems within computational geometry, including orthogonal line segment intersection reporting, all the nearest neighbors, 3D maxima problem, and batched orthogonal range queries. All these problems can be solved by a computation process very similarly to binary mergesort with an additional problem-dependent twist. This general framework to solve computational geometry problems is denoted as *distribution sweeping*.

Open Problems

Since the seminal paper by Frigo et al. [8] introducing the cache-oblivious framework, there has been a lot of work on developing algorithms with a good theoretical performance, but only a limited amount of work has been done on implementing these algorithms. An important issue for future work is to get further experimental results consolidating the cache-oblivious model as a relevant model for dealing efficiently with hierarchical memory.

Experimental Results

A detailed experimental study of the cache-oblivious sorting algorithm funnelsort was performed in [5]. The main result of [5] is that a carefully implemented cache-oblivious sorting algorithm can be faster than a tuned implementation of quicksort already for input sizes well within the limits of RAM. The implementation is also at least as fast as the recent cache-aware implementations included in the test. On disk, the difference is even more pronounced regarding quicksort and the cache-aware algorithms, whereas the algorithm is slower than a careful implementation of multiway mergesort optimized for external memory such as in TPIE [6].

URL to Code

http://kristoffer.vinther.name/projects/funnelsort/

Cross-References

- Cache-Oblivious Model
- External Sorting and Permuting
- I/O-Model

Recommended Reading

1. Aggarwal A, Vitter JS (1988) The input/output complexity of sorting and related problems. Commun ACM 31(9):1116–1127
2. Arge L, Bender MA, Demaine ED, Holland-Minkley B, Munro JI (2002) Cache-oblivious priority queue and graph algorithm applications. In: Proceedings of the 34th annual ACM symposium on theory of computing. ACM, New York, pp 268–276
3. Brodal GS, Fagerberg R (2002) Cache oblivious distribution sweeping. In: Proceedings of the 29th international colloquium on automata, languages, and programming. Lecture notes in computer science, vol 2380, pp 426–438. Springer, Berlin
4. Brodal GS, Fagerberg R (2003) On the limits of cache-obliviousness. In: Proceedings of the 35th annual ACM symposium on theory of computing. ACM, New York, pp 307–315
5. Brodal GS, Fagerberg R, Vinther K (2007) Engineering a cache-oblivious sorting algorithm. ACM J Exp Algoritmics (Special Issue of ALENEX 2004) 12(2.2):23
6. Department of Computer Science, Duke University. TPIE: a transparent parallel I/O environment. http://www.cs.duke.edu/TPIE/. Accessed 2002

7. Franceschini G (2004) Proximity mergesort: optimal in-place sorting in the cache-oblivious model. In: Proceedings of the 15th annual ACM-SIAM symposium on discrete algorithms (SODA). SIAM, Philadelphia, p 291
8. Frigo M, Leiserson CE, Prokop H, Ramachandran S (1999) Cache-oblivious algorithms. In: Proceedings of the 40th annual symposium on foundations of computer science. IEEE Computer Society Press, Los Alamitos, pp 285–297
9. Hoare CAR (1962) Quicksort. Comput J 5(1):10–15
10. Williams JWJ (1964) Algorithm 232: Heapsort. Commun ACM 7(6):347–348

Cache-Oblivious Spacetree Traversals

Michael Bader[1] and Tobias Weinzierl[2]
[1]Department of Informatics, Technical University of Munich, Garching, Germany
[2]School of Engineering and Computing Sciences, Durham University, Durham, UK

Keywords

Cache-oblivious algorithms; Grid traversals; Octree; Quadtree; Space-filling curves; Spacetree; Tree-structured grids

Years and Authors of Summarized Original Work

2009; Weinzierl
2013; Bader

Background

In scientific computing and related fields, mathematical functions are often approximated on meshes where each mesh cell contains a local approximation (e.g., using polynomials) of the represented quantity (density functions, physical quantities such as temperature or pressure, etc.). The grid cells may adaptively refine within areas of high interest or where the applied numerical algorithms demand improved resolution. The resolution even may dynamically change throughout the computation.

In this context, we consider *tree-structured* adaptive meshes, i.e., meshes that result from a recursive subdivision of grid cells. They can be represented via trees – quadtrees or octrees being the most prominent examples. In typical problem settings, quantities are stored on entities (vertices, edges, faces, cells) of the grid. The computation of these variables is usually characterized by local interaction rules and involves variables of adjacent grid cells only. Hence, efficient algorithms are required for the (parallel) traversal of such tree-structured grids and their associated variables.

Problem Definition

Consider a hierarchical mesh of grid cells (triangles, squares, tetrahedra, cubes, etc.), in which all grid cells result from recursively splitting an existing grid cell into a fixed number k of geometrically similar child cells. The resulting grid is equivalent to a tree with uniform degree k. We refer to it as a *spacetree*. Special cases are quadtrees (based upon squares, i.e., dimension $d = 2$, and $k = 4$) and octrees (cubes, $d = 3$, and $k = 8$). Depending on the problem, only the mesh defined by the leaves of the tree may be of interest, or a *multilevel* grid may be considered. The latter also includes all cells corresponding to interior nodes of the tree. Also, meshes resulting from a collection of spacetrees may be considered. Such a generalized data structure is called a *forest* of spacetrees. A mathematical function shall be defined on such a mesh via coefficients that are associated with entities (vertices, edges, faces, cells) of the grid cells. Each coefficient contributes to the representation of the mathematical function in the grid cells adjacent to its entity. For typical computations, we then require efficient algorithms for *mesh traversals* processing all unknowns on entities.

Mesh traversal (definition): Run through all leaf, i.e., unrefined, grid cells and process all

function coefficients associated to each cell or to entities adjacent to it.

Multiscale traversal (definition): Perform one mesh traversal including all (or a certain subset of the) grid cells (tree-interior and leaf cells), thus processing the coarse-grid cells of the grid hierarchy as well.

Mesh traversals may be used to define a sequential order (linearization) on the mesh cells or coefficients. Sequential orders that preserve locality may be used to define partitions for parallel processing and load balancing. Of special interest are algorithms that minimize the memory accesses during traversals as well as the memory required to store the tree-structured grids.

Key Results

Space-Filling Curve Orders on Tree-Structured Grids

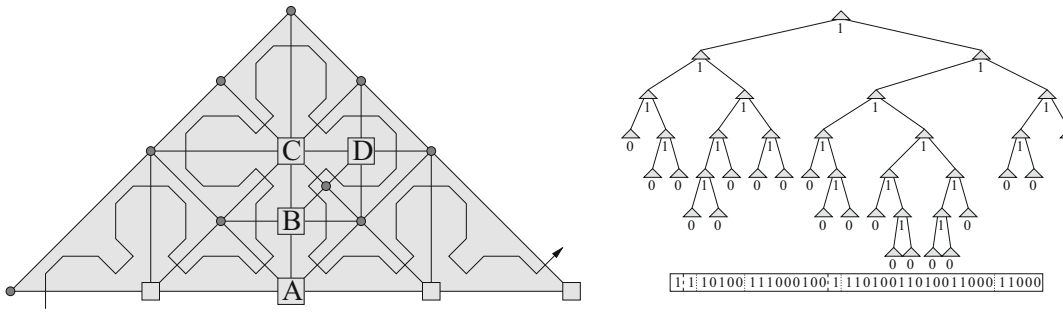
Space-filling curves (SFCs) [1, 5] are continuous surjective mappings from a one-dimensional interval to a higher-dimensional target domain (typically squares, cubes, etc.). They are constructed via an "infinite" recursion process analogous to the generation of tree-structured meshes. Space-filling curves thus induce a sequential order on a corresponding tree-structured grid (an example is given in Fig. 1).

The construction of the curve may be described via a grammar, in which the nonterminals reflect the local orientation of the curve within a grid cell (e.g., Fig. 2). Terminals are used to indicate transfers between grid cells or levels.

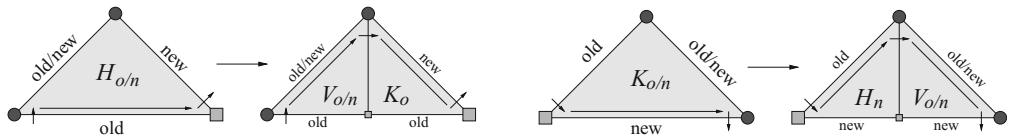
Together with this grammar, a bitstream encoding of the refinement information (as in Fig. 1) provides a minimal-memory data structure to encode a given tree-structured grid. Using Hilbert or Lebesgue (Morton order) SFCs, for example, respective algorithms can be formulated for quadtrees and octrees. Peano curves lead to traversals for (hyper)cube-based spacetrees with 3-refinement along each dimension.

Space-Filling-Curve Traversals

Depth-first traversals of the SFC/bitstream-encoded tree visit all leaf cells of the tree-structured grid in space-filling order (*SFC*



Cache-Oblivious Spacetree Traversals, Fig. 1 Recursively structured triangular mesh and corresponding binary tree with bitstream encoding. The illustrated iteration of a Sierpinski SFC defines a sequential order on the leaf cells (equivalent to a depth-first traversal of the tree) and classifies vertices into groups left (•) and right (□) of the curve. Vertices *A*, *B*, *C*, and *D* are visited in last-in-first-out order during the traversal



Cache-Oblivious Spacetree Traversals, Fig. 2 Replacement rules of a grammar (with six non-terminals $H_{o/n}, K_{o/n}, V_{o/n}$; illustration for $V_{o/n}$ is skipped) to construct the Sierpinski curve. The nonterminals classify for each vertex and edge of a cell, whether it is located left (•) or right (□) of the curve. The *old/new* labels indicate whether the grid cell adjacent to this edge occurs earlier/later in the Sierpinski order

traversal) sequentially. Cell-local data can be held in a stream. All other entities (in 2D: vertices and edges) are to be processed by all adjacent grid cells, i.e., are processed multiple times. For them, a storage scheme for intermediate values or repeated access is required. Figure 1 illustrates that the SFC induces a left/right classification of these entities. During SFC traversals, these entities are accessed in a LIFO fashion, such that intermediate values can be stored on two stacks (left vs. right). Local access rules may be inferred from an augmented grammar (as in Fig. 2). While the left/right classification determines the involved stack, the old/new classification determines whether entities are accessed for the first time during traversal (*first touch*) or have been processed by all adjacent cells (*last touch*). First and last touch trigger loading and storing the respective variables from/onto data streams. These *stack properties* hold for several space-filling curves.

SFC Traversals in the Cache-Oblivious Model

Memory access in SFC traversals is restricted to stack and stream accesses. Random access to memory is entirely avoided. Thus, the number of cache and memory accesses can be accurately described by the I/O or cache-oblivious model (see Cross-References). For the 2D case, assume that for a subtree with K grid cells, the number of edges on the boundary of the respective subgrid is $O(\sqrt{K})$ – which is always satisfied by regularly refined meshes. It can be shown that if we choose K such that all boundary elements fit into cache (size: M words), only boundary edges will cause non-compulsory cache misses [2]. For an entire SFC traversal, the number of cache misses is $O\left(\frac{N}{MB}\right)$, which is asymptotically optimal (B is the number of words per cache line). For adaptively refined meshes, it is an open question what kind of restrictions are posed on the mesh by the $O(\sqrt{K})$ criterion. While it is easy to construct degenerate grids that violate the condition, it is interesting whether grids that result from useful refinement processes (with physically motivated refinement criteria) tend to satisfy the $O(\sqrt{K})$ requirement.

Multiscale Depth-First and Breadth-First Traversals

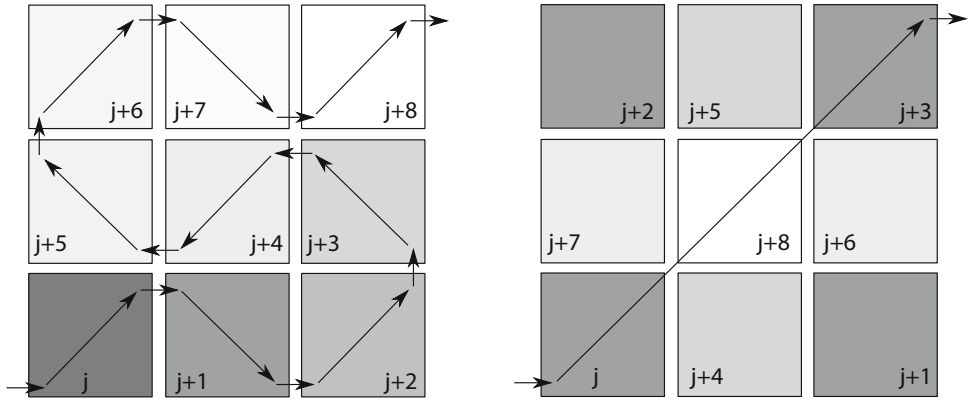
Multiscale traversals find applications in multiphysics simulations, where different physical models are used on the different levels, as well as in (additive) multigrid methods or data analysis, where the different grid levels hold data in different resolutions. If algorithms compute not only results on one level, it is typically sufficient to have two levels available throughout the traversal at one time. Multiscale algorithms then can be constructed recursively.

As variables exist on multiple levels, their (intermediate) access scheme is more elaborate than for a pure SFC traversal. A stack-based management is trivial if we apply one set of stacks per resolution level. Statements on cache obliviousness then have to be weakened, as the maximum number of stacks is not resolution independent anymore. They depend on the number of refinement levels. For depth-first and Peano, $2d + 2$ stacks have been proven to be sufficient (d the spatial dimension). Such a multiscale scheme remains cache oblivious independent of the refinement. It is unknown though doubtful whether schemes for other curves and depth-first traversal exist that allow accesses to unknowns using a resolution-independent number of stacks for arbitrary d.

Toward Parallel Tree Traversals

Data decomposition is the predominant parallelization paradigm in scientific computing: operations are executed on different sets of data in parallel. For distinct sets, the parallelization does not require any synchronization mechanism. For spacetree data structures, distinct pieces of data are given by spacetree cells that do not share a vertex. A parallel data traversal then can be rewritten as a mesh traversal where (in the parallel traversal phases) succeeding cells along the traversal do not share grid entities – a contradiction to connected space-filling curves. For three-partitioning in 2D, such a reordering allows a maximum concurrency level of four (see Fig. 3).

For breadth-first traversals, a reordering is trivial if we drop the space-filling curve ordering and instead reorder all leaves of one level to ob-



Cache-Oblivious Spacetree Traversals, Fig. 3 Peano space-filling curve with numbering on a level $n+1$ (*left*). The ordering then is rearranged to have a concurrency level of four (*right*) illustrated via different shades of gray

tain the highest concurrency level. For depth-first traversals, in contrast, the maximum concurrency level is strictly bounded, even if we drop the SFC paradigm. It remains one for all bipartitioning schemes, is at most 2^d for three-partitioning, and is at most $(\lfloor k/2 \rfloor)^d$ for k-partitioning.

Recursion Unrolling and Parallelism

As concurrency on the cell level is important for many applications, *recursion unrolling* becomes an important technique: (Regular) Subtrees or, more general, fragments along the space-filling curve's depth-first ordering are identified and locally replaced by a breadth-first traversal. This can be done without modifying any data access order on the surfaces of the cut-out curve fragment if the data load and store sequence is preserved along the fragment throughout the unrolling while only computations are reordered. Recursion unrolling then has an impact on the execution overhead as it eliminates recursive function calls and it improves the concurrency of the computations. It can be controlled by an on-the-fly analysis of the tree structure and thus seamlessly integrates into changing grids.

Other Tree-Structured Meshes and Space-Filling Curves

Traversals and stacks can team up only for certain space-filling curves and dimensions:

- In 2D, the stack property is apparently satisfied by all *connected* space-filling curves (for connected SFCs, two contiguous subdomains share an edge). SFC traversals are induced by a grammar that allows a left/right classification. However, no formal proof for this claim has been given yet.
- In 3D and all higher dimensions, the Peano curve is the only connected curve that has been found to satisfy all required properties [6].
- For octree meshes (in 3D), it is an open problem whether SFC traversals exist that can exploit stack properties. Hilbert curves and Lebesgue curves yield data access patterns with spatial and temporal locality but do not provide a stack property.

Applications

Practical applications comprise (parallel) numerical simulations on spacetree meshes that require adaptive refinement and coarsening in each time step or after each iteration [3,4,6]. SFC traversals on spacetrees induce sequential orders that may be exploited to create balanced partitions with favorable quality (due to SFCs being Hölder continuous). Using spacetrees as helper data structures, respective SFC orders can be defined also for entirely unstructured meshes or particle sets.

Space-filling curves are thus a frequently used tool to define parallel partitions.

Cross-References

- ▸ Cache-Oblivious Model
- ▸ I/O-Model

Recommended Reading

1. Bader M (2013) Space-filling curves – an introduction with applications in scientific computing. Texts in computational science and engineering, vol 9. Springer, Heidelberg/New York http://link.springer.com/book/10.1007/978-3-642-31046-1/page/1
2. Bader M, Rahnema K, Vigh CA (2012) Memory-efficient Sierpinski-order traversals on dynamically adaptive, recursively structured triangular grids. In: Jonasson K (ed) Applied parallel and scientific computing – 10th international conference, PARA 2010. Lecture notes in computer science, vol 7134. Springer, Berlin/New York, pp 302–311
3. Burstedde C, Wilcox LC, Ghattas O (2011) `p4est`: scalable algorithms for parallel adaptive mesh refinement on forests of octrees. SIAM J Sci Comput 33(3):1103–1133
4. Mitchell WF (2007) A refinement-tree based partitioning method for dynamic load balancing with adaptively refined grids. J Parallel Distrib Comput 67(4):417–429
5. Sagan H (1994) Space-filling curves. Universitext. Springer, New York
6. Weinzierl T (2009) A framework for parallel PDE solvers on multiscale adaptive Cartesian grids. Dissertation, Institut für Informatik, Technische Universität München, München, http://www.dr.hut-verlag.de/978-3-86853-146-6.html

Canonical Orders and Schnyder Realizers

Stephen Kobourov
Department of Computer Science, University of Arizona, Tucson, AZ, USA

Keywords

Canonical order; Planar graph drawing algorithms; Schnyder realizer

Years and Authors of Summarized Original Work

1990; de Fraysseix, Pach, Pollack
1990; Schnyder

Problem Definition

Every planar graph has a crossings-free drawing in the plane. Formally, a *straight-line drawing* of a planar graph G is one where each vertex is placed at a point in the plane and each edge is represented by a straight-line segment between the two corresponding points such that no two edges cross each other, except possibly at their common end points. A *straight-line grid drawing* of G is a straight-line drawing of G where each vertex of G is placed on an integer grid point. The area for such a drawing is defined by the minimum-area axis-aligned rectangle, or *bounding box*, that contains the drawing.

Wagner in 1936 [12], Fáry in 1948 [5], and Stein in 1951 [10] proved independently that every planar graph has a straight-line drawing. It was not until 1990 that the first algorithms for drawing a planar graph on a grid of polynomial area were developed. The concepts of canonical orders [4] and Schnyder realizers [9] were independently introduced for the purpose of efficiently computing straight-line grid drawings on the $O(n) \times O(n)$ grid. These two seemingly very different combinatorial structures turn out to be closely related and have since been used in many different problems and in many applications.

Key Results

We first describe canonical orders for planar graphs and a linear time procedure to construct them. Then we describe Schnyder realizers and a linear time procedure to compute them. Finally, we show how they can be used to compute straight-line grid drawings for planar graph.

Canonical Order

A planar graph G along with a planar embedding (a cyclic order of the neighbors for each vertex) is

called a *plane graph*. Given a graph G, testing for planarity and computing a planar embedding can be done in linear time [6]. Let G be a maximal plane graph with outer vertices u, v, w in counterclockwise order. Then a *canonical order* or *shelling order* of G is a total order of the vertices $v_1 = u, v_2 = v, v_3, \ldots, v_n = w$ that meets the following criteria for every $4 \leq i \leq n$:

(a) The subgraph $G_{i-1} \subseteq G$ induced by $v_1, v_2, \ldots, v_{i-1}$ is 2-connected, and the boundary of its outerface is a cycle C_{i-1} containing the edge (v_1, v_2).
(b) The vertex v_i is in the outerface of G_{i-1}, and its neighbors in G_{i-1} form a subinterval of the path $C_{i-1} - (u, v)$ with at least two vertices; see Fig. 1a–1b.

Every maximal plane graph G admits a canonical order; see Fig. 1a–1b. Moreover, computing such an order can be done in $O(n)$ time where n is the number of vertices in G. Before proving these claims, we need a simple lemma.

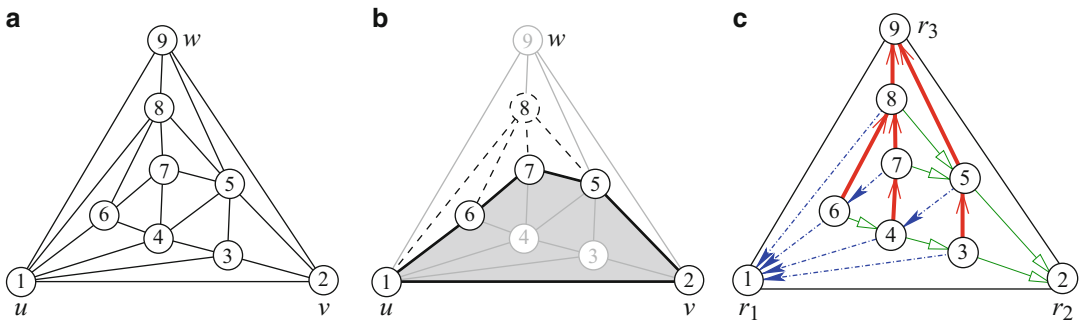
Lemma 1 *Let G be a maximal plane graph with canonical order $v_1, v_2 = v, v_3, \ldots, v_n$. Then for $i \in \{3, \ldots, n-1\}$, any separating pair $\{x, y\}$ of G_i is a chord of C_i.*

The proof of the lemma is simple. Recall that a *chord* of a cycle $\mathcal{C}$ is an edge between nonadjacent vertices of $\mathcal{C}$. Since G is a maximal plane graph and since each vertex $v_j, j \in \{n, \ldots, i+1\}$ is in the outerface of G_j, all the internal faces of G_i are triangles. Adding a dummy vertex d along with edges from d to each vertex on the outerface of G_i yields a maximal plane graph G'. Then G' is 3-connected, and for each separation pair $\{x, y\}$ of G, the set $T = \{x, y, d\}$ is a separating set of G' since $G = G' \setminus d$. The set T is a separating triangle in G' [1], and therefore, the edge (x, y) is a chord on C_i.

Theorem 1 *A canonical order of a maximal plane graph G can be computed in linear time.*

This is also easy to prove. If the number of vertices n in G is 3, then the canonical ordering of G is trivially defined. Let $n > 3$ and choose the vertices $v_n = w, v_{n-1}, \ldots, v_3$ in this order so that conditions (a)–(b) of the definition are satisfied. Since G is a maximal plane graph, G is 3-connected, and hence, $G_{n-1} = G \setminus w$ is 2-connected. Furthermore, the set of vertices adjacent to $v_n = w$ forms a cycle C_{n-1}, which is the boundary of the outerface of G_{n-1}. Thus, conditions (a)–(b) hold for $k = n$.

Assume by induction hypothesis that the vertices $v_n, v_{n-1}, \ldots, v_{i+1}, i \geq 3$ have been appropriately chosen. We now find the next vertex v_i. If we can find a vertex x on C_i, which is not an end vertex of a chord, then we can choose $v_k = x$. Indeed, if deleting x from G_i violated the 2-connectivity, then the cut vertex y of $G_i - x$, together with x, would form a separating pair for G_i, and hence, (x, y) would be a chord in G_i (from the lemma above). We now show that we can find a vertex v_i on C_i which is not an end vertex of a chord.



Canonical Orders and Schnyder Realizers, Fig. 1 (**a**) A canonical order of vertices for a maximal plane graph G, (**b**) insertion of v_8 in G_7, (**c**) a Schnyder realizer for G

If there is no chord of C_i, then we can choose any vertex of C_i other than u and v as v_k. Otherwise, label the vertices of $C_i - \{(u, v)\}$ by $p_1 = u, p_2, \ldots, p_t = v$ consecutively from u to v. By definition, any chord $(p_k, p_l), k < l$ must have $k < l-1$. We say that a chord $(p_k, p_l), k < l$, includes another chord $(p_{k'}, p_{l'}), k' < l'$, if $k \leq k' < l' \leq l$. Then take an inclusion-minimal chord (p_k, p_l) and any vertex p_j for $k < j < l$ can be chosen as v_i. Since v_i is not an end vertex of a chord for $C_{i-1}, G_{i-1} = G_i \setminus v_i$ remains 2-connected. Furthermore, due to the maximal planarity of G, the neighborhood of v_i on C_{i-1} forms a subinterval for $C_{i-1} - (u, v)$.

The algorithm, implicit in the above argument, can be implemented to run in linear time by keeping a variable for each vertex x on C_i, counting the number of chords x is incident to. After each vertex v_i is chosen, the variables for all its neighbors can be updated in $O(\deg(v_i))$ time. Summing over all vertices in the graph leads to an overall linear running time [2], and this concludes the proof of the theorem.

Schnyder Realizer

Let G be a maximal plane graph. A *Schnyder realizer* $\mathcal{S}$ of G is a partition of the internal edges of G into three sets T_1, T_2, and T_3 of directed edges, so that for each interior vertex v, the following conditions hold:

(a) v has outdegree exactly one in each of T_1, T_2, and T_3.
(b) The clockwise order of edges incident to v is outgoing T_1, incoming T_2, outgoing T_3, incoming T_1, outgoing T_2, and incoming T_3; see Fig. 1c.

Since a maximal plane graph has exactly $n - 3$ internal vertices and exactly $3n - 9$ internal edges, the three outgoing edges for each internal vertex imply that all the edges incident to the outer vertices are incoming. In fact, these two conditions imply that for each outer vertex $r_i, i = 1, 2, 3$, the incident edges belong to the same set, say T_i, where r_1, r_2, r_3 are in counterclockwise order around the outerface and each set of edges T_i forms a directed tree, spanning all the internal vertices and one external vertex r_i, oriented towards r_i [3]; see Fig. 1c. Call r_i the root of T_i for $i = 1, 2, 3$.

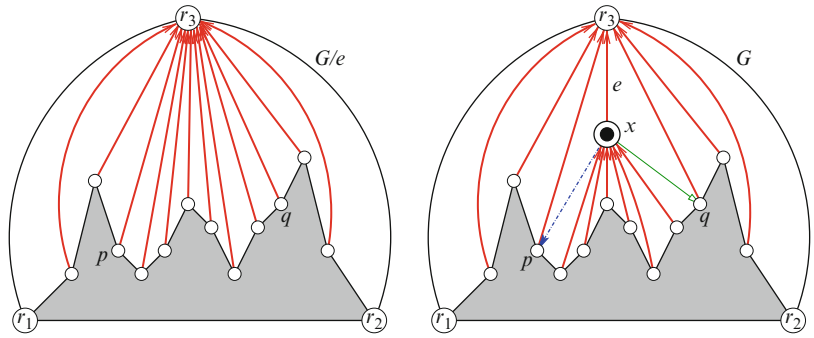
Note that the existence of a decomposition of a maximal planar graph G into three trees was proved earlier by Nash-Williams [8] and by Tutte [11]. Kampen [7] showed that these three trees can be oriented towards any three specified root vertices r_1, r_2, r_3 of G so that each vertex other than these roots has exactly one outgoing edges in each tree. Schnyder [9] proved the existence of the special decomposition defined above, along with a linear time algorithm to compute it. Before we describe the algorithm, we need to define the operation of *edge contraction*. Let G be a graph and $e = (x, y)$ be an edge of G. Then we denote by G/e, the simple graph obtained by deleting x, y and all their incident edges from G, adding a new vertex z and inserting an edge (z, v) for each vertex v that is adjacent to either x or y in G. Note that for a maximal plane graph G, contracting an edge $e = (x, y)$ yields a maximal plane graph if and only if there are exactly two common neighbors of x and y. Two end vertices of an edge $e = (x, y)$ have exactly two common neighbors if and only if the edge e is not on the boundary of a separating triangle.

Lemma 2 *Let G be a maximal plane graph with at least 4 vertices, where u is an outer vertex of G. Then there exists an internal vertex v of G such that (u, v) is an edge in G and vertices u and v have exactly two common neighbors.*

This is easy to prove. If G has exactly 4 vertices, then it is K_4 and the internal vertex of G has exactly two common neighbors. Consider graph G with more than 4 vertices. If u is not on the boundary of any separating triangle, then taking any neighbor of u as v is sufficient. Else, if u is on the boundary of a separating triangle Δ, we can find a desired vertex v by induction on the subgraph of G inside Δ.

Theorem 2 *A Schnyder realizer of a maximal plane graph G can be computed in linear time.*

The proof of the theorem is by induction. If G has exactly 3 vertices, its Schnyder realizer is computed trivially. Consider graph G with



Canonical Orders and Schnyder Realizers, Fig. 2 Computing a Schnyder realizer of a maximal plane graph G from that of G/e

more than 3 vertices. Let r_1, r_2, and r_3 be the three outer vertices in counterclockwise order. Then by the above lemma, there is an internal vertex x in G and edge $e = (r_3, x)$ so that r_3 and x have exactly two common neighbors. Let $G' = G/e$. Then by the induction hypothesis, G' has a Schnyder realizer with the three trees T_1, T_2, and T_3, rooted at r_1, r_2, and r_3. We now modify this to find a Schnyder realizer for G. The orientation and partitioning of all the edges not incident to x remain unchanged from G/e. Among the edges incident to x, we add e to T_3, oriented towards r_3. We add the two edges that are just counterclockwise of e and just clockwise of e in the ordering around x, to T_1 and T_2, respectively, both oriented away from x. Finally we put all the remaining edges in T_3, oriented towards x; see Fig. 2. It is now straightforward to check that these assignment of edges to the trees satisfy the two conditions. The algorithm implicit in the proof can be implemented in linear time, given the edge contraction sequence. The edge contraction sequence itself can be computed in linear time by taking the reverse order of a canonical order of the vertices and in every step contracting the edge between r_3 and the current vertex in this order.

Drawing Planar Graphs

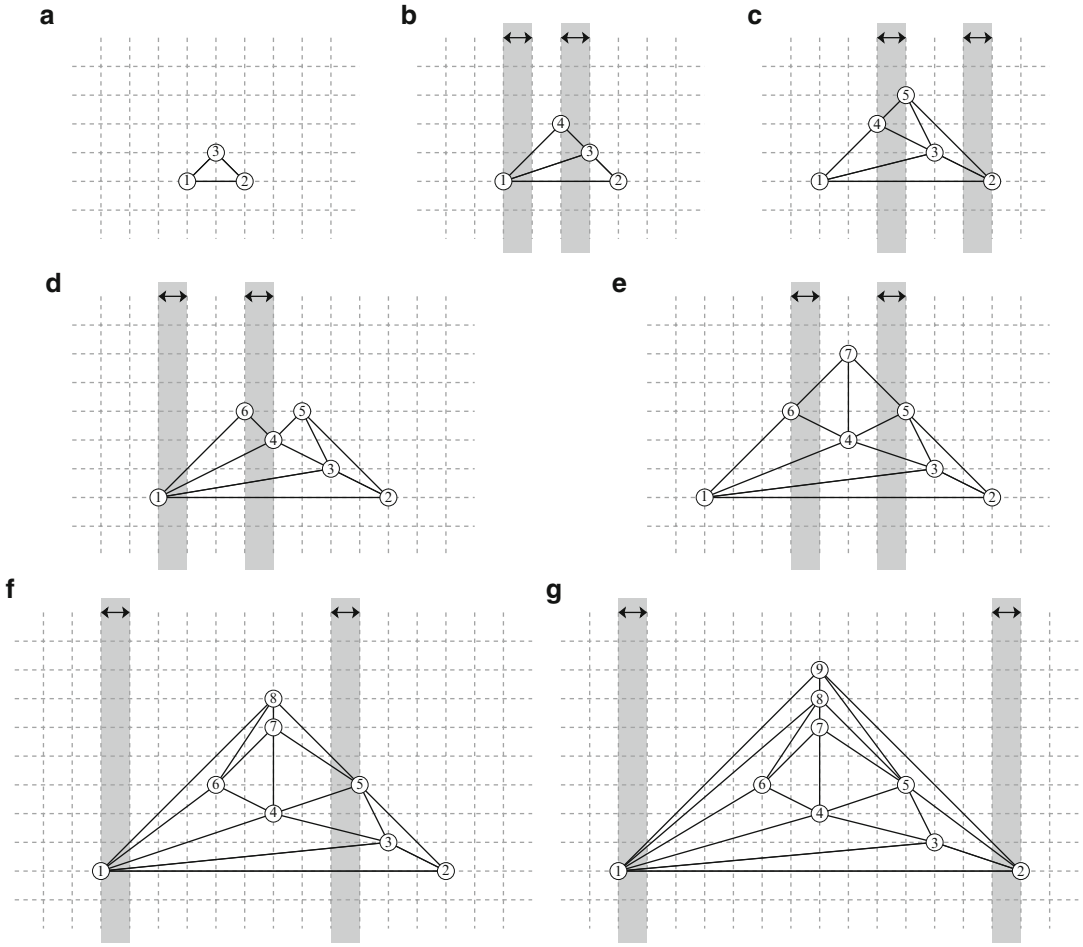
We now show how canonical orders and Schnyder realizers can be used to compute straight-line grid drawings of maximal plane graphs.

Theorem 3 *Let G be a maximal plane graph with n vertices. A straight-line grid drawing of G on the $(2n-4) \times (n-2)$ grid can be computed in linear time.*

This is a constructive proof. Let $\mathcal{O} = v_1, \ldots, v_n$ be a canonical order of G, G_i, the subgraph of G induced by the vertices $v_1, \ldots, v_i$, and C_i the boundary of the outerface of $G_i, i = 3, 4, \ldots, n$. We incrementally obtain straight-line drawing Γ_i of G_i for $i = 3, 4, \ldots, n$. We also maintain the following invariants for Γ_i:

(i) The x-coordinates of the vertices on the path $C_i \setminus \{(v_1, v_2)\}$ are monotonically increasing as we go from v_1 to v_2.
(ii) Each edge of the path $C_i - \{(v_1, v_2)\}$ is drawn with slope 1 or -1.

We begin with G_3, giving v_1, v_2, and v_3 coordinates $(0, 0)$, $(2, 0)$, and $(1, 1)$; the drawing Γ_3 satisfies conditions (i)–(ii); see Fig. 3a. Suppose the drawing for Γ_{i-1} for some $i > 3$ has already been computed; we now show how to obtain Γ_i. We need to add vertex v_i and its incident edges in G_i to Γ_{i-1}. Let $w_1 = v_1, \ldots, w_t = v_2$ be the vertices on $C_{i-1} \setminus \{(v_1, v_2)\}$ in this order from v_1 to v_2. By the property of canonical orders, v_i is adjacent to a subinterval of this path. Let $w_l, \ldots, w_r, 1 \le l < r \le t$, be the vertices adjacent to v_i, in this order. We want to place v_i at the intersection point p between the straight line from w_l with slope 1 and the straight line from w_r with slope -1. Note that by condition (ii),

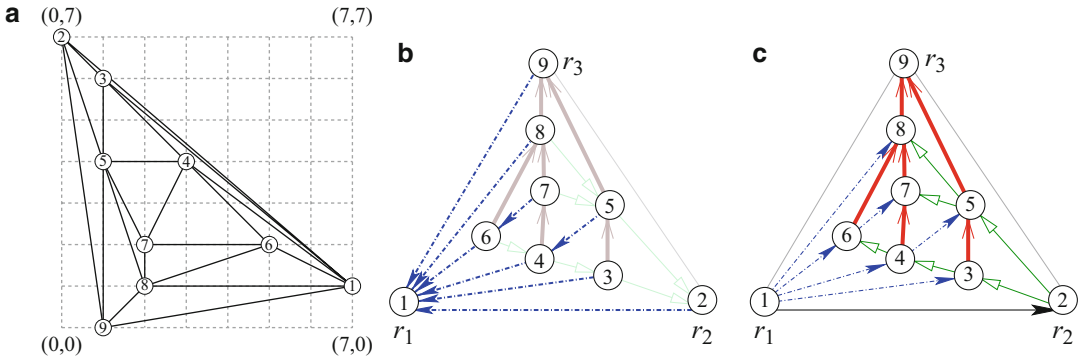


Canonical Orders and Schnyder Realizers, Fig. 3 Illustration for the straight-line drawing algorithm using canonical order

the two vertices w_l and w_r are at even Manhattan distance in Γ, and hence, point p is a grid point. However, if we place v_i at p, then the edges $v_i w_l$ and $v_i w_r$ might overlap with the edges $w_l w_{l+1}$ and $w_{r-1} w_r$, since they are drawn with slopes 1 or -1. We thus shift all the vertices to the left of w_l in Γ_{i-1} (including w_l) one unit to the left at all the vertices to the right of w_r in Γ_{i-1} (including w_r) one unit to the right; see Fig. 3.

Consider a rooted tree T, spanning all the internal vertex of G along with one external vertex v_n, where v_n is the root of T and for each internal vertex x, the parent of x is the highest numbered successor in G. (Later we see that T can be one of the three trees in a Schnyder realizer of G.) For any internal vertex x of G, denote by $U(x)$ the set of vertices that are in the subtree of T rooted at v (including v itself). Then the shifting of the vertices above can be obtained by shifting the vertices in $U(w_i), i = 1, \ldots, l$ one unit to the left and the vertices in $U(w_i), i = r, \ldots, t$ one unit to the right. After these shifts, v_i can be safely placed at the intersection of the line with slope 1 from w_l and the line with slope -1 from w_r. Note that this drawing satisfies conditions (i)–(ii). This algorithm can be implemented in linear time, even though the efficient vertex shifting requires careful relative offset computation [2].

The next theorem shows how Schnyder realizers can be used to compute a straight-line grid



Canonical Orders and Schnyder Realizers, Fig. 4 (**a**) A straight-line drawing for the graph in Fig. 1 using Schnyder realizer, (**b**)–(**c**) computation of a canonical order from a Schnyder realizer

drawing of a plane graph. Let T_1, T_2, and T_3 be the trees in a Schnyder realizer of G, rooted at outer vertices r_1, r_2, and r_3. Since each internal vertex v of G has exactly one outgoing edge in each of the trees, there is a directed path $P_i(v)$ in each of the three trees T_i from v to r_i. These three paths $P_1(v), P_2(v)$, and $P_3(v)$ are vertex disjoint except for v, and they define three regions $R_1(v), R_2(v)$, and $R_3(v)$ for v. Here $R_i(v)$ is the region between the two paths $P_{i-1}(v)$ and $P_{i+1}(v)$, where the addition and subtraction are modulo 3. Let $\eta_i(v)$ denote the number of vertices in $R_i(v) \setminus P_{i-1}(v)$, where $i = 1, 2, 3$ and the subtraction is modulo 3. Extend these definitions to the outer vertices as follows: $\eta_i(r_i) = n - 2, \eta_{i+1}(r_i) = 1, \eta_{i-1}(r_i) = 0$.

Theorem 4 *The coordinates* $((\eta_1(v), \eta_2(v)))$ *for each vertex* v *in* G *give a straight-line drawing* Γ *of* G *on a grid of size* $(n-2) \times (n-2)$.

Place each vertex v at the point with coordinates $(\eta_1(v), \eta_2(v)$, and $\eta_3(v))$. Since $\eta(v) = \eta_1(v) + \eta_2(v) + \eta_3(v)$ counts the number of vertices in all the three regions of v, each vertex of G except v is counted exactly once in $\eta(v)$. Thus, $\eta_1(v) + \eta_2(v) + \eta_3(v) = n - 1$ for each vertex v. Thus, the drawing Γ' obtained by these coordinates $(\eta_1(v), \eta_2(v), \eta_3(v))$ lies on the plane $x + y + z = n - 1$. Furthermore, the drawing does not induce any edge crossings; see [9]. Thus, Γ' is a straight-line drawing of G on the plane $x + y + z = n - 1$. Then Γ is just a projection of Γ' on the plane $z = 0$ and hence is planar. Since each coordinate in the drawing is bounded between 0 and $n - 1$, the area is at most $(n-2) \times (n-2)$; see Fig. 4a.

Equivalency of Canonical Orders and Schnyder Realizers

Here we show that canonical orders and Schnyder realizers are in fact equivalent in the sense that a canonical order of a graph defines a Schnyder realizer and vice versa [3].

Lemma 3 *A canonical order for a maximal plane graph* G *defines a unique Schnyder realizer where the three parents for each vertex* v *of* G *are its leftmost predecessor, its rightmost predecessor, and its highest-labeled successor.*

See Fig. 1a, 1c for a canonical order $\mathcal{O}$ and the corresponding Schnyder realizer $\mathcal{S}$ defined by $\mathcal{O}$ for a maximal plane graph. One can easily verify that this definition of $\mathcal{S}$ satisfies the two conditions for each internal vertex for a maximal plane graph. A canonical order $\mathcal{O}$ and the Schnyder realizer $\mathcal{S}$ obtained from $\mathcal{O}$ for a maximal plane graph are said to be *compatible*.

We now describe two ways to obtain a canonical order from a Schnyder realizer $\mathcal{S}$. In both cases, we obtain a canonical order which is compatible with $\mathcal{S}$.

Lemma 4 *Let* G *be a maximal plane graph with outer vertices* r_1, r_2, *and* r_3 *in counterclockwise order and let* T_1, T_2, *and* T_3 *be the three trees in a Schnyder realizer of* G *rooted at* r_1, r_2, *and* r_3. *Then a compatible canonical order of* G *can be obtained as follows:*

1. *By taking the counterclockwise depth-first traversal order of the vertices in the graph* $T_1 \cup \{(r_2, r_1), (r_3, r_1)\}$*; see Fig. 4b.*
2. *By taking the topological order of the directed acyclic graph* $T_1^{-1} \cup T_2^{-1} \cup T_3$*, where* $T_i^{-1}, i = 1, 2$ *is the Schnyder tree* T_i *with reversed edge directions; see Fig. 4c.*

It is not difficult to show that the directed graph $T_1^{-1} \cup T_2^{-1} \cup T_3$ is in fact acyclic [3, 9]. Then it is easy to verify that the canonical orders obtained from a Schnyder realizer $\mathcal{S}$ are compatible with $\mathcal{S}$; i.e., defining a Schnyder realizer from them by Lemma 3 produces the original Schnyder realizer $\mathcal{S}$.

Cross-References

- Bend Minimization for Orthogonal Drawings of Plane Graphs
- Convex Graph Drawing
- Force-Directed Graph Drawing
- Planarity Testing

Recommended Reading

1. Baybars I (1982) On k-path hamiltonian maximal planar graphs. Discret Math 40(1):119–121
2. Chrobak M, Payne TH (1995) A linear-time algorithm for drawing a planar graph on a grid. Inf Process Lett 54(4):241–246
3. de Fraysseix H, de Mendez PO (2001) On topological aspects of orientations. Discret Math 229(1–3):57–72
4. de Fraysseix H, Pach J, Pollack R (1990) How to draw a planar graph on a grid. Combinatorica 10(1):41–51
5. Fáry I (1948) On straight lines representation of planar graphs. Acta Scientiarum Mathematicarum 11:229–233
6. Hopcroft JE, Tarjan RE (1974) Efficient planarity testing. J ACM 21(4):549–568
7. Kampen G (1976) Orienting planar graphs. Discret Math 14(4):337–341
8. Nash-Williams CSJA (1961) Edge-disjoint spanning trees of finite graphs. J Lond Math Soc 36:445–450
9. Schnyder W (1990) Embedding planar graphs on the grid. In: ACM-SIAM symposium on discrete algorithms (SODA 1990), San Francisco, pp 138–148
10. Stein SK (1951) Convex maps. Am Math Soc 2(3):464–466
11. Tutte WT (1961) On the problem of decomposing a graph into n connected factors. J Lond Math Soc 36:221–230
12. Wagner K (1936) Bemerkungen zum Vierfarbenproblem. Jahresbericht der Deutschen Mathematiker-Vereinigung 46:26–32

Causal Order, Logical Clocks, State Machine Replication

Xavier Défago
School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, Japan

Keywords

State-machine replication: active replication

Years and Authors of Summarized Original Work

1978; Lamport

Problem Definition

This entry covers several problems, related with each other. The first problem is concerned with maintaining the causal relationship between events in a distributed system. The motivation is to allow distributed systems to reason about time with no explicit access to a physical clock. Lamport [5] defines a notion of logical clocks that can be used to generate timestamps that are consistent with causal relationships (in a conservative sense). He illustrates logical clocks (also called Lamport clocks) with a distributed mutual exclusion algorithm. The algorithm turns out to be an illustration of state-machine replication. Basically, the algorithm generates a total ordering of the events that is consistent across processes. With all processes starting in the same state, they evolve consistently with no need for further synchronization.

System Model

The system consists of a collection of processes. Each process consists of a sequence of events.

Processes have no shared memory and communicate only by exchanging messages. The exact definition of an event depends on the system actually considered and the abstraction level at which it is considered. One distinguishes between three kinds of events: internal (affects only the process executing it), send, and receive events.

Causal Order

Causal order is concerned with the problem that the occurrence of some events may affect other events in the future, while other events may not influence each other. With processes that do not measure time, the notion of simultaneity must be redefined in such a way that simultaneous events are those that cannot possibly affect each other. For this reason, it is necessary to define what it means for an event to happen before another event.

The following "happened before" relation is defined as an irreflexive partial ordering on the set of all events in the system [5].

Definition 1 The relation "$\rightarrow$" on the set of events of a system is the smallest relation satisfying the following three conditions:

1. If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.
2. If a is the sending of a message by one process and b is the receipt of the same message by another process, then $a \rightarrow b$.
3. If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$.

Definition 2 Two distinct events a and b are said to be *concurrent* if $a \not\rightarrow b$ and $b \not\rightarrow a$.

Logical Clocks

Lamport also defines clocks in a generic way, as follows.

Definition 3 A clock C_i for a process p_i is a function which assigns a number $C_i\langle a\rangle$ to any event a on that process. The entire system of clocks is represented by the function C which assigns to any event b the number $C\langle b\rangle$, where $C\langle b\rangle = C_j\langle b\rangle$ if b is an event in process p_j. The system of clocks must meet the following *clock condition*.

- For any events a and b, if $a \rightarrow b$ then $C\langle a\rangle < C\langle b\rangle$.

Assuming that there is some arbitrary total ordering $\prec$ of the processes (e.g., unique names ordered lexicographically), Lamport extends the "happened before" relation and defines a relation "$\Rightarrow$" as a total ordering on the set of all events in the system.

Definition 4 The total order relation $\Rightarrow$ is defined as follows. If a is an event in process p_i and b is an event in process p_j, then $a \Rightarrow b$ if and only if either one of the following conditions is satisfied.

1. $C_i\langle a\rangle < C_j\langle b\rangle$
2. $C_i\langle a\rangle = C_j\langle b\rangle$ and $p_i \prec p_j$.

In fact, Lamport [5] also discusses an adaptation of these conditions to physical clocks, and provides a simple clock synchronization algorithm. This is however not discussed further here.

State Machine Replication

The problem of state-machine replication was originally presented by Lamport [4, 5]. In a later review of the problem, Schneider [8] defines the problem as follows (formulation adapted to the context of the entry).

Problem 1 (State-machine replication)
INPUT: A set of concurrent requests.
OUTPUT: A sequence of the requests processed at each process, such that:

1. *Replica coordination*: all replicas receive and process the same sequence of requests.
2. *Agreement*: every non-faulty state-machine replica receives every request.
3. *Order*: every non-faulty state-machine replica processes the requests it receives in the same relative order.

In his paper on logical time [5] and discussed in this entry, Lamport does not consider failures. He does however consider them in another paper on

state-machine replication for fault-tolerance [4], which he published the same year.

Key Results

Lamport [5] proposed many key results related to the problems described above.

Logical Clocks

Lamport [5] defines an elegant system of logical clocks that meets the clock condition presented in Definition 3. The clock of a process p_i is represented by a register C_i, such that $C_i\langle a\rangle$ is the value held by C_i when a occurs. Each message m carries a timestamp T_m, which equals the time at which m was sent. The clock system can be described in terms of the following rules.

1. Each process p_i increments C_i between any two successive events.
2. If event a is the sending of a message m by process p_i, then the message m contains a timestamp $T_m = C_i\langle a\rangle$.
3. Upon receiving a message m, process p_j sets C_j to $\max(C_j, T_m + 1)$ (before actually executing the receive event).

State Machine Replication

As an illustration for the use of logical clocks, Lamport [5] describes a mutual exclusion algorithm. He also mentions that the approach is more general and discusses the concept of state-machine replication that he refines in a different paper [4].

The mutual exclusion algorithm is based on the idea that every process maintains a copy of a request queue, and the algorithm ensures that the copies remain consistent across the processes. This is done by generating a total ordering of the request messages, according to timestamps obtained from the logical clocks of the sending processes.

The algorithm described works under the following simplifying assumptions:

- Every message that is sent is eventually received.
- For any processes p_i and p_j, messages from p_i to p_j are received in the same order as they are sent.
- A process can send messages directly to every other processes.

The algorithm requires that each process maintains its own request queue, and ensures that the request queues of different processes always remain consistent. Initially, request queues contain a single message $(T_0, p_0, \mathit{request})$, where p_0 is the process that holds the resource and the timestamp T_0 is smaller than the initial value of every clock. Then, the algorithm works as follows.

1. When a process p_i requests the resource, it sends a request message $(T_m, p_i, \mathit{request})$ to all other processes and puts the message in its request queue.
2. When a process p_j receives a message $(T_m, p_i, \mathit{request})$, it puts that message in its request queue and sends an acknowledgment $(T_{m'}, p_j, \mathit{ack})$ to p_i.
3. When a process p_i releases the resource, it removes all instances of messages $(-, p_i, \mathit{request})$ from its queue, and sends a message $(T_{m'}, p_i, \mathit{release})$ to all other processes.
4. When a process p_j receives a release message from process p_i, it removes all instances of messages $(-, p_i, \mathit{request})$ from its queue, and sends a timestamped acknowledgment to p_i.
5. Messages in the queue are sorted according to the total order relation $\Rightarrow$ of Definition 4. A process p_i can use the resource when (a) a message $(T_m, p_i, \mathit{request})$ appears first in the queue, and (b) p_i has received from all other processes a message with a timestamp greater than T_m (or equal from any process p_j where $p_i \prec p_j$).

Applications

A brief overview of some applications of the concepts presented in this entry has been provided.

First of all, the notion of causality in distributed systems (or lack thereof) leads

to a famous problem in which a user may potentially see an answer before she can see the relevant question. The time-independent characterization of causality of Lamport lead to the development of efficient solutions to enforce causal order in communication. In his later work, Lamport [3] gives a more general definition to the "happened before" relation, so that a system can be characterized at various abstraction levels.

About a decade after Lamport's work on logical clock, Fidge [2] and Mattern [6] have developed the notion of vector clocks, with the advantage of a complete characterization of causal order. Indeed, the clock condition enforced by Lamport's logical clocks is only a one-way implication (see Definition 3). In contrast, vector clocks extend Lamport clocks by ensuring that, for any events a and b, if $C\langle a\rangle < C\langle b\rangle$, then $a \rightarrow b$. This is for instance useful for choosing a set of checkpoints after recovery of a distributed system, for distributed debugging, or for deadlock detection. Other extensions of logical time have been proposed, that have been surveyed by Raynal and Singhal [7].

The state-machine replication also has many applications. In particular, it is often used for replicating a distributed service over several processors, so that the service can continue to operate even in spite of the failure of some of the processors. State-machine replication ensures that the different replicas remain consistent.

The mutual exclusion algorithm proposed by Lamport [5] and described in this entry is actually one of the first known solution to the *atomic broadcast* problem (see relevant entry). Briefly, in a system with several processes that broadcast messages concurrently, the problem requires that all processes deliver (and process) all message in the same order. Nowadays, there exist several approaches to solving the problem. Surveying many algorithms, Défago et al. [1] have classified Lamport's algorithm as *communication history* algorithms, because of the way the ordering is generated.

Cross-References

- ▸ Atomic Broadcast
- ▸ Clock Synchronization
- ▸ Concurrent Programming, Mutual Exclusion
- ▸ Linearizability
- ▸ Quorums

Recommended Reading

1. Défago X, Schiper A, Urbán P (2004) Total order broadcast and multicast algorithms: taxonomy and survey. ACM Comput Surv 36:372–421
2. Fidge CJ (1991) Logical time in distributed computing systems. IEEE Comput 24:28–33
3. Lamport L (1986) On interprocess communication. Part I: basic formalism. Distrib Comput 1:77–85
4. Lamport L (1978) The implementation of reliable distributed multiprocess systems. Comput Netw 2:95–114
5. Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21:558–565
6. Mattern F (1989) Virtual time and global states of distributed systems. In: Cosnard M, Quinton P (eds) Parallel and distributed algorithms. North-Holland, Amsterdam, pp 215–226
7. Raynal M, Singhal M (1996) Capturing causality in distributed systems. IEEE Comput 29:49–56
8. Schneider FB (1990) Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput Surv 22:299–319

Certificate Complexity and Exact Learning

Lisa Hellerstein
Department of Computer Science and Engineering, NYU Polytechnic School of Engineering, Brooklyn, NY, USA

Keywords

Boolean formulae; Cerfiticates; Computational learning; Exact learning; Query complexity

Years and Authors of Summarized Original Work

1995; Hellerstein, Pilliapakkamnatt, Raghavan, Wilkins

Problem Definition

This problem concerns the query complexity of proper learning in a widely studied learning model: exact learning with membership and equivalence queries. Hellerstein et al. [10] showed that the number of (polynomially sized) queries required to learn a concept class in this model is closely related to the size of certain certificates associated with that class. This relationship gives a combinatorial characterization of the concept classes that can be learned with polynomial query complexity. Similar results were shown by Hegedüs based on the work of Moshkov [8, 13].

The Exact Learning Model

Concepts are functions $f : X \to \{0, 1\}$ where X is an arbitrary domain. In exact learning, there is a hidden concept f from a known class of concepts C, and the problem is to exactly identify the concept f.

Algorithms in the exact learning model obtain information about f, the target concept, by querying two oracles, a membership oracle and an equivalence oracle. A membership oracle for f answers membership queries (i.e., point evaluation queries), which are of the form "What is $f(x)$?" where $x \in X$. The membership oracle responds with the value $f(x)$. An equivalence oracle for f answers equivalence queries, which are of the form "Is $h \equiv f$?" where h is a representation of a concept defined on the domain X. Representation h is called a hypothesis. The equivalence oracle responds "yes" if $h(x) = f(x)$ for all $x \in X$. Otherwise, it returns a counterexample, a value $x \in X$ such that $f(x) \neq h(x)$.

The exact learning model is due to Angluin [2]. Angluin viewed the combination of membership and equivalence oracles as constituting a "minimally adequate teacher." Equivalence queries can be simulated both in Valiant's well-known PAC model, and in the online mistake-bound learning model.

Let R be a set of representations of concepts, and let C_R be the associated set of concepts. For example, if R were a set of DNF formulas, then C_R would be the set of Boolean functions (concepts) represented by those formulas. An exact learning algorithm is said to learn R if, given access to membership and equivalence oracles for any f in C_R, it ends by outputting a hypothesis h that is a representation of f.

Query Complexity of Exact Learning

There are two aspects to the complexity of exact learning: query complexity and computational complexity. The results of Hellerstein et al. concern query complexity.

The query complexity of an exact learning algorithm measures the number of queries it asks and the size of the hypotheses it uses in those queries (and as the final output). We assume that each representation class R has an associated size function that assigns a nonnegative number to each $r \in R$. The size of a concept c with respect to R, denoted by $|c|_R$, is the size of the smallest representation of c in R; if $c \notin C_R$, $|c|_R = \infty$. Ideally, the query complexity of an exact learning algorithm will be polynomial in the size of the target and other relevant parameters of the problem.

Many exact learning results concern learning classes of representations of Boolean functions. Algorithms for learning such classes R are said to have polynomial query complexity if the number of hypotheses used, and the size of those hypotheses, is bounded by some polynomial $p(m, n)$, where n is the number of variables on which the target f is defined, and $m = |f|_R$. We assume that algorithms for learning Boolean representation classes are given the value of n as input.

Since the number and size of queries used by an algorithm are a lower bound on the time taken by that algorithm, query complexity lower bounds imply computational complexity lower bounds.

Improper Learning and the Halving Algorithm

An algorithm for learning a representation class R is said to be proper if all hypotheses used in its equivalence queries are from R, and it outputs a representation from R. Otherwise, the algorithm is said to be improper.

When C_R is a finite concept class, defined on a finite domain X, a simple, generic algorithm called the halving algorithm can be used to exactly learn R using $\log |C_R|$ equivalence queries and no membership queries. The halving algorithm is based on the following idea. For any $V \subseteq C_R$, define the majority hypothesis MAJ_V to be the concept defined on X such that for all $x \in X$, $\mathrm{MAJ}_V(x) = 1$ if $g(x) = 1$ for more than half the concepts g in V, and $\mathrm{MAJ}_V(x) = 0$ otherwise. The halving algorithm begins by setting $V = C_R$. It then repeats the following:

1. Ask an equivalence query with the hypothesis MAJ_V.
2. If the answer is yes, then output MAJ_V.
3. Otherwise, the answer is a counterexample x. Remove from V all g such that $g(x) = \mathrm{MAJ}_V(x)$.

Each counterexample eliminates the majority of the elements currently in V, so the size of V is reduced by a factor of at least 2 with each equivalence query. It follows that the algorithm cannot ask more than $\log_2 |C_R|$ queries.

The halving algorithm cannot necessarily be implemented as a proper algorithm, since the majority of hypotheses may not be representable in C_R. Even when they are representable in C_R, the representations may be exponentially larger than the target concept.

Proper Learning and Certificates

In the exact model, the query complexity of proper learning is closely related to the size of certain certificates.

For any concept f defined on a domain X, a certificate that f has property P is a subset $S \subseteq X$ such that for all concepts g defined on X, if $g(x) = f(x)$ for all $x \in S$, then g has property P. The size of the certificate S is $|S|$, the number of elements in it.

We are interested in properties of the form "g is not a member of the concept class C." To take a simple example, let D be the class of constant-valued n-variable Boolean functions, i.e., D consists of the two functions $f_1(x_1, \ldots, x_n) = 1$ and $f_2(x_1, \ldots, x_n) = 0$. Then if g is an n-variable Boolean function that is not a member of D, a certificate that g is not in C could be just a pair $a \in \{0,1\}^n$ and $b \in \{0,1\}^n$ such that $g(a) = 1$ and $g(b) = 0$.

For C a class of concepts defined on X define the exclusion dimension of C to be the maximum, over all concepts g not in C, of the size of the smallest certificate that g is not in C. Let $XD(C)$ denote the exclusion dimension of C. In the above example, $XD(C) = 2$.

Key Results

Theorem 1 *Let R be a finite class of representations. Then there exists a proper learning algorithm in the exact model that learns C using at most $XD(C) \log |C|$ queries. Further, any such algorithm for C must make at least $XD(C)$ queries.*

Independently, Hegedüs proved a theorem that is essentially identical to the above theorem. The algorithm in the theorem is a variant of the ordinary halving algorithm. As noted by Hegedüs, a similar result to Theorem 1 was proved earlier by Moshkov, and Moshkov's techniques can be used to improve the upper bound by a factor of $\frac{2}{\log_2 XD(C)}$.

An extension of the above result characterizes the representation classes that have polynomial query complexity. The following theorem presents the extended result as it applies to representation classes of Boolean functions.

Theorem 2 *Let R be a class of representations of Boolean functions. Then there exists a proper*

learning algorithm in the exact model that learns R with polynomial query complexity iff there exists a polynomial $p(m,n)$ such that for all $m,n > 0$, and all n-variable Boolean functions g, if $|g|_R > m$, then there exists a certificate of size at most $p(m,n)$ proving that $|g|_R > m$.

A concept class having certificates of the type specified in this theorem is said to have polynomial certificates.

The algorithm in the above theorem does not run in polynomial time. Hellerstein et al. give a more complex algorithm that runs in polynomial time using a $\Sigma_4{}^P$ oracle, provided R satisfies certain technical conditions. Köbler and Lindner subsequently gave an algorithm using a $\Sigma_2{}^P$ oracle [12].

Theorem 2 and its generalization give a technique for proving bounds on proper learning in the exact model. Proving upper bounds on the size of the appropriate certificates yields upper bounds on query complexity. Proving lower bounds on the size of appropriate certificates yields lower bounds on query complexity and hence also on time complexity. Moreover, unlike many computational hardness results in learning, computational hardness results achieved in this way do not rely on any unproven complexity theoretic or cryptographic hardness assumptions.

One of the most widely studied problems in computational learning theory has been the question of whether DNF formulas can be learned in polynomial time in common learning models. The following result on learning DNF formulas was proved using Theorem 2, by bounding the size of the relevant certificates.

Theorem 3 *There is a proper algorithm that learns DNF formulas in the exact model with query complexity bounded above by a polynomial $p(m,r,n)$, where m is the size of the smallest DNF representing the target function f, n is the number of variables on which f is defined, and r is the size of the smallest CNF representing f.*

The size of a DNF is the number of its terms; the size of a CNF is the number of its clauses. The above theorem does not imply polynomial-time learnability of arbitrary DNF formulas, since the running time of the algorithm depends not just on the size of the smallest DNF representing the target but also on the size of the smallest CNF.

Building on results of Alekhnovich et al., Feldman showed that if NP $\neq$ RP, DNF formulas cannot be properly learned in polynomial time in the PAC model augmented with membership queries. The same negative result then follows immediately for the exact model [1, 7]. Hellerstein and Raghavan used certificate size lower bounds and Theorem 1 to prove that DNF formulas cannot be learned by a proper exact algorithm with polynomial query complexity, if the algorithm is restricted to using DNF hypotheses that are only slightly larger than the target [9].

The main results of Hellerstein et al. apply to learning with membership and equivalence queries. Hellerstein et al. also considered the model of exact learning with membership queries alone and showed that in this model, a projection-closed Boolean function class is polynomial query learnable iff it has polynomial teaching dimension. Teaching dimension was previously defined by Goldman and Kearns. Hegedüs defined the extended teaching dimension and showed that all classes are polynomially query learnable with membership queries alone iff they have polynomial extended teaching dimension.

Balcázar et al. introduced the general dimension, which generalizes the combinatorial dimensions discussed above [5]. It can be used to characterize polynomial query learnability for a wide range of different queries. Balcan and Hanneke have investigated related combinatorial dimensions in the active learning setting [4].

Open Problems

It remains open whether DNF formulas can be learned in polynomial time in the exact model, using hypotheses that are not DNF formulas.

Feldman's results show the computational hardness of proper learning of DNF in the exact learning model based on complexity-theoretic assumptions. However, it is unclear whether query complexity is also a barrier to efficient learning of DNF formulas. It is still open whether the class of DNF formulas has

polynomial certificates; showing they do not have polynomial certificates would give a hardness result for proper learning of DNF based only on query complexity, with no complexity-theoretic assumptions (and without the hypothesis-size restrictions used by Hellerstein and Raghavan). DNF formulas do have certain sub-exponential certificates [11].

It is open whether decision trees have polynomial certificates.

Certificate techniques are used to prove lower bounds on learning when we restrict the type of hypotheses used by the learning algorithm. These types of results are called representation dependent, since they depend on the restriction of the representations used as hypotheses. Although there are some techniques for proving representation-independent hardness results, there is a need for more powerful techniques.

Cross-References

- ▸ Cryptographic Hardness of Learning
- ▸ Hardness of Proper Learning
- ▸ Learning DNF Formulas
- ▸ Learning with the Aid of an Oracle

Recommended Reading

1. Alekhnovich M, Braverman M, Feldman V, Klivans AR, Pitassi T (2004) Learnability and automatizability. In: Proceedings of the 45th annual IEEE symposium on foundations of computer science (FOCS '04), Rome. IEEE Computer Society, Washington, DC, pp 621–630
2. Angluin D (1987) Queries and concept learning. Mach Learn 2(4):319–342
3. Angluin D (2004) Queries revisited. Theor Comput Sci 313(2):175–194
4. Balcan N, Hanneke S (2012) Robust interactive learning. In: Proceedings of the twenty fifth annual conference on learning theory (COLT '12), Edinburgh, pp 20.1–20.34
5. Balcázar JL, Castro J, Guijarro D, Köbler J, Lindner W (2007) A general dimension for query learning. J Comput Syst Sci 73(6):924–940
6. Balcázar JL, Castro J, Guijarro D, Simon H-U (2002) The consistency dimension and distribution-dependent learning from queries. Theor Comput Sci 288(2):197–215
7. Feldman V (2006) Hardness of approximate two-level logic minimization and PAC learning with membership queries. In: Proceedings of the 38th annual ACM symposium on the theory of computing (STOC '06), Seattle. ACM, New York, pp 363–372
8. Hegedüs T (1995) Generalized teaching dimensions and the query complexity of learning. In: Proceedings of the 8th annual conference on computational learning theory (COLT '95), Santa Cruz, pp 108–117
9. Hellerstein L, Raghavan V (2005) Exact learning of DNF formulas using DNF hypotheses. J Comput Syst Sci 70(4):435–470
10. Hellerstein L, Pillaipakkamnatt K, Raghavan V, Wilkins D (1996) How many queries are needed to learn? J ACM 43(5):840–862
11. Hellerstein L, Kletenik D, Sellie L, Servedio R (2012) Tight bounds on proper equivalence query learning of DNF. In: Proceedings of the 25th annual conference on learning theory (COLT '12), Edinburgh, pp 31.1—31.18
12. Köbler J, Lindner W (2000) Oracles in Σ^{P}_{2} are sufficient for exact learning. Int J Found Comput Sci 11(4):615–632
13. Moshkov MY (1983) Conditional tests. Probl Kibern (in Russian) 40:131–170

Channel Assignment and Routing in Multi-Radio Wireless Mesh Networks

Mansoor Alicherry, Randeep Bhatia, and Li (Erran) Li
Bell Laboratories, Alcatel-Lucent, Murray Hill, NJ, USA

Keywords

Ad-hoc networks; Graph coloring

Years and Authors of Summarized Original Work

2005; Alicherry, Bhatia, Li

Problem Definition

One of the major problems facing wireless networks is the capacity reduction due to interference among multiple simultaneous transmissions. In wireless mesh networks providing mesh

routers with multiple-radios can greatly alleviate this problem. With multiple-radios, nodes can transmit and receive simultaneously or can transmit on multiple channels simultaneously. However, due to the limited number of channels available the interference cannot be completely eliminated and in addition careful channel assignment must be carried out to mitigate the effects of interference. Channel assignment and routing are inter-dependent. This is because channel assignments have an impact on link bandwidths and the extent to which link transmissions interfere. This clearly impacts the routing used to satisfy traffic demands. In the same way traffic routing determines the traffic flows for each link which certainly affects channel assignments. Channel assignments need to be done in a way such that the communication requirements for the links can be met. Thus, the problem of throughput maximization of wireless mesh networks must be solved through channel assignment, routing, and scheduling.

Formally, given a wireless mesh backbone network modeled as a graph (V, E): The node $t \in V$ represents the wired network. An edge $e = (u, v)$ exists in E iff u and v are within *communication range* R_T. The set $V_G \subseteq V$ represents the set of gateway nodes. The system has a total of K channels. Each node $u \in V$ has $I(u)$ network interface cards, and has an aggregated demand $l(u)$ from its associated users. For each edge e the set $I(e) \subset E$ denotes the set of edges that it interferes with. A pair of nodes that use the same channel and are within *interference range* $R_I x$ may interfere with each other's communication, even if they cannot directly communicate. Node pairs using different channels can transmit packets simultaneously without interference. The problem is to maximize λ where at least $\lambda l(u)$ amount of throughput can be routed from each node u to the Internet (represented by a node t). The $\lambda l(u)$ throughput for each node u is achieved by computing $g(1)$ a network flow that associates with each edge $e = (u, v)$ values $f(e(i)), 1 \leq i \leq K$ where $f(e(i))$ is the rate at which traffic is transmitted by node u for node v on channel i; (2) a feasible channel assignment $F(u)$ ($F(u)$ is an ordered set where the ith interface of u operates on the ith channel in $F(u)$) such that, whenever $f(e(i)) > 0, \ i \in F(u) \cap F(v)$; (3) a *feasible* schedule S that decides the set of edge channel pair (e, i) (edge e using channel, i.e., $f(e(i)) > 0$ scheduled at time slot τ, for $\tau = 1, 2, \ldots, T$ where T is the period of the schedule. A schedule is feasible if the edges of no two edge pairs $(e_1, i), (e_2, i)$ scheduled in the same time slot for a common channel i interfere with each other ($e_1 \notin I(e_2)$ and $e_2 \notin I(e_1)$). Thus, a feasible schedule is also referred to as an interference free edge schedule. An indicator variable $X_{e,i,\tau}, e \in E, i \in F(e), \tau \geq 1$ is used. It is assigned 1 if and only if link e is active in slot τ on channel i. Note that $1/T \sum_{1 \leq \tau \leq T} X_{e,i,\tau} c(e) = f(e(i))$. This is because communication at rate $c(e)$ happens in every slot that link e is active on channel i and since $f(e(i))$ is the average rate attained on link e for channel i. This implies $1/T \sum_{1 \leq \tau \leq T} X_{e,i,\tau} = \frac{f(e(i))}{c(e)}$.

Joint Routing, Channel Assignment, and Link Scheduling Algorithm

Even the interference free edge scheduling subproblem given the edge flows is NP-hard [5]. An approximation algorithm called RCL for the joint routing, channel assignment, and link scheduling problem has been developed. The algorithm performs the following steps in the given order:

1. **Solve LP:** First optimally solve a LP relaxation of the problem. This results in a flow on the flow graph along with a not necessarily feasible channel assignment for the node radios. Specifically, a node may be assigned more channels than the number of its radios. However, this channel assignment is "optimal" in terms of ensuring that the interference for each channel is minimum. This step also yields a lower bound on the λ value which is used in establishing the worst case performance guarantee of the overall algorithm.

2. **Channel Assignment:** This step presents a channel assignment algorithm which is used to adjust the flow on the flow graph (routing changes) to ensure a feasible channel assignment. This flow adjustment also strives to keep the increase in interference for each channel to a minimum.
3. **Interference Free Link Scheduling:** This step obtains an interference free link schedule for the edge flows corresponding to the flow on the flow graph.

Each of these steps is described in the following subsections.

A Linear Programming-Based Routing Algorithm

A linear program LP (1) to find a flow that maximizes λ is given below:

$$\max \lambda \tag{1}$$

Subject to

$$\lambda l(v) + \sum_{e=(u,v)\in E} \sum_{i=1}^{K} f(e(i)) = \sum_{e=(v,u)\in E} \sum_{i=1}^{K} f(e(i)) \,, \forall v \in V - V_G \tag{2}$$

$$f(e(i)) \leq c(e) \,, \quad \forall e \in E \tag{3}$$

$$\sum_{1\leq i\leq K} \left(\sum_{e=(u,v)\in E} \frac{f(e(i))}{c(e)} + \sum_{e=(v,u)\in E} \frac{f(e(i))}{c(e)} \right) \leq I(v) \,, v \in V \tag{4}$$

$$\frac{f(e(i))}{c(e)} + \sum_{e'\in I(e)} \frac{f(e'(i))}{c(e')} \leq c(q) \,, \quad \forall e \in E, \; 1 \leq i \leq K. \tag{5}$$

The first two constraints are *flow constraints*. The first one is the flow conservation constraint; the second one ensures no link capacity is violated. The third constraint is the *node radio constraints*. Recall that a IWMN node $v \in V$ has $I(v)$ radios and hence can be assigned at most $I(v)$ channels from $1 \leq i \leq K$. One way to model this constraint is to observe that due to interference constraints v can be involved in at most $I(v)$ simultaneous communications (with different one hop neighbors). In other words this constraint follows from $\sum_{1\leq i\leq K} \sum_{e=(u,v)\in E} X_{e,i,\tau} + \sum_{1\leq i\leq K} \sum_{e=(v,u)\in E} X_{e,i,\tau} \leq I(v)$. The fourth constraint is the *link congestion constraints* which are discussed in detail in section "Link Flow Scheduling". Note that all the constraints listed above are necessary conditions for any feasible solution. However, these constraints are not necessarily sufficient. Hence if a solution is found that satisfies these constraints it may not be a feasible solution. The approach is to start with a "good" but not necessarily feasible solution that satisfies all of these constraints and use it to construct a feasible solution without impacting the quality of the solution.

A solution to this LP can be viewed as a flow on a *flow graph* $H = (V, E^H)$ where $E^H = \{e(i) | \forall e \in E, 1 \leq i \leq K\}$. Although the optimal solution to this LP yields the best possible λ (say λ^*) from a practical point of view more improvements may be possible:

- The flow may have directed cycles. This may be the case since the LP does not try to minimize the amount of interference directly. By removing the flow on the directed cycle (equal amount off each edge) flow conservation is maintained and in addition since there are fewer transmissions the amount of interference is reduced.
- The flow may be using a long path when shorter paths are available. Note that longer paths imply more link transmissions. In this case it is often the case that by moving the flow to shorter paths, system interference may be reduced.

The above arguments suggest that it would be practical to find among all solutions that attain the

optimal λ value of λ^* the one for which the total value of the following quantity is minimized:

$$\sum_{1 \le i \le K} \sum_{e=(v,u) \in E} \frac{f(e(i))}{c(e)}.$$

The LP is then re-solved with this objective function and with λ fixed at λ^*.

Channel Assignment

The solution to the LP (1) is a set of flow values $f(e(i))$ for edge e and channel i that maximize the value λ. Let λ^* denote the optimal value of λ. The flow $f(e(i))$ implies a channel assignment where the two end nodes of edge e are both assigned channel i if and only if $f(e(i)) > 0$. Note that for the flow $f(e(i))$ the implied channel assignment may not be feasible (it may require more than $I(v)$ channels at node v). The channel assignment algorithm transforms the given flow to fix this infeasibility. Below only a sketch of the algorithm is given. More details can be found in [1].

First observe that in an idle scenario, where all nodes v have the same number of interfaces I (i.e., $I = I(v)$) and where the number of available channels K is also I, the channel assignment implied by the LP (1) is feasible. This is because even the trivial channel assignment where all nodes are assigned all the channels 1 to I is feasible. The main idea behind the algorithm is to first transform the LP (1) solution to a new flow in which every edge e has flow $f(e(i)) > 0$ only for the channels $1 \le i \le I$. The basic operation that the algorithm uses for this is to equally distribute, for every edge e, the flow $f(e(i))$, for $I < i \le K$ to the edges $e(j)$, for $1 \le i \le I$. This ensures that all $f(e(i)) = 0$, for $I < i \le K$ after the operation. This operation is called Phase I of the Algorithm. Note that the Phase I operation does not violate the flow conservation constraints or the node radio constraints (5) in the LP (1). It can be shown that in the resulting solution the flow $f(e(i))$ may exceed the capacity of edge e by at most a factor $\phi = K/I$. This is called the "inflation factor" of Phase I. Likewise in the new flow, the *link congestion constraints* (5) may also be violated for edge e and channel i by no more than the inflation factor φ. In other words in the resulting flow

$$\frac{f(e(i))}{c(e)} + \sum_{e' \in I(e)} \frac{f(e'(i))}{c(e')} \le \phi c(q)\,.$$

This implies that if the new flow is scaled by a fraction $1/\phi$ than it is feasible for the LP (1). Note that the implied channel assignment (assign channels 1 to I to every node) is also feasible. Thus, the above algorithm finds a feasible channel assignment with a λ value of at least λ^*/ϕ.

One shortcoming of the channel assignment algorithm (Phase I) described so far is that it only uses I of the K available channels. By using more channels the interference may be further reduced thus allowing for more flow to be pushed in the system. The channel assignment algorithm uses an additional heuristic for this improvement. This is called Phase II of the algorithm.

Now define an operation called "channel switch operation." Let A be a maximal connected component (the vertices in A are not connected to vertices outside A) in the graph formed by the edges e for a given channel i for which $f(e(i)) > 0$. The main observation to use is that for a given channel j, the operation of completely moving flow $f(e(i))$ to flow $f(e(j))$ for every edge e in A, does not impact the feasibility of the implied channel assignment. This is because there is no increase in the number of channels assigned per node after the flow transformation: the end nodes of edges e in A which were earlier assigned channel i are now assigned channel j instead. Thus, the transformation is equivalent to switching the channel assignment of nodes in A so that channel i is discarded and channel j is gained if not already assigned.

The Phase II heuristic attempts to re-transform the unscaled Phase I flows $f(e(i))$ so that there are multiple connected components in the graphs $G(e, i)$ formed by the edges e for each channel $1 \le i \le I$. This re-transformation is done so that the LP constraints are kept satisfied with an inflation factor of at most φ, as is the case for the unscaled flow after Phase I of the algorithm.

Next in Phase III of the algorithm the connected components within each graph $G(e,\ i)$ are grouped such that there are as close to K (but no more than) groups overall and such that the maximum interference within each group is minimized. Next the nodes within the lth group are assigned channel l, by using the channel switch operation to do the corresponding flow transformation. It can be shown that the channel assignment implied by the flow in Phase III is feasible. In addition the underlying flows $f(e(i))$ satisfy the LP (1) constraints with an inflation factor of at most $\phi = K/I$.

Next the algorithm scales the flow by the largest possible fraction (at least $1/\phi$) such that the resulting flow is a feasible solution to the LP (1) and also implies a feasible channel assignment solution to the channel assignment. Thus, the overall algorithm finds a feasible channel assignment (by not necessarily restricting to channels 1 to I only) with a λ value of at least λ^*/ϕ.

Link Flow Scheduling

The results in this section are obtained by extending those of [4] for the single channel case and for the Protocol Model of interference [2]. Recall that the time slotted schedule S is assumed to be periodic (with period T) where the indicator variable $X_{e,i,\tau}, e \in E, i \in F(e), \tau \geq 1$ is 1 if and only if link e is active in slot τ on channel i and i is a channel in common among the set of channels assigned to the end-nodes of edge e.

Directly applying the result (Claim 2) in [4] it follows that a necessary condition for interference free link scheduling is that for every $e \in E, i \in F(e), \tau \geq 1$: $X_{e,i,\tau} + \sum_{e' \in I(e)} X_{e',i,\tau} \leq c(q)$. Here $c(q)$ is a constant that only depends on the interference model. In the interference model this constant is a function of the fixed value q, the ratio of the interference range R_I to the transmission range R_T, and an intuition for its derivation for a particular value $q = 2$ is given below.

Lemma 1 $c(q) = 8$ *for* $q = 2$.

Proof Recall that an edge $e' \in I(e)$ if there exist two nodes $x, y \in V$ which are at most $2R_T$ apart and such that edge e is incident on node x and edge e' is incident on node y. Let $e = (u, v)$. Note that u and v are at most R_T apart. Consider the region C formed by the union of two circles C_u and C_v of radius $2R_T$ each, centered at node u and node v, respectively. Then $e' = (u', v') \in I(e)$ if an only if at least one of the two nodes u', v' is in C; Denote such a node by $C(e')$. Given two edges $e_1, e_2 \in I(e)$ that do not interfere with each other it must be the case that the nodes $C(e_1)$ and $C(e_2)$ are at least $2R_T$ apart. Thus, an upper bound on how many edges in $I(e)$ do not pair-wise interfere with each other can be obtained by computing how may nodes can be put in C that are pair-wise at least $2R_T$ apart. It can be shown [1] that this number is at most 8. Thus, in schedule S in a given slot only one of the two possibilities exist: either edge e is scheduled or an "independent" set of edges in $I(e)$ of size at most 8 is scheduled implying the claimed bound. □

A necessary condition: (*Link Congestion Constraint*) Recall that $\frac{1}{T}\sum_{1\leq\tau\leq T} X_{e,i,\tau} = \frac{f(e(i))}{c(e)}$. Thus: Any valid "interference free" edge flows must satisfy for every link e and every channel i the Link Congestion Constraint:

$$\frac{f(e(i))}{c(e)} + \sum_{e' \in I(e)} \frac{f(e'(i))}{c(e')} \leq c(q). \qquad (6)$$

A matching sufficient condition can also established [1].

A sufficient condition: (*Link Congestion Constraint*) If the edge flows satisfy for every link e and every channel i the following *Link Schedulability Constraint* than an interference free edge communication schedule can be found using an algorithm given in [1].

$$\frac{f(e(i))}{c(e)} + \sum_{e' \in I(e)} \frac{f(e'(i))}{c(e')} \leq 1. \qquad (7)$$

The above implies that if a flow $f(e(i))$ satisfies the *Link Congestion Constraint* then by scaling the flow by a fraction $1/c(q)$ it can be scheduled free of interference.

Key Results

Theorem *The RCL algorithm is a $Kc(q)/I$ approximation algorithm for the Joint Routing and Channel Assignment with Interference Free Edge Scheduling problem.*

Proof Note that the flow $f(e(i))$ returned by the channel assignment algorithm in Sect. "Channel Assignment" satisfies the *Link Congestion Constraint*. Thus, from the result of Sect. "Link Flow Scheduling" it follows that by scaling the flow by an additional factor of $1/c(q)$ the flow can be realized by an interference free link schedule. This implies a feasible solution to the joint routing, channel assignment and scheduling problem with a λ value of at least $\lambda^*/\phi c(q)$. Thus, the RCL algorithm is a $\phi c(q) = Kc(q)/I$ approximation algorithm. □

Applications

Infrastructure mesh networks are increasingly been deployed for commercial use and law enforcement. These deployment settings place stringent requirements on the performance of the underlying IWMNs. Bandwidth guarantee is one of the most important requirements of applications in these settings. For these IWMNs, topology change is infrequent and the variability of aggregate traffic demand from each mesh router (client traffic aggregation point) is small. These characteristics admit periodic optimization of the network which may be done by a system management software based on traffic demand estimation. This work can be directly applied to IWMNs. It can also be used as a benchmark to compare against heuristic algorithms in multi-hop wireless networks.

Open Problems

For future work, it will be interesting to investigate the problem when routing solutions can be enforced by changing link weights of a distributed routing protocol such as OSPF. Also, can the worst case bounds of the algorithm be improved (e.g., a constant factor independent of K and I)?

Cross-References

- Graph Coloring
- Stochastic Scheduling

Recommended Reading

1. Alicherry M, Bhatia R, Li LE (2005) Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks. In: Proceedings of the ACM MOBICOM, pp 58–72
2. Gupta P, Kumar PR (2000) The capacity of wireless networks. IEEE Trans Inf Theory IT-46(2):388–404
3. Jain K, Padhye J, Padmanabhan VN, Qiu L (2003) Impact of interference on multi-hop wireless network performance. In: Proceedings of the ACM MOBICOM, pp 66–80
4. Kumar VSA, Marathe MV, Parthasarathy S, Srinivasan A (2004) End-to-end packet-scheduling in wireless ad-hoc networks. In: Proceedings of the ACM-SIAM symposium on discrete algorithms, pp 1021–1030
5. Kumar VSA, Marathe MV, Parthasarathy S, Srinivasan A (2005) Algorithmic aspects of capacity in wireless networks. In: Proceedings of the ACM SIGMETRICS, pp 133–144
6. Kyasanur P, Vaidya N (2005) Capacity of multi-channel wireless networks: impact of number of channels and interfaces. In: Proceedings of the ACM MOBICOM, pp 43–57

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach

Martin D.F. Wong[1] and Honghua Hannah Yang[2]
[1]Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA
[2]Strategic CAD Laboratories, Intel Corporation, Hillsboro, OR, USA

Keywords

Hypergraph partitioning; Netlist partitioning

Years and Authors of Summarized Original Work

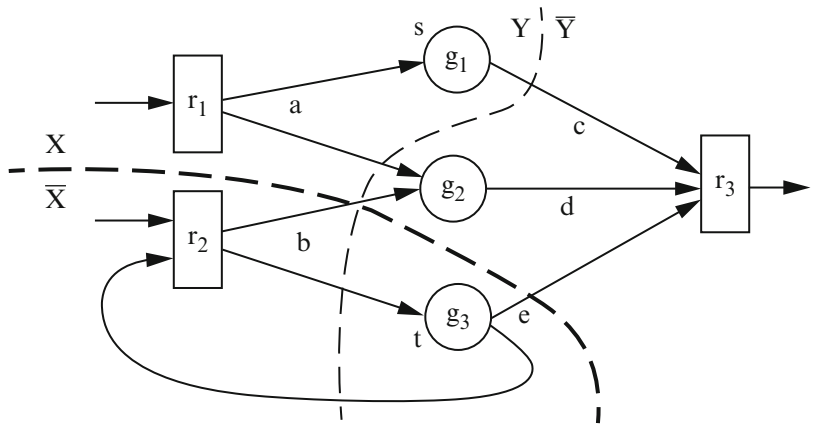
1994; Yang, Wong

Problem Definition

Circuit partitioning is a fundamental problem in many areas of VLSI layout and design. *Min-cut balanced bipartition* is the problem of partitioning a circuit into two disjoint components with equal weights such that the number of nets connecting the two components is minimized. The min-cut balanced bipartition problem was shown to be NP-complete [5]. The problem has been solved by heuristic algorithms, e.g., Kernighan and Lin type (K&L) iterative improvement methods [4, 11], simulated annealing approaches [10], and analytical methods for the ratio-cut objective [2, 7, 13, 15]. Although it is a natural method for finding a min-cut, the network max-flow min-cut technique [6, 8] has been overlooked as a viable approach for circuit partitioning. In [16], a method was proposed for exactly modeling a circuit netlist (or, equivalently, a hypergraph) by a flow network, and an algorithm for balanced bipartition based on repeated applications of the max-flow min-cut technique was proposed as well. Our algorithm has the same asymptotic time complexity as one max-flow computation.

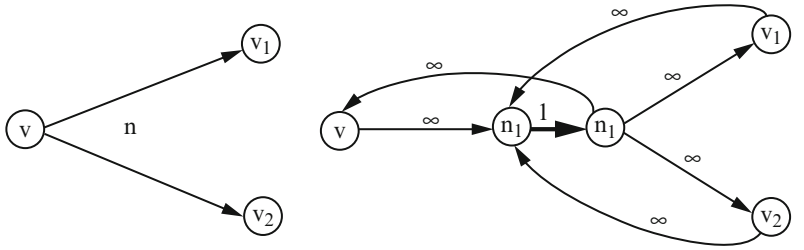
A *circuit netlist* is defined as a digraph $N = (V, E)$, where V is a set of nodes representing logic gates and registers and E is a set of edges representing wires between gates and registers. Each node $v \in V$ has a weight $w(v) \in R^+$. The total weight of a subset $U \subseteq V$ is denoted by $w(U) = \Sigma_{v \in U} w(v)$. $W = w(V)$ denotes the total weight of the circuit. A *net* $n = (v; v_1, \ldots, v_l)$ is a set of outgoing edges from node v in N. Given two nodes s and t in N, an $s - t$ *cut* (or *cut* for short) $\left(X, \bar{X}\right)$ of N is a bipartition of the nodes in V such that $s \in X$ and $t \in \bar{X}$. The *net-cut* $net\left(X, \bar{X}\right)$ of the cut is the set of nets in N that are incident to nodes in both X and $\bar{X}$. A cut $\left(X, \bar{X}\right)$ is a *min-net-cut* if $\left|net\left(X, \bar{X}\right)\right|$ is minimum among all $s - t$ cuts of N. In Fig. 1, net $a = [r_1; g_1, g_2)$, net cuts $net\left(X, \bar{X}\right) = \{b, e\}$ and $net\left(Y, \bar{Y}\right) = \{c, a, b, e\}$, and $\left(X, \bar{X}\right)$ is a min-net-cut.

Formally, given an aspect ratio r and a deviation factor ε, *min-cut r-balanced bipartition* is the problem of finding a bipartition $\left(X, \bar{X}\right)$ of the netlist N such that (1) $(1 - \epsilon)rW \leq W(X) \leq (1 + \epsilon)rW$ and (2) the size of the cut $net\left(X, \bar{X}\right)$ is minimum among all bipartitions satisfying (1). When $r = 1/2$, this becomes a min-cut balanced-bipartition problem.

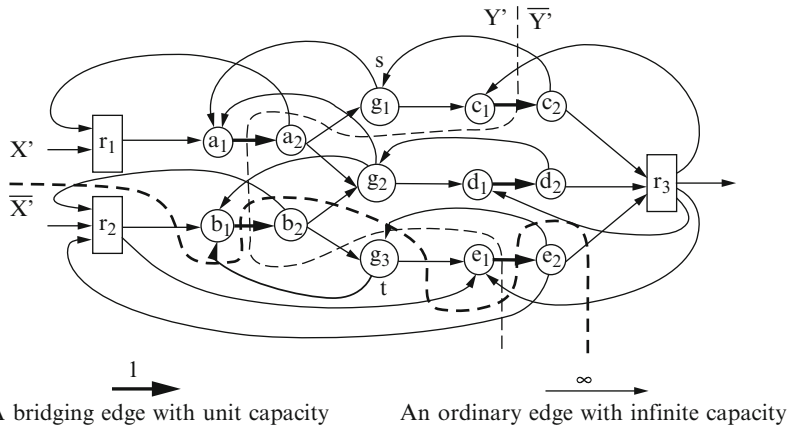


Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Fig. 1 A circuit netlist with two net-cuts

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Fig. 2 Modeling a net in N in the flow network N'



Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Fig. 3 The flow network for Fig. 1



Key Results

Optimal-Network-Flow-Based Min-Net-Cut Bipartition

The problem of finding a min-net-cut in $N = (V, E)$ is reduced to the problem of finding a cut of minimum capacity. Then the latter problem is solved using the max-flow min-cut technique. A flow network $N' = (V', E')$ is constructed from $N = (V, E)$ as follows (see Figs. 2 and 3):

1. V' contains all nodes in V.
2. For each net $n = (v; v_1, \ldots, v_l)$ in N, add two nodes n_1 and n_2 in V' and a *bridging edge* $bridge(n) = (n_1, n_2)$ in E'.
3. For each node $u \in \{v, v_1, \ldots, v_l\}$ incident on net n, add two edges (u, n_1) and (n_2, u) in E'.
4. Let s be the source of N' and t the sink of N'.
5. Assign unit capacity to all bridging edges and infinite capacity to all other edges in E'.
6. For a node $v \in V'$ corresponding to a node in V, $w(v)$ is the weight of v in N. For a node $u \in V'$ split from a net, $w(u) = 0$.

Note that all nodes incident on net n are connected to n_1 and are connected from n_2 in N'. Hence, the flow network construction is symmetric with respect to all nodes incident on a net. This construction also works when the netlist is represented as a hypergraph.

It is clear that N' is a strongly connected digraph. This property is the key to reducing the bidirectional min-net-cut problem to a minimum-capacity cut problem that counts the capacity of the forward edges only.

Theorem 1 *N has a cut of net-cut size at most C if and only if N' has a cut of capacity at most C.*

Corollary 1 *Let $(X', \bar{X}')$ be a cut of minimum capacity C in N'. Let $N_{cut} = \{n \mid bridge(n) \in (X', \bar{X}')\}$. Then $N_{cut} = (X, \bar{X})$ is a min-net-cut in N and $|N_{cut}| = C$.*

Corollary 2 *A min-net-cut in a circuit $N = (V, E)$ can be found in $O(|V||E|)$ time.*

Min-Cut Balanced-Bipartition Heuristic

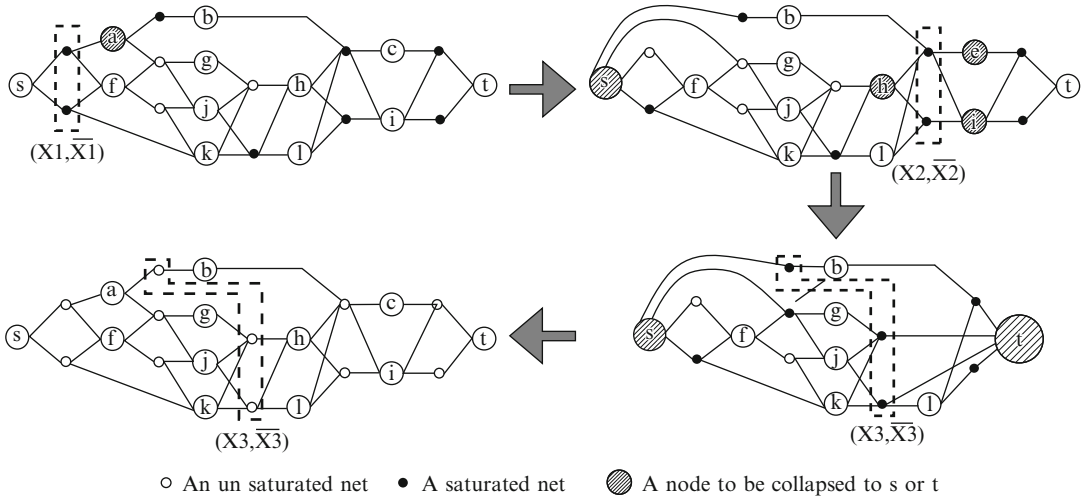
First, a repeated max-flow min-cut heuristic algorithm, flow-balanced bipartition (FBB), is developed for finding an r-balanced bipartition that minimizes the number of crossing nets. Then, an efficient implementation of FBB is developed that has the same asymptotic time complexity as one max-flow computation. For ease of presentation, the FBB algorithm is described on the original circuit rather than the flow network constructed from the circuit. The heuristic algorithm is described in Fig. 4. Figure 5 shows an example.

Table 1 compares the best bipartition net-cut sizes of FBB with those produced by the analytical-method-based partitioners EIG1 [7] and PARABOLI (PB) [13]. The results produced by PARABOLI were the best previously known results reported on the benchmark circuits. The results for FBB were the best of ten runs. On average, FBB outperformed EIG1 and PARABOLI by 58.1 and 11.3 %, respectively. For circuit S38417, the suboptimal result from FBB can be improved by (1) running more times and (2) applying clustering techniques to the circuit based on connectivity before partitioning.

In the FBB algorithm, the node-collapsing method is chosen instead of a more gradual method (e.g., [9]) to ensure that the capacity of a cut always reflects the real net-cut size. To pick a node at steps 4.2 and 5.2, a threshold R is given for the number of nodes in the uncollapsed subcircuit. A node is randomly picked if the number of nodes is larger than R. Otherwise, all nodes adjacent to C are tried and the one whose collapse induces a min-net-cut with the smallest size is picked. A naive implementation of step 2 by computing the max-flow from the zero flow would incur a high time complexity. Instead, the flow value in the flow network is retained, and additional flow is explored to saturate the bridging edges of the min-net-cut from one iteration to the next. The procedure is shown in Fig. 6. Initially, the flow network retains the flow function computed in the previous iteration. Since the max-flow computation using the augmenting-path method is insensitive to the initial flow values in the flow network and the order in which the augmenting paths are found, the above procedure correctly finds a max-flow with the same flow value as a max-flow computed in the collapsed flow network from the zero flow.

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Fig. 4 FBB algorithm

Algorithm: Flow-Balanced-Bipartition (FBB)
1. Pick a pair of nodes s and t in N;
2. Find a min-net-cut C in N;
   Let X be the subcircuit reachable from s through augmenting paths in the flow network, and $\overline{X}$ the rest;
3. **if** $(1-\epsilon)rW \le w(X) \le (1+\epsilon)rW$
   return C as the answer;
4. **if** $w(X) < (1-\epsilon)rW$
4.1. Collapse all nodes in X to s;
4.2. Pick a node $v \in \overline{X}$ adjacent to C and collapse it to s;
4.3. Goto 1;
5. **if** $w(X) > (1+\epsilon)rW$
5.1. Collapse all nodes in $\overline{X}$ to t;
5.2. Pick a node $v \in X$ adjacent to C and collapse it to t;
5.3. Goto 1;



Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Fig. 5 FBB on the example in Fig. 3 for $r = 1/2, \epsilon = 0.15$ and unit weight for each node. The algorithm terminates after finding cut $(X_2, \bar{X}_2)$. A small solid node indicates that the bridging edge corresponding to the net is saturated with flow

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Table 1 Comparison of EIG1, PB, and FBB $(r = 1/2, \epsilon = 0.1)$. All allow ≤ 10 % deviation

Circuit				Best net-cut size			Improve. % over		
Name	Gates and latches	Nets	Avg. deg	EIG1	PB	FBB	EIG1	PB	FBB elaps. sec.
S1423	731	743	2.7	23	16	13	43.5	18.8	1.7
S9234	5,808	5,805	2.4	227	74	70	69.2	5.4	55.7
S13207	8,696	8,606	2.4	241	91	74	69.3	18.9	100.0
S15850	10,310	10,310	2.4	215	91	67	68.8	26.4	96.5
S35932	18,081	17,796	2.7	105	62	49	53.3	21.0	2,808
S38584	20,859	20,593	2.7	76	55	47	38.2	14.5	1,130
S38417	24,033	23,955	2.4	121	49	58	52.1	−18.4	2,736
Average							58.5	11.3	

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Fig. 6 Incremental max-flow computation

Procedure: Incremental Flow Computation

1. **while** ∃ an additional augmenting path from s to t increase flow value along the augmenting path;
2. Mark all nodes u s.t.∃ an augmenting path from s to u;
3. Let C' be the set of bridging edges whose starting nodes are marked and ending nodes are not marked;
4. Return the nets corresponding to the bridging edges in C' as the min-net-cut C, and the marked nodes as X.

Theorem 2 *FBB has time complexity* $O(|V||E|)$ *for a connected circuit* $N = (V, E)$.

Theorem 3 *The number of iterations and the final net-cut size are nonincreasing functions of* $\in$.

In practice, FBB terminates much faster than this worst-case time complexity as shown in the section "Experimental Results." Theorem 3 allows us to improve the efficiency of FBB and the partition quality for a larger $\in$. This is not true for other partitioning approaches such as the K&L heuristics.

Applications

Circuit partitioning is a fundamental problem in many areas of VLSI layout and design automation. The FBB algorithm provides the first efficient predictable solution to the min-cut balanced-circuit-partitioning problem. It directly relates the efficiency and the quality of the solution produced by the algorithm to the deviation factor $\in$. The algorithm can be easily extended to handle nets with different weights by simply assigning the weight of a net to its bridging edge in the flow network. K-way min-cut partitioning for $K > 2$ can be accomplished by recursively applying FBB or by setting $r = 1/K$ and then using FBB to find one partition at a time. A flow-based method for directly solving the problem can be found in [12]. Prepartitioning circuit clustering according to the connectivity or the timing information of the circuit can be easily incorporated into FBB by treating a cluster as a node. Heuristic solutions based on K&L heuristics or simulated annealing with low temperature can be used to further fine-tune the solution.

Experimental Results

The FBB algorithm was implemented in SIS/MISII [1] and tested on a set of large ISCAS and MCNC benchmark circuits on a SPARC 10 workstation with 36-MHz CPU and 32 MB memory.

Table 2 compares the average bipartition results of FBB with those reported by Dasdan and Aykanat in [3]. SN is based on the K&L heuristic algorithm in Sanchis [14]. PFM3 is based on the K&L heuristic with free moves as described in [3]. For each circuit, SN was run 20 times and PFM3 10 times from different randomly generated initial partitions. FBB was run 10 times from different randomly selected s and t. With only one exception, FBB outperformed both SN and PFM3 on the five circuits. On average, FBB found a bipartition with 24.5 and 19.0 % fewer crossing nets than SN and PFM3, respectively. The runtimes of SN, PFM3, and FBB were not compared since they were run on different workstations.

Cross-References

▸ Circuit Placement
▸ Circuit Retiming

Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach, Table 2 Comparison of SN. PFM3. and FBB ($r = 1/2, \epsilon = 0.1$)

Circuit				Avg. net-cut size			FBB bipart. ratio	Improve. %	
Name	Gates and latches	Nets	Avg. deg	SN	PFM3	FBB		Over SN	Over PFM3
C1355	514	523	3.0	38.9	29.1	26.0	1:1.08	33.2	10.7
C2670	1,161	1,254	2.6	51.9	46.0	37.1	1:1.15	28.5	19.3
C3540	1,667	1,695	2.7	90.3	71.0	79.8	1:1.11	11.6	–12.4
C7552	3,466	3,565	2.7	44.3	81.8	42.9	1:1.08	3.2	47.6
S838	478	511	2.6	27.1	21.0	14.7	1:1.04	45.8	30.0
Ave							1:1.10	24.5	19.0

Recommended Reading

1. Brayton RK, Rudell R, Sangiovanni-Vincentelli AL (1987) MIS: a multiple-level logic optimization. IEEE Trans CAD 6(6):1061–1081
2. Cong J, Hagen L, Kahng A (1992) Net partitions yield better module partitions. In: Proceedings of the 29th ACM/IEEE design automation conference, Anaheim, pp 47–52
3. Dasdan A, Aykanat C (1994) Improved multiple-way circuit partitioning algorithms. In: International ACM/SIGDA workshop on field programmable gate arrays, Berkeley
4. Fiduccia CM, Mattheyses RM (1982) A linear time heuristic for improving network partitions. In: Proceedings of the ACM/IEEE design automation conference, Las Vegas, pp 175–181
5. Garey M, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, Gordonsville
6. Goldberg AW, Tarjan RE (1988) A new approach to the maximum flow problem. J SIAM 35: 921–940
7. Hagen L, Kahng AB (1991) Fast spectral methods for ratio cut partitioning and clustering. In: Proceedings of the IEEE international conference on computer-aided design, Santa Clara, pp 10–13
8. Hu TC, Moerder K (1985) Multiterminal flows in a hypergraph. In: Hu TC, Kuh ES (eds) VLSI circuit layout: theory and design. IEEE, New York, pp 87–93
9. Iman S, Pedram M, Fabian C, Cong J (1993) Finding uni-directional cuts based on physical partitioning and logic restructuring. In: 4th ACM/SIGDA physical design workshop, Lake Arrowhead
10. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 4598:671–680
11. Kernighan B, Lin S (1970) An efficient heuristic procedure for partitioning of electrical circuits. Bell Syst Tech J 49:291–307
12. Liu H, Wong DF (1998) Network-flow-based multiway partitioning with area and pin constraints. IEEE Trans CAD Integr Circuits Syst 17(1): 50–59
13. Riess BM, Doll K, Frank MJ (1994) Partitioning very large circuits using analytical placement techniques. In: Proceedings of the 31th ACM/IEEE design automation conference, San Diego, pp 646–651
14. Sanchis LA (1989) Multiway network partitioning. IEEE Trans Comput 38(1):62–81
15. Wei YC, Cheng CK (1989) Towards efficient hierarchical designs by ratio cut partitioning. In: Proceedings of the IEEE international conference on computer-aided design, Santa Clara, pp 298–301
16. Yang H, Wong DF (1994) Efficient network flow based min-cut balanced partitioning. In: Proceedings of the IEEE international conference on computer-aided design, San Jose, pp 50–55

Circuit Placement

Andrew A. Kennings[1] and Igor L. Markov[2]
[1]Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada
[2]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

Keywords

Algorithm; Circuit; Combinatorial optimization; Hypergraph; Large-scale optimization; Linear programming; Network flow; Nonlinear optimization; Partitioning; Physical design; Placement; VLSI CAD

Synonyms

Analytical placement; EDA; Layout; Mathematical programming; Min-cost max-flow; Min-cut placement; Netlist

Years and Authors of Summarized Original Work

2000; Caldwell, Kahng, Markov
2006; Kennings, Vorwerk
2012; Kim, Lee, Markov

Problem Definition

This problem is concerned with determining constrained positions of objects while minimizing a measure of interconnect between the objects, as in physical layout of integrated circuits, commonly done in 2 dimensions. While most formulations are NP-hard, modern circuits are so large that practical placement algorithms must have near-linear run time and memory requirements, but not necessarily produce optimal solutions. Research in placement algorithms has identified scalable techniques which are now being adopted in the electronic design automation industry.

One models a circuit by a hypergraph $G_h(V_h, E_h)$ with (i) vertices $V_h = \{v_1, \ldots, v_n\}$ representing logic gates, standard cells, larger modules, or fixed I/O pads and (ii) hyperedges $E_h = \{e_1, \ldots, e_m\}$ representing connections between modules. Vertices and hyperedges connect through *pins* for a total of P pins in the hypergraph. Each vertex $v_i \in V_h$ has width w_i, height h_i, and area A_i. Hyperedges may also be weighted. Circuit placement seeks center positions (x_i, y_i) for vertices that optimize a hypergraph-based objective subject to constraints (see below). A placement is captured by $\mathbf{x} = (x_1, \cdots, x_n)$ and $\mathbf{y} = (y_1, \cdots, y_n)$.

Objective: Let C_k be the index set of the hypergraph vertices incident to hyperedge e_k. The total half-perimeter wire length (HPWL) of the circuit hypergraph is given by $\text{HPWL}(G_h) = \sum_{e_k \in E_h} \text{HPWL}(e_k) = \sum_{e_k \in E_h} \left[\max_{i,j \in C_k} |x_i - x_j| + \max_{i,j \in C_k} |y_i - y_j|\right]$. HPWL is piecewise linear, separable in the x and y directions, convex, but not strictly convex. Among many objectives for circuit placement, it is the simplest and most common.

Constraints:

1. **No overlap**. The area occupied by any two vertices cannot overlap; i.e., either $|x_i - x_j| \geq \frac{1}{2}(w_i + w_j)$ or $|y_i - y_j| \geq \frac{1}{2}(h_i + h_j)$, $\forall v_i, v_j \in V_h$.
2. **Fixed outline**. Each vertex $v_i \in V_h$ must be placed entirely within a specified rectangular region bounded by $x_{\min}$ ($y_{\min}$) and $x_{\max}$ ($y_{\max}$) which denote the left (bottom) and right (top) boundaries of the specified region.
3. **Discrete slots**. There is only a finite number of discrete positions, typically on a grid. However, in large-scale circuit layout, slot constraints are often ignored during *global placement* and enforced only during *legalization* and *detail placement*.

Other constraints may include alignment, minimum and maximum spacing, etc. Many placement techniques temporarily relax overlap constraints into density constraints to avoid vertices clustered in small regions. A $m \times n$ regular bin structure B is superimposed over the fixed outline and vertex area is assigned to bins based on the positions of vertices. Let D_{ij} denote the density of bin $B_{ij} \in B$, defined as the total cell area assigned to bin B_{ij} divided by its capacity. Vertex overlap is limited implicitly by satisfying $D_{ij} \leq K$, $\forall B_{ij} \in B$, for some $K \leq 1$ (density target).

Problem 1 (Circuit Placement) INPUT: Circuit hypergraph $G_h(V_h, E_h)$ and a fixed outline for the placement area.
OUTPUT: Positions for each vertex $v_i \in V_h$ such that (1) wire length is minimized and (2) the area-density constraints $D_{ij} \leq K$ are satisfied for all $B_{ij} \in B$.

Key Results

An unconstrained optimal position of a single placeable vertex connected to fixed vertices can be found in linear time as the median of adjacent positions [7]. Unconstrained HPWL minimization for multiple placeable vertices can be formulated as a linear program [6, 11]. For each $e_k \in E_h$, upper and lower bound variables U_k and L_k are added. The cost of e_k (x-direction only) is the difference between U_k and L_k. Each $U_k(L_k)$ comes with p_k inequality constraints that restrict its value to be larger (smaller) than the position of every vertex $i \in C_k$.

Linear programming has poor scalability and integrating constraint-tracking into optimization is difficult. Other approaches include nonlinear optimization and partitioning-based methods.

Combinatorial Techniques for Wire Length Minimization

The no-overlap constraints are not convex and cannot be directly added to the linear program for HPWL minimization. Vertices often cluster in small regions of high density. One can lower bound the distance between closely placed vertices with a single linear constraint that depends on the relative placement of these vertices [11]. The resulting optimization problem is incrementally resolved, and the process repeats until the desired density is achieved.

The *min-cut placement* technique is based on balanced min-cut partitioning of hypergraphs and is more focused on density constraints [12]. Vertices of the initial hypergraph are first partitioned in two similar-sized groups. One of them is assigned to the left half of the placement region, and the other one to the right half. Partitioning is performed by the Multilevel Fiduccia-Mattheyses (MLFM) heuristic [10] to minimize connections between the two groups of vertices (the net-cut objective). Each half is partitioned again but takes into account the connections to the other half [12]. At the large scale, ensuring the similar sizes of bipartitions corresponds to density constraints, and cut minimization corresponds to HPWL minimization. When regions become small and contain <10 vertices, optimal positions can be found with respect to discrete slot constraints by branch-and-bound [2]. Balanced hypergraph partitioning is NP-hard [4], but the MLFM heuristic takes $O((V + E)\log V)$ time. The entire min-cut placement procedure takes $O((V + E)(\log V)^2)$ time and can process hypergraphs with millions of vertices in several hours.

A special case of interest is that of one-dimensional placement. When all vertices have identical width and none of them are fixed, one obtains the NP-hard MINIMUM LINEAR ARRANGEMENT problem [4] which can be approximated in polynomial time within $O(\log V)$ and solved exactly for trees in $O(V^3)$ time as shown by Yannakakis. The min-cut technique described above also works well for the related NP-hard MINIMUM-CUT LINEAR ARRANGEMENT problem [4].

Quadratic and Nonlinear Wire Length Approximations

Quadratic and generic nonlinear optimization may be faster than linear programming while reasonably approximating the original formulation.

Quadratic, Linearized Quadratic, and Bound-to-Bound Placement

The hypergraph is represented by a weighted graph where w_{ij} represents the weight on the 2-pin edge connecting vertices v_i and v_j in the weighted graph. When an edge is absent, $w_{ij} = 0$, and in general $w_{ii} = -\Sigma_{i \neq j} w_{ij}$. A quadratic placement (x-direction only) is given by

$$\Phi(x) = \sum_{i,j} w_{ij} \left[(x_i - x_j)^2\right] = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x} + \text{const.} \quad (1)$$

The global minimum of $\Phi(x)$ is found by solving $\mathbf{Q}\mathbf{x} + \mathbf{c} = \mathbf{0}$ which is a sparse, symmetric positive definite system of linear equations (assuming $\geq$1 fixed vertex), efficiently solved using any number of iterative solvers. Quadratic placement may have different optima depending on the model (clique or star) used to represent hyperedges. However, for a k-pin hyperedge, if $w_{ij} = W_c$ in a clique model and $w_{ij} = kW_c$ is a star model, then the models are equivalent in quadratic placement [6].

Quadratic placement can produce lower quality placements. To approximate a linear objective, one can iteratively solve Eq. 1 with $w_{ij} = 1/|x_i - x_j|$ computed at every iteration. Alternatively, one can solve a single β-regularized optimization problem given by $\Phi^\beta(\mathbf{x}) = \min_x \sum_{i,j} w_{ij}\sqrt{(x_i - x_j)^2 + \beta}$, $\beta > 0$, e.g., using a Primal-Dual Newton method [1].

In bound-to-bound placement, instead of a clique or star model, hyperedges are decomposed based on the relative placement of vertices. For a k-pin hyperedge, the extreme vertices (min and max) are connected to each other and to each internal vertex with weights $w_{ij} = 1/(k-1)|x_i - x_j|$. With these weights, the quadratic objective captures HPWL exactly, but only for the given placement. As placement changes, updates to the quadratic placement objective are required to reduce discrepancies [8].

Half-perimeter Wire Length Placement:

HPWL can be provably approximated by strictly convex and differentiable functions. For 2-pin hyperedges, β-regularization can be used [1]. For an k-pin hyperedge ($k \geq 3$), one can rewrite HPWL as the maximum (l_∞-norm) of

all $k(k-1)/2$ pairwise distances $|x_i - x_j|$ and approximate the l_∞-norm by the l_p-norm. This removes all non-differentiabilities except at $\mathbf{0}$ which is then removed with β-regularization. The resulting HPWL approximation is given by

$$\mathrm{HPWL}_{\mathrm{REG}}(G_h) = \sum_{e_k \in E_h} \left(\sum_{i,j \in C_k} |x_i - x_j|^p + \beta \right)^{1/p} \tag{2}$$

which overestimates HPWL with arbitrarily small relative error as $p \to \infty$ and $\beta \to 0$ [6]. Alternatively, HPWL can be approximated via the log-sum-exp (LSE) formula given by

$$\mathrm{HPWL}_{\mathrm{LSE}}(G_h) = \alpha \sum_{e_k \in E_h} \left[\ln \left(\sum_{i \in C_k} \exp\left(\frac{x_i}{\alpha}\right) \right) + \ln \left(\sum_{v_i \in C_k} \exp\left(\frac{-x_i}{\alpha}\right) \right) \right] \tag{3}$$

where $\alpha > 0$ is a smoothing parameter [5]. Both approximations can be optimized using conjugate gradient methods. Other convex and differentiable HPWL approximations exist.

Analytic Techniques for Target Density Constraints

The target density constraints are non-differentiable and are typically handled by approximation.

Force-Based Spreading

The key idea is to add constant forces $\mathbf{f}$ that pull vertices always from overlaps, and recompute the forces over multiple iterations to reflect changes in vertex distribution. For quadratic placement, the new optimality conditions are $\mathbf{Qx} + \mathbf{c} + \mathbf{f} = \mathbf{0}$ [7]. The constant force can perturb a placement in any number of ways to satisfy the target density constraints. The force $\mathbf{f}$ is computed using a discrete version of Poisson's equation.

Fixed-Point Spreading

A fixed point f is a pseudo-vertex with zero area, fixed at (x_f, y_f), and connected to one vertex $H(f)$ in the hypergraph through the use of a pseudo-edge with weight $w_{f,H(f)}$. Each fixed point introduces a single quadratic term into the objective function; quadratic placement with fixed points is given by $\Phi(x) = \sum_{i,j} w_{i,j}(x_i - x_j)^2 + \sum_f w_{f,H(f)}(x_{H(f)} - x_f)^2$. By manipulating the positions of fixed points, one can perturb a placement to satisfy the target density constraints. Fixed points improve the controllability and stability of placement iterations, in particular by improving the conditioning number of resulting numerical problem instances. A particularly effective approach to find fixed points is through the use of fast *LookAhead Legalization* (LAL) [8,9]. Given locations found by quadratic placement, LAL gradually modifies them into a relatively overlap-free placement that satisfies density constraints and seeks to preserve the ordering of x and y positions, while avoiding unnecessary movement. The resulting locations are used as fixed target points. LAL can be performed by top-down geometric partitioning with nonlinear scaling between partitions. As described in [8, 9], this approach is particularly effective at handling rectilinear obstacles. Subsequent work developed extensions to account for routing congestion and other considerations arising in global placement. At the most recent (ISPD 2014) placement contest, the contestants ranked in top three used the framework outlined in [8].

Generalized Force-Directed Spreading

The Helmholtz equation models a diffusion process and makes it ideal for spreading vertices [3]. The Helmholtz equation is given by

$$\begin{aligned} \frac{\partial^2 \phi(x,y)}{\partial x^2} + \frac{\partial^2 \phi(x,y)}{\partial y^2} - \epsilon \phi(x,y) &= D(x,y),\ (x,y) \in \mathrm{R} \\ \frac{\partial \phi}{\partial v} &= 0, \qquad (x,y) \text{ on the boundary of } R \end{aligned} \tag{4}$$

where $\epsilon > 0$, v is an outer unit normal, R represents the fixed outline, and $D(x, y)$ represents the continuous density function. The boundary conditions, $\frac{\partial \phi}{\partial v} = 0$, specify that forces pointing outside of the fixed outline be set to zero – this is a key difference with the Poisson method which assumes that forces become zero at infinity. The value ϕ_{ij} at the center of each bin B_{ij} is found by discretization of Eq. 4 using finite differences. The density constraints are replaced by $\phi_{ij} = \hat{K}, \forall B_{ij} \in B$ where $\hat{K}$ is a scaled representative of the density target K. Wire length minimization subject to the smoothed density constraints can be solved via Uzawa's algorithm. For quadratic wire length, this algorithm is a generalization of force-based spreading.

Potential Function Spreading

Target density constraints can also be satisfied via a penalty function. The area assigned to bin B_{ij} by vertex v_i is represented by Potential(v_i, B_{ij}) which is a bell-shaped function. The use of piecewise quadratic functions makes the potential function non-convex but smooth and differentiable [5]. The wire length approximation can be combined together with a penalty term given by Penalty $= \sum_{B_{ij} \in B} \left(\sum_{v_i \in V_h} \text{Potential}(v_i, B_{ij}) - K \right)^2$ to arrive at an unconstrained optimization problem which is solved using a conjugate gradient method [5].

Applications

Practical applications involve more sophisticated interconnect objectives, such as circuit delay, routing congestion, power dissipation, power density, and maximum thermal gradient. The above techniques are adapted to handle multiobjective optimization. Many such extensions are based on heuristic assignment of net weights that encourage the shortening of some (e.g., timing critical and frequently switching) connections at the expense of other connections. To moderate routing congestion, predictive congestion maps are used to decrease the maximal density constraint for placement in congested regions. Another application is in physical synthesis, where incremental placement is used to evaluate changes in circuit topology.

Experimental Results and Data Sets

Circuit placement has been actively studied for the past 30 years, and a wealth of experimental results have been reported. A 2003 result showed that placement tools could produce results as much as 1.41× to 2.09× known optimal wire lengths on average. In a 2006 placement contest, academic software for placement produced results that differed by as much as 1.39× on average when the objective was the simultaneous minimization of wire length, routability, and run time. Placement run times for instances with 2M movable objects ranged into hours. More recently, the gap in wire length between different tools has decreased, and run times have improved, in part due to the use of multicore CPUs and vectorized arithmetics. Over the last 10 years, wire length has improved by 20–25 % and run time by 15–20 times [8, 9]. More recent work in circuit placement has focused on other objectives such as routability in addition to wire length minimization.

Modern benchmark suites include the ISPD05, ISPD06, ISPD11, and ISPD14 suites (http://www.ispd.cc). Additional benchmark suites include the ICCAD12 (http://cad_contest.cs.nctu.edu.tw/CAD-contest-at-ICCAD2012), ICCAD13 (http://cad_contest.cs.nctu.edu.tw/CAD-contest-at-ICCAD2013), and ICCAD14 (http://cad_contest.ee.ncu.edu.tw/CAD-contest-at-ICCAD2014) suites. Instances in these benchmark suites contain between several hundred thousand to several million placeable objects. Additional benchmark suites also exist.

Cross-References

- Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach
- Floorplan and Placement
- Performance-Driven Clustering

Recommended Reading

1. Alpert CJ, Chan T, Kahng AB, Markov IL, Mulet P (1998) Faster minimization of linear wirelength for global placement. IEEE Trans CAD 17(1):3–13
2. Caldwell AE, Kahng AB, Markov IL (2000) Optimal partitioners and end-case placers for standard-cell layout IEEE Trans CAD 19(11):1304–1314
3. Chan T, Cong J, Sze K (2005) Multilevel generalized force-directed method for circuit placement. In: Proceedings of international symposium on physical design, San Francisco, pp 185–192
4. Crescenzi P, Kann V (1998) A compendium of NP optimization problems. Springer, Berlin/ Heidelberg
5. Kahng AB, Wang Q (2005) Implementation and extensibility of an analytic placer. IEEE Trans CAD 24(5):734–747
6. Kennings A, Markov IL (2002) Smoothing max-terms and analytical minimization of half-perimeter wirelength. VLSI Design 14(3):229–237
7. Kennings A, Vorwerk K (2006) Force-directed methods for generic placement. IEEE Trans CAD 25(10):2076–2087
8. Kim M-C, Lee D, Markov IL (2012) SimPL: an effective placement algorithm. IEEE Trans CAD 31(1):50–60
9. Lin T, Chu C, Shinnerl JR, Bustany I, Nedelchev I (2013) POLAR: placement based on novel rough legalization and refinement. In: International conference on computer-aided design, San Jose, pp 357–362
10. Papa DA, Markov IL (2007) Hypergraph partitioning and clustering. In: Gonzalez T (ed) Approximation algorithms and metaheuristics. Chapman & Hall/CRC computer and information science series. Chapman & Hall/CRC, Florida
11. Reda S, Chowdhary A (2006) Effective linear programming based placement methods. In: International symposium on physical design, San Jose, pp 186–191
12. Roy JA, Adya SN, Papa DA, Markov IL (2006) Min-cut floorplacement. IEEE Trans CAD 25(7):1313–1326

Circuit Retiming

Hai Zhou
Electrical Engineering and Computer Science (EECS) Department, Northwestern University, Evanston, IL, USA

Keywords

Min-area retiming; Min-period retiming

Years and Authors of Summarized Original Work

1991; Leiserson, Saxe

Problem Definition

Circuit retiming is one of the most effective structural optimization techniques for sequential circuits. It moves the registers within a circuit without changing its function. Besides clock period, retiming can be used to minimize the number of registers in the circuit. It is also called minimum area retiming problem Leiserson and Saxe [3] started the research on retiming and proposed algorithms for both minimum period and minimum area retiming. Both their algorithms for minimum area and minimum period will be presented here.

The problems can be formally described as follows. Given a directed graph $G = (V, E)$ representing a circuit – each node $v \in V$ represents a gate and each edge $e \in E$ represents a signal passing from one gate to another – with gate delays $d : V \to \mathbb{R}^+$ and register numbers $w : E \to \mathbb{N}$, the minimum area problem asks for a relocation of registers $w' : E \to \mathbb{N}$ such that the number of registers in the circuit is minimum under a given clock period φ. The minimum period problem asks for a solution with the minimum clock period.

Notations

To guarantee that the new registers are actually a relocation of the old ones, a label $r : V \to \mathbb{Z}$ is used to represent how many registers are moved from the outgoing edges to the incoming edges of each node. Using this notation, the new number of registers on an edge (u, v) can be computed as

$$w'[u, v] = w[u, v] + r[v] - r[u].$$

The same notation can be extended from edges to paths. However, between any two nodes u and v, there may be more than one path. Among these

paths, the ones with the minimum number of registers will decide how many registers can be moved outside of u and v. The number is denoted by $W[u, v]$ for any $u, v \in V$, that is,

$$W[u, v] \triangleq \min_{p: u \rightsquigarrow v} \sum_{(x,y) \in p} w[x, y]$$

The maximal delay among all the paths from u to v with the minimum number of registers is also denoted by $D[u, v]$, that is,

$$D[u, v] \triangleq \max_{w[p: u \rightsquigarrow v] = W[u,v]} \sum_{x \in p} d[x]$$

Constraints

Based on the notations, a valid retiming r should not have any negative number of registers on any edge. Such a validity condition is given as

$$P0(r) \triangleq \forall (u, v) \in E : w[u, v] + r[v] - r[u] \geq 0$$

On the other hand, given a retiming r, the minimum number of registers between any two nodes u and v is $W[u, v] - r[u] + r[v]$. This number will not be negative because of the previous constraint. However, when it is zero, there will be a path of delay $D[u, v]$ without any register on it. Therefore, to have a retimed circuit working for clock period φ, the following constraint must be satisfied.

$$P1(r) \triangleq \forall u, v \in V : D[u, v] > \phi \Rightarrow W[u, v] + r[v] - r[u] \geq 1$$

Key Results

The object of the minimum area retiming is to minimize the total number of registers in the circuit, which is given by $\sum_{(u,v) \in E} w'[u, v]$. Expressing $w'[u, v]$ in terms of r, the objective becomes

$$\sum_{v \in V} (\text{in}[v] - \text{out}[v]) * r[v] + \sum_{(u,v) \in E} w[u, v]$$

where $\text{in}[v]$ is the in-degree and $\text{out}[v]$ is the out-degree of node v. Since the second term is a constant, the problem can be formulated as the following integer linear program.

$$\begin{aligned} &\text{Minimize} \sum_{v \in V} (\text{in}[v] - \text{out}[v]) * r[v] \\ &\text{s.t. } w[u, v] + r[v] - r[u] \geq 0 \quad \forall (u, v) \in E \\ &\quad W[u, v] + r[v] - r[u] \geq 1 \quad \forall u, v \in V : D[u, v] > \phi \\ &\quad r[v] \in \mathbb{Z} \quad \forall v \in V \end{aligned}$$

Since the constraints have only difference inequalities with integer-constant terms, solving the relaxed linear program (without the integer constraint) will only give integer solutions. Even better, it can be shown that the problem is the dual of a minimum cost network flow problem and, thus, can be solved efficiently.

Theorem 1 *The integer linear program for the minimum area retiming problem is the dual of the following minimum cost network flow problem.*

$$\begin{aligned} &\text{Minimize} \sum_{(u,v) \in E} w[u, v] * f[u, v] \\ &\quad + \sum_{D[u,v] > \phi} (W[u, v] - 1) * f[u, v] \\ &\text{s.t. } \text{in}[v] + \sum_{(v,w) \in E \vee D[v,w] > \phi} f[v, w] = \text{out}[v] \\ &\quad + \sum_{(u,v) \in ED[u,v] > \phi} f[u, v] \quad \forall v \in V \\ &\quad f[u, v] \geq 0 \quad \forall (u, v) \in ED[u, v] > \phi \end{aligned}$$

From the theorem, it can be seen that the network graph is a dense graph where a new edge (u, v) needs to be introduced for any node pair u, v such that $D[u, v] > \phi$. There may be redundant constraints in the system.

For example, if $W[u, w] = W[u, v] + w[v, w]$ and $D[u, v] > \phi$ then the constraint $W[u, w] + r[w] - r[u] \geq 1$ is redundant, since there are already $W[u, v] + r[v] - r[u] \geq 1$ and $w[v, w] + r[w] - r[v] \geq 0$. However, it may not be easy to check and remove all redundancy in the constraints.

In order to build the minimum cost flow network, it is needed to first compute both matrices W and D. Since $W[u, v]$ is the shortest path from u to v in terms of w, the computation of W can be done by an all-pair shortest paths algorithm such as Floyd-Warshall's algorithm [1]. Furthermore, if the ordered pair $(w[x, y], -d[x])$ is used as the edge weight for each $(x, y) \in E$, an all-pair shortest paths algorithm can also be used to compute both W and D. The algorithm will add weights by component-wise addition and will compare weights by lexicographic ordering.

Leiserson and Saxe's [3] first algorithm for the minimum period retiming was also based on the matrices W and D. The idea was that the constraints in the integer linear program for the minimum area retiming can be checked efficiently by Bellman-Ford's shortest paths algorithm [1], since they are just difference inequalities. This gives a feasibility checking for any given clock period φ. Then the optimal clock period can be found by a binary search on a range of possible periods. The feasibility checking can be done in $O(|V|^3)$ time, thus the runtime of such an algorithm is $O(|V|^3 \log |V|)$.

Their second algorithm got rid of the construction of the matrices W and D. It still used a clock period feasibility checking within a binary search. However, the feasibility checking was done by incremental retiming. It works as follows: Starting with $r = 0$, the algorithm computes the arrival time of each node by the longest paths computation on a DAG (Directed Acyclic Graph). For each node v with an arrival time larger than the given period φ, the $r[v]$ will be increased by one. The process of the arrival time computation and r increasing will be repeated $|V| - 1$ times. After that, if there is still arrival time that is larger than φ, then the period is infeasible. Since the feasibility checking is done in $O(|V||E|)$ time, the runtime for the minimum period retiming is $O(|V|E| \log |V|)$.

Applications

Shenoy and Rudell [7] implemented Leiserson and Saxe's minimum period and minimum area retiming algorithms with some efficiency improvements. For minimum period retiming, they implemented the second algorithm and, in order to find out infeasibility earlier, they introduced a pointer from one node to another where at least one register is required between them. A cycle formed by the pointers indicates the feasibility of the given period. For minimum area retiming, they removed some of the redundancy in the constraints and used the cost-scaling algorithm of Goldberg and Tarjan [2] for the minimum cost flow computation.

As can be seen from the second minimum period retiming algorithm here and Zhou's algorithm [9] in another entry (▸ Circuit Retiming: An Incremental Approach), incremental computation of the longest combinational paths (i.e., those without register on them) is more efficient than constructing the dense graph (via matrices W and D). However, the minimum area retiming algorithm is still based on a minimum cost network flow on the dense graph. A more efficient algorithm based on incremental retiming has recently been designed for the minimum area problem by Wang and Zhou [8].

Experimental Results

Sapatnekar and Deokar [6] and Pan [5] proposed continuous retiming as an efficient approximation for minimum period retiming and reported the experimental results. Maheshwari and Sapatnekar [4] also proposed some efficiency improvements to the minimum area retiming algorithm and reported their experimental results.

Cross-References

▸ Circuit Retiming: An Incremental Approach

Recommended Reading

1. Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms, 2nd edn. MIT, Cambridge

2. Goldberg AV, Taijan RE (1987) Solving minimum cost flow problem by successive approximation. In: Proceedings of ACM symposium on the theory of computing, New York, pp 7–18. Full paper in: Math Oper Res 15:430–466 (1990)
3. Leiserson CE, Saxe JB (1991) Retiming synchronous circuitry. Algorithmica 6:5–35
4. Maheshwari N, Sapatnekar SS (1998) Efficient retiming of large circuits. IEEE Trans Very Large-Scale Integr Syst 6:74–83
5. Pan P (1997) Continuous retiming: algorithms and applications. In: Proceedings of international conference on computer design, Austin. IEEE, Los Alamitos pp 116–121
6. Sapatnekar SS, Deokar RB (1996) Utilizing the retiming- skew equivalence in a practical algorithm for retiming large circuits. IEEE Trans Comput Aided Des 15:1237–1248
7. Shenoy N, Rudell R (1994) Efficient implementation of retiming. In: Proceedings of international conference on computer-aided design, San Jose. IEEE, Los Alamitos, pp 226–233
8. Wang J, Zhou H (2008) An efficient incremental algorithm for min-area retiming. In: Proceedings of design automation conference, Anaheim, CA, pp 528–533
9. Zhou H (2005) Deriving a new efficient algorithm for min-period retiming. In: Asia and South Pacific design automation conference, Shanghai, Jan 2005. ACM, New York

Circuit Retiming: An Incremental Approach

Hai Zhou
Electrical Engineering and Computer Science (EECS) Department, Northwestern University, Evanston, IL, USA

Keywords

Minimum period retiming; Min-period retiming

Years and Authors of Summarized Original Work

2005; Zhou

Problem Definition

Circuit retiming is one of the most effective structural optimization techniques for sequential circuits. It moves the registers within a circuit without changing its function. The minimal period retiming problem needs to minimize the longest delay between any two consecutive registers, which decides the clock period.

The problem can be formally described as follows. Given a directed graph $G = (V, E)$ representing a circuit – each node $v \in V$ represents a gate and each edge $e \in E$ represents a signal passing from one gate to another – with gate delays $d : V \to \mathbb{R}^+$ and register numbers $w : E \to \mathbb{N}$, it asks for a relocation of registers $w' : E \to \mathbb{N}$ such that the maximal delay between two consecutive registers is minimized.

Notations To guarantee that the new registers are actually a relocation of the old ones, a label $r : V \to \mathbb{Z}$ is used to represent how many registers are moved from the outgoing edges to the incoming edges of each node. Using this notation, the new number of registers on an edge (u, v) can be computed as

$$w'[u, v] = w[u, v] + r[v] - r[u].$$

Furthermore, to avoid explicitly enumerating the paths in finding the longest path, another label $t : V \to \mathbb{R}^+$ is introduced to represent the output arrival time of each gate, that is, the maximal delay of a gate from any preceding register. The condition for t to be at least the combinational delays is

$$\forall [u, v] \in E : w'[u, v] = 0 \Rightarrow t[v] \geq t[u] + d[u].$$

Constraints and Objective Based on the notations, a valid retiming r should not have any negative number of registers on any edge. Such a validity condition is given as

$$P0(r) \triangleq \forall (u, v) \in E : w[u, v] + r[v] - r[u] \geq 0.$$

As already stated, the conditions for t to be valid arrival time is given by the following two predicates:

$$P1(t) \triangleq \forall v \in V : t[v] \geq d[v]$$
$$P2(r,t) \triangleq \forall (u,v) \in E : r[u] - r[v] = w[u,v] \Rightarrow t[v] - t[u] \geq d[v].$$

A predicate P is used to denote the conjunction of the above conditions:

$$P(r,t) \triangleq P0(r) \wedge P1(t) \wedge P2(r,t).$$

A minimal period retiming is a solution r, t satisfying the following optimality condition:

$$P3 \triangleq \forall r', t' : P(r', t') \Rightarrow \max(t) \leq \max(t')$$

where

$$\max(t) \triangleq \max_{v \in V}[v].$$

Since only a valid retiming (r', t') will be discussed in the sequel, to simplify the presentation, the range condition $P(r', t')$ will often be omitted; the meaning shall be clear from the context.

Key Results

This section will show how an efficient algorithm is designed for the minimal period retiming problem. Contrary to the usual way of only presenting the final product, i.e., the algorithm, but not the ideas on its design, a step-by-step design process will be shown to finally arrive at the algorithm.

To design an algorithm is to construct a procedure such that it will terminate in finite steps and will satisfy a given predicate when it terminates. In the minimal period retiming problem, the predicate to be satisfied is $P0 \wedge P1 \wedge P2 \wedge P3$. The predicate is also called the *post-condition*. It can be argued that any nontrivial algorithm will have at least one loop; otherwise, the processing length is only proportional to the text length. Therefore, some part of the post-condition will be iteratively satisfied by the loop, while the remaining part will be initially satisfied by an initialization and made invariant during the loop.

The first decision needed to make is to partition the post-condition into possible invariant and loop goal. Among the four conjuncts, the predicate $P3$ gives the optimality condition and is the most complex one. Therefore, it will be used as a loop goal. On the other hand, the predicates $P0$ and $P1$ can be easily satisfied by the following simple initialization:

$$r, t := 0, d.$$

Based on these, the plan is to design an algorithm with the following scheme:

$$\begin{aligned} &r, t := 0, d \\ &\text{do}\{P0 \wedge P1\} \\ &\quad \neg P2 \rightarrow \text{update } t \\ &\quad \neg P3 \rightarrow \text{update } r \\ &\text{od}\{P0 \wedge P1 \wedge P2 \wedge P3\}. \end{aligned}$$

The first command in the loop can be refined as

$$\begin{aligned} \exists (u,v) \in E : r[u] - r[v] = w[u,v] \wedge t[v] &\\ - t[u] < d[v] \rightarrow t[v] &\\ := t[u] + d[v]. & \end{aligned}$$

This is simply the Bellman-Ford relaxations for computing the longest paths.

The second command is more difficult to refine. If $\neg P3$, that is, there exists another valid retiming r', t' such that $\max(t) > \max(t')$, then on any node v such that $t[v] = \max(t)$ it must have $t'[v] < t[v]$. One property known on these nodes is

$$\begin{aligned} &\forall v \in V : t'[v] < t[v] \\ \Rightarrow &(\exists u \in V : r[u] - r[v] > r'[u] - r'[v]), \end{aligned}$$

which means that if the arrival time of v is smaller in another retiming r', t', then there must be a node u such that r' gives more registers between u and v. In fact, one such a u is the starting node of the longest combinational path to v that gives the delay of $t[v]$.

To reduce the clock period, the variable r needs to be updated to make it closer to r'. It should be noted that it is not the absolute values of r but their differences that are relevant in the retiming. If r, t is a solution to a retiming

problem, then $r+c, t$, where $c \in \mathbb{Z}$ is an arbitrary constant, is also a solution. Therefore r can be made "closer" to r' by allocating more registers between u and v, that is, by either decreasing $r[u]$ or increasing $r[v]$. Notice that v can be easily identified by $t[v] = \max(t)$. No matter whether $r[v]$ or $r[u]$ is selected to change, the amount of change should be only one since r should not be overadjusted. Thus, after the adjustment, it is still true that $r[v] - r[u] \leq r'[v] - r'[u]$ or equivalently $r[v] - r'[v] \leq r[u] - r'[u]$. Since v is easy to identify, $r[v]$ is selected to increase. The arrival time $t[v]$ can be immediately reduced to $d[v]$. This gives a refinement of the second command:

$$\neg P3 \wedge P2 \wedge \exists v \in V : t[v] = \max(t) \rightarrow r[v], t[v] := r[v] + 1, d[v].$$

Since registers are moved in the above operation, the predicate $P2$ may be violated. However, the first command will take care of it. That command will increase t on some nodes; some may even become larger than $\max(t)$ before the register move. The same reasoning using r', t' shows that their r values shall be increased, too. Therefore, to implement this as-soon-as-possible (ASAP) increase of r, a snapshot of $\max(t)$ needs to be taken when $P2$ is valid. Physically, such a snapshot records one feasible clock period ϕ and can be implemented by adding one more command in the loop:

$$P2 \wedge \phi > \max(t) \rightarrow \phi := \max(t).$$

However, such an ASAP operation may increase $r[u]$ even when $w[u, v] - r[u] + r[v] = 0$ for an edge (u, v). It means that $P0$ may no longer be an invariant. But moving $P0$ from invariant to loop goal will not cause a problem since one more command can be added in the loop to take care of it:

$$\exists (u, v) \in E : r[u] - r[v] > w[u, v] \rightarrow r[v] := r[u] - w[u, v].$$

Putting all things together, the algorithm now has the following form:

$r, t, \phi := 0, d, \infty;$
do$\{P1\}$
$\exists (u, v) \in E : r[u] - r[v] = w[u, v]$
$\wedge t[v] - t[u] < d[v] \rightarrow t[v] := t[u] + d[v]$
$\neg P3 \wedge \exists v \in V : t[v] \geq \phi$
$\rightarrow r[v], t[v] := r[v] + 1, d[v]$
$P0 \wedge P2 \wedge \phi > \max(t) \rightarrow \phi := \max(t)$
$\exists (u, v) \in E : r[u] - r[v] > w[u, v]$
$\rightarrow r[v] := r[u] - w[u, v]$
od$\{P0 \wedge P1 \wedge P2 \wedge P3\}$.

The remaining task to complete the algorithm is how to check $\neg P3$. From previous discussion, it is already known that $\neg P3$ implies that there is a node u such that $r[u] - r'[u] \geq r'[v] - r'[v]$ every time after $r[v]$ is increased. This means that $\max_{v \in V} r[v] - r'[v]$ will not increase. In other words, there is at least one node v whose $r[v]$ will not change. Before $r[v]$ is increased, it also has $w_{u \rightsquigarrow v} - r[u] + r[v] \leq 0$, where $w_{u \rightsquigarrow v} \geq 0$ is the original number of registers on one path from u to v, which gives $r[v] - r[u] \leq 1$ even after the increase of $r[v]$. This implies that there will be at least $i + 1$ nodes whose r is at most i for $0 \leq i < |V|$. In other words, the algorithm can keep increasing r and when there is any r reaching $|V|$ it shows that $P3$ is satisfied. Therefore, the complete algorithm will have the following form:

$r, t, \phi := 0, d, \infty;$
do$\{P1\}$
$\exists (u, v) \in E : r[u] - r[v] = w[u, v]$
$\wedge t[v] - t[u] < d[v] \rightarrow t[v] := t[u] + d[v]$
$(\forall v \in V : r[v] < |V|)$
$\wedge \exists v \in V : t[v] \geq \phi \rightarrow r[v], t[v] := r[v] + 1, d[v]$
$(\exists v \in V : r[v] \geq |V|)$
$\wedge \exists v \in V : t[v] \geq \phi \rightarrow r[v], t[v] := r[v] + 1, d[v]$
$P0 \wedge P2 \wedge \phi > \max(t) \rightarrow \phi := \max(t)$
$\exists (u, v) \in E : r[u] - r[v] > w[u, v]$
$\rightarrow r[v] := r[u] - w[u, v]$
od$\{P0 \wedge P1 \wedge P2 \wedge P3\}$.

The correctness of the algorithm can be proved easily by showing that the invariant $P1$ is maintained and the negation of the guards implies

$P0 \wedge P2 \wedge P3$. The termination is guaranteed by the monotonic increase of r and an upper bound on it. In fact, the following theorem gives its worst-case runtime.

Theorem 1 *The worst-case running time of the given retiming algorithm is upper bounded by* $O(|V|^2|E|)$.

The runtime bound of the retiming algorithm is got under the worst-case assumption that each increase on r will trigger a timing propagation on the whole circuit ($|E|$ edges). This is only true when the r increase moves all registers in the circuit. However, in such a case, the r is upper bounded by 1, thus the running time is not larger than $O(|V||E|)$. On the other hand, when the r value is large, the circuit is partitioned by the registers into many small parts, thus the timing propagation triggered by one r increase is limited within a small tree.

Applications

In the basic algorithm, the optimality $P3$ is verified by an $r[v] \geq |V|$. However, in most cases, the optimality condition can be discovered much earlier. Since each time $r[v]$ is increased, there must be a "safeguard" node u such that $r[u] - r'[u] \geq r[v] - r'[v]$ after the operation. Therefore, if a pointer is introduced from v to u when $r[v]$ is increased, the pointers cannot form a cycle under $\neg P3$. In fact, the pointers will form a forest where the roots have $r = 0$ and a child can have an r at most one larger than its parent. Using a cycle by the pointers as an indication of $P3$, instead of an $r[v] \geq |V|$, the algorithm can have much better practical performance.

Retiming is usually used to optimize either the clock period or the number of registers in the circuit. The discussed algorithm solves only the minimal period retiming problem. The retiming problem for minimizing the number of registers under a given period has been solved by Leiserson and Saxe [1] and is presented in another entry in this encyclopedia. Their algorithm reduces the problem to the dual of a minimal cost network problem on a denser graph. An efficient iterative algorithm similar to Zhou's algorithm has been designed for the minimal register problem recently [3].

Experimental Results

Experimental results are reported by Zhou [4] which compared the runtime of the algorithm with an efficient heuristic called ASTRA [2]. The results on the ISCAS89 benchmarks are reproduced here in Table 1 from [4], where columns A and B are the running time of the two stages in ASTRA.

Cross-References

▸ Circuit Retiming

Circuit Retiming: An Incremental Approach, Table 1 Experimental results

Name	#gates	Clock period		$\sum r$	#updates	Time(s)	ASTRA	
		Before	After				A(s)	B(s)
s1423	490	166	127	808	7,619	0.02	0.03	0.02
s1494	558	89	88	628	7,765	0.02	0.01	0.01
s9234	2,027	89	81	2,215	76,943	0.12	0.11	0.09
s9234.1	2,027	89	81	2,164	77,644	0.16	0.11	0.10
s13207	2,573	143	82	4,086	28,395	0.12	0.38	0.12
s15850	3,448	186	77	12,038	99,314	0.36	0.43	0.17
s35932	12,204	109	100	16,373	108,459	0.28	0.24	0.65
s38417	8,709	110	56	9,834	155,489	0.58	0.89	0.64
s38584	11,448	191	163	19,692	155,637	0.41	0.50	0.67
s38584.1	11,448	191	183	9,416	114,940	0.48	0.55	0.78

Recommended Reading

1. Leiserson CE, Saxe JB (1991) Retiming synchronous circuitry. Algorithmica 6:5–35
2. Sapatnekar SS, Deokar RB (1996) Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits. IEEE Trans Comput Aided Des 15:1237–1248
3. Wang J, Zhou H (2008) An efficient incremental algorithm for min-area retiming. In: Proceedings of the design automation conference, Anaheim, pp 528–533
4. Zhou H (2005) Deriving a new efficient algorithm for min-period retiming. In: Asia and South Pacific design automation conference, Shanghai, Jan 2005

Clique Enumeration

Etsuji Tomita
The Advanced Algorithms Research Laboratory,
The University of Electro-Communications,
Chofu, Tokyo, Japan

Keywords

Enumeration; Maximal clique; Maximal independent set; Time complexity

Years and Authors of Summarized Original Work

1977; Tsukiyama, Ide, Ariyoshi, Shirakawa
2004; Makino, Uno
2006; Tomita, Tanaka, Takahashi

Problem Definition

We discuss a simple undirected and connected graph $G = (V, E)$ with a finite set V of vertices and a finite set $E \subseteq V \times V$ of edges. A pair of vertices v and w is said to be adjacent if $(v, w) \in E$. For a subset $R \subseteq V$ of vertices, $G(R) = (R, E \cap (R \times R))$ is an induced subgraph. An induced subgraph $G(Q)$ is said to be a *clique* if $(v, w) \in E$ for all $v, w \in Q \subseteq V$ with $v \neq w$. In this case, we may simply state that Q is a clique. In particular, a clique that is not properly contained in any other clique is called *maximal*. An induced subgraph $G(S)$ is said to be an *independent set* if $(v, w) \notin E$ for all $v, w \in S \subseteq V$. For a vertex $v \in V$, let $\Gamma(v) = \{w \in V | (v, w) \in E\}$. We call $|\Gamma(v)|$ the degree of v.

The problem is to enumerate all maximal cliques of the given graph $G = (V, E)$. It is equivalent to enumerate all maximal independent sets of the *complementary graph* $\bar{G} = \left(V, \bar{E}\right)$, where $\bar{E} = \{(v, w) \in V \times V | (v, w) \notin E, v \neq w\}$.

Key Results

Efficient Algorithms for Clique Enumeration

Efficient algorithms to solve the problem can be found in the following approaches (1) and (2).

(1) Clique Enumeration by Depth-First Search with Pivoting Strategy

The basis of the first approach is a simple depth-first search. It begins with a clique of size 0 and continues with finding all of the progressively larger cliques until they can be verified as maximal. Formally, this approach maintains a global variable $Q = \{p_1, p_2, \ldots, p_d\}$ that consists of vertices of a current clique found so far. Let

$$SUBG = V \cap \Gamma(p_1) \cap \Gamma(p_2) \cap \cdots \cap \Gamma(p_d).$$

We begin the algorithm by letting $Q = \varnothing$ and $SUBG := V$ (the set of all vertices). We select a certain vertex p from $SUBG$ and add p to $Q (Q := Q \cup \{p\})$. Then we compute $SUBG_p := SUBG \cap \Gamma(p)$ as the new set of vertices in question. In particular, the first selected vertex $u \in SUBG$ is called a *pivot*. This procedure (EXPAND()) is applied recursively while $SUBG_p \neq \varnothing$.

When $SUBG_p = \varnothing$ is reached, Q constitutes a *maximal* clique. We then backtrack by removing the lastly inserted vertex from Q and $SUBG$. We select a new vertex p from the resulting $SUBG$ and continue the same procedure until $SUBG = \emptyset$. This process can be represented by a depth-first *search forest*. See Fig. 2b as an

```
procedure CLIQUES(G)
begin                                                        /* Q := ∅ */
 1 : EXPAND(V,V)
end of CLIQUES

      procedure EXPAND(SUBG, CAND)
      begin
 2 :    if SUBG = ∅
 3 :      then print("clique,")                    /* Q is a maximal clique */
 4 :      else u := a vertex u in SUBG that maximizes | CAND ∩ Γ(u) |;   /* pivot */
 5 :           while CAND − Γ(u) ≠ ∅
 6 :             do q := a vertex in (CAND − Γ(u));
 7 :                print (q,",");                          /* Q := Q ∪{ q } */
 8 :                SUBG_q := SUBG ∩ Γ(q);
 9 :                CAND_q := CAND ∩ Γ(q);
10 :                EXPAND(SUBG_q, CAND_q);
11 :                CAND := CAND −{ q};
12 :                print ("back,")                       /* Q := Q −{ q } */
                  od
      fi
      end of EXPAND
```

Clique Enumeration, Fig. 1 Algorithm CLIQUES

example of an essential part of a search forest. It clearly generates all maximal cliques.

The above-generated maximal cliques, however, could contain duplications or nonmaximal ones, so we *prune* unnecessary parts of the search forest as in the Bron-Kerbosch algorithm [3].

First, let *FINI* be a subset of vertices of *SUBG* that have already been processed by the algorithm. (*FINI* is short for *finished*.) Then we denote by *CAND* the set of remaining candidates for expansion: $CAND := SUBG - FINI$, where for two sets X and Y, $X - Y = \{v | v \in X \text{ and } v \notin Y\}$. At the beginning, $FINI := \varnothing$ and $CAND := SUBG$. In the subgraph $G(SUBG_q)$ with $SUBG_q := SUBG \cap \Gamma(q)$, let

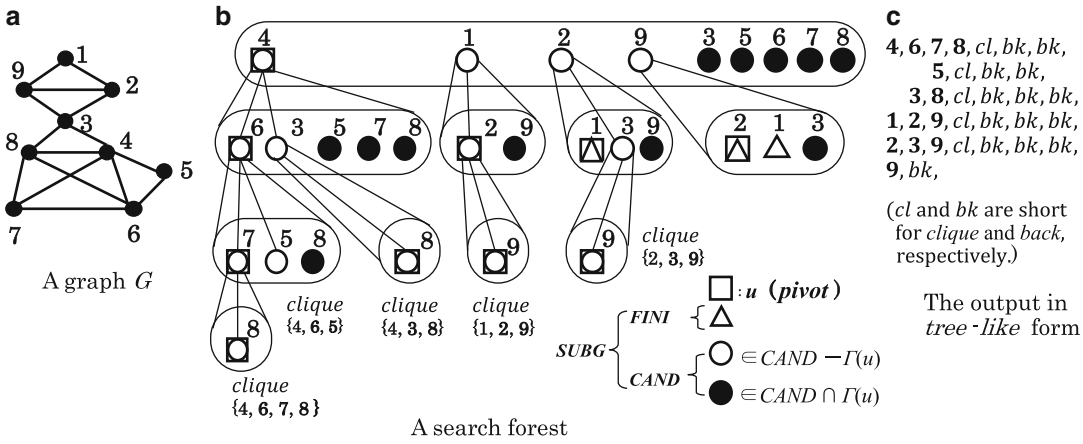
$$FINI_q := SUBG_q \cap FINI,$$
$$CAND_q := SUBG_q - FINI_q.$$

Then only the vertices in $CAND_q$ can be candidates for expanding the clique $Q \cup \{q\}$ to find *new* larger cliques.

Second, for the first selected pivot u in *SUBG*, any maximal clique R in $G(SUBG \cap \Gamma(u))$ is not maximal in $G(SUBG)$, since $R \cup \{u\}$ is a larger clique in $G(SUBG)$. Therefore, searching for maximal cliques from $SUBG \cap \Gamma(u)$ should be excluded.

When the previously described pruning method is also taken into consideration, we find that the only search subtrees to be expanded are from the vertices in $(SUBG - SUBG \cap \Gamma(u)) - FINI = CAND - \Gamma(u)$. Here, in order to minimize $|CAND - \Gamma(u)|$, we choose the pivot $u \in SUBG$ to be the one that *maximizes* $|CAND \cap \Gamma(u)|$. This is *crucial* to establish the *optimality* of the worst-case time complexity of the algorithm. This kind of pivoting strategy was proposed by Tomita et al. [11]. (Recommended Reading [11] was reviewed by Pardalos and Xue [10] and Bomze et al. [2].)

The algorithm CLIQUES by Tomita et al. [12] is shown in Fig. 1. It enumerates all maximal cliques based upon the above approach, but all maximal cliques enumerated are presented in a tree-like form. Here, if Q is a *maximal* clique that is found at statement 2, then the algorithm only prints out a string of characters *clique* instead of Q itself at statement 3. Otherwise, it is impossible to achieve the optimal worst-case running time. Instead, in addition to printing *clique* at statement 3, we print out q followed by a *comma* at statement 7 every time q is picked out as a new element of a larger clique, and we print out a string of characters *back* at statement 12 after q is moved from *CAND* to *FINI* at statement 11. We can easily obtain a tree representation of all



Clique Enumeration, Fig. 2 An example run of CLIQUES [12]. (**a**) A graph G. (**b**) A search forest. (**c**) The output in *tree-like* form

the maximal cliques from the sequence printed by statements 3, 7, and 12. The output in a tree-like format is also important *practically* since it saves space in the output file. An example run of CLIQUES to Fig. 2a is shown in Fig. 2b, c with appropriate indentations.

The worst-case time complexity of CLIQUES was proved to be $O(3^{n/3})$ for an n-vertex graph [11, 12]. This is *optimal* as a function of n since there exist up to $3^{n/3}$ cliques in an n-vertex graph [9].

Eppstein et al. [5] used this approach and proposed an algorithm for enumerating all maximal cliques that runs in time $O(dn3^{d/3})$ for an n-vertex graph G, where d is the *degeneracy* of G that is defined to be the smallest number such that every subgraph of G contains a vertex of degree at most d. If graph G is *sparse*, d can be much smaller than n and hence $O(dn3^{d/3})$ can be much smaller than $O(3^{n/3})$.

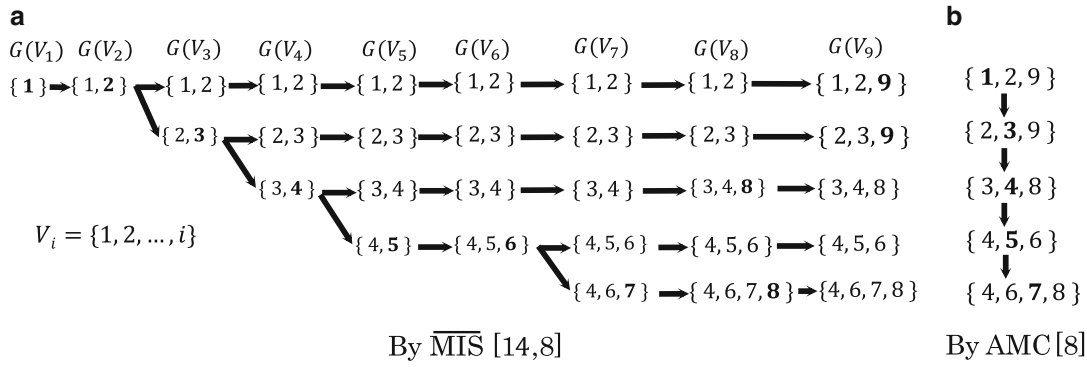
(2) Clique Enumeration by Reverse Search

The second approach is regarded to be based upon the *reverse search* that was introduced by Avis and Fukuda [1] to solve enumeration problems efficiently.

Given the graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$ where $n = |V|$, let $V_i = \{v_1, v_2, \ldots, v_i\}$. Then $\{v_1\}$ is simply a maximal clique in $G(V_1)$. All the maximal cliques in $G(V_i)$ are enumerated based on those in $G(V_{i-1})$ step by step for $i = 2, 3, \ldots, n$. The process forms an *enumeration tree* whose root is $\{v_1\}$, where the root is considered at depth 1 of the enumeration tree for the sake of simplicity. The children at depth i are all the maximal cliques in $G(V_i)$ for $i = 2, 3, \ldots, n$. For two subsets $X, Y \subseteq V$, we say that X precedes Y in lexicographic order if for $v_i \in (X - Y) \cup (Y - X)$ with the minimum index i it holds that $v_i \in X$.

Let Q be a maximal clique in $G(V_{i-1})$. If v_i is adjacent to all vertices in Q, then $Q \cup \{v_i\}$ is the only child of Q at depth i. Otherwise, Q itself is the first child of Q at depth i. In addition, if $Q \cap \Gamma(v_i) \cup \{v_i\}$ is a *maximal* clique in $G(V_i)$, then it is a candidate for the second child of Q at depth i. The unique parent of $Q \cap \Gamma(v_i) \cup \{v_i\}$ is defined to be the *lexicographically first* maximal clique in $G(V_{i-1})$ that contains $Q \cap \Gamma(v_i)$. (In general, there exist multiple numbers of distinct maximal cliques that contain $Q \cap \Gamma(v_i)$ at depth $i - 1$.)

The algorithm of Tsukiyama et al. [14] traverses the above enumeration tree in a depth-first way. Such a traversal is considered to be reverse search [8]. To be more precise, the algorithm MIS in [14] is to enumerate all maximal independent sets, and we are concerned here with its complementary algorithm in [8] that enumerates all maximal cliques, which we call here $\overline{\text{MIS}}$. An example run of $\overline{\text{MIS}}$ to Fig. 2a is shown in Fig. 3a. Algorithms MIS and $\overline{\text{MIS}}$ run in time $O(m'n)$ and



Clique Enumeration, Fig. 3 Enumeration trees for Fig. 2a in reverse search. (**a**) By $\overline{\text{MIS}}$ [14, 8]. (**b**) By AMC [8]

$O(mn)$ per maximal clique, respectively, where $n = |V|, m = |E|$, and $m' = \left|\bar{E}\right|$ [8, 14].

Chiba and Nishizeki [4] reduced the time complexity of $\overline{\text{MIS}}$ to $O(a(G)m)$ per maximal clique, where $a(G)$ is the *arboricity* of G with $m/(n-1) \leq a(G) \leq O(m^{1/2})$ for a connected graph G. Johnson et al. [7] presented an algorithm that enumerates all maximal cliques in lexicographic order in time $O(mn)$ per maximal clique [7, 8].

Makino and Uno [8] proposed new algorithms that are based on the algorithm of Tsukiyama et al. [14]. Let $C(Q)$ denote the lexicographically first maximal clique containing Q in graph G. The root of their enumeration tree is the lexicographically first maximal clique Q_0 in G. For a maximal clique $Q \neq Q_0$ in the enumeration tree, define the parent of Q to be $C(Q \cap V_i)$ where i is the maximum index such that $C(Q \cap V_i) \neq Q$. Such a parent uniquely exists for every $Q \neq Q_0$. In the enumeration tree, $Q' = C(Q \cap V_j \cap \Gamma(v_j) \cup \{v_j\})$ is a child of Q if and only if Q is a parent of Q'. (In general, a parent has at most $|V|$ children.) This concludes the description of the enumeration tree of AllMaxCliques (AMC for short) in [8]. An example run to Fig. 2a is shown in Fig. 3b, where the bold-faced vertex is the minimum i such that $Q \cap V_i = Q$. Algorithm AMC runs in time $O(M(n))$ per maximal clique, where $M(n)$ denotes the time required to multiply two $n \times n$ matrices. Another algorithm in [8] runs in time $O(\Delta^4)$ per maximal clique, where Δ is the maximum degree of G. Here, if G is sparse, then Δ can be small. In addition, they presented an algorithm that enumerates all *maximal bipartite cliques* in a bipartite graph in time $O(\Delta^3)$ per maximal bipartite clique.

Applications

Clique enumeration has diverse applications in clustering, data mining, information retrieval, bioinformatics, computer vision, wireless networking, computational topology, and many other areas. Here, one of Makino and Uno's algorithms [8] was successfully applied for enumerating *frequent closed itemsets* [16]. See Recommended Reading [2, 5, 6, 8, 10, 12, 13, 16] for details. For practical applications, enumeration of *pseudo cliques* is sometimes more important [15].

Experimental Results

Experimental Results are shown in Recommended Reading [12, 6, 14, 8]. Tomita et al.'s algorithm CLIQUES [12] is fast especially for graphs with high and medium density. Eppstein et al.'s algorithm [5] is effective for very large and sparse graphs [6]. Makino and Uno's algorithms [8] can be fast for sparse graphs especially when they have a small number of maximal cliques.

Cross-References

▸ Reverse Search; Enumeration Algorithms

Recommended Reading

1. Avis D, Fukuda K (1996) Reverse search for enumeration. Discret Appl Math 65:21–46
2. Bomze IM, Budinich M, Pardalos PM, Pelillo M (1999) The maximum clique problem. In: Du D-Z, Pardalos PM (eds) Handbook of combinatorial optimization, supplement, vol A. Kluwer Academic, Dordrecht, pp 1–74
3. Bron C, Kerbosch J (1973) Algorithm 457, finding all cliques of an undirected graph. Commun ACM 16:575–577
4. Chiba N, Nishizeki T (1985) Arboricity and subgraph listing algorithms. SIAM J Comput 14: 210–223
5. Eppstein D, Löffler M, Strash D (2010) Listing all maximal cliques in sparse graphs in near-optimal time. In: ISAAC 2010, Jeju Island. Lecture notes in computer science, vol 6506, pp 403–414
6. Eppstein D, Strash D (2011) Listing all maximal cliques in large sparse real-world graphs. In: SEA 2011, Chania. Lecture notes in computer science, vol 6630, pp 364–375
7. Johnson DS, Yanakakis M, Papadimitriou CH (1998) On generating all maximal independent sets. Inf Process Lett 27:119–123
8. Makino K, Uno T (2004) New algorithms for enumerating all maximal cliques. In: SWAT 2004, Humlebaek. Lecture notes in computer science, vol 3111, pp 260–272
9. Moon JW, Moser L (1965) On cliques in graphs. Isr J Math 3:23–28
10. Pardalos PM, Xue J (1994) The maximum clique problem. J Glob Optim 4:301–328
11. Tomita E, Tanaka A, Takahashi H (1988) The worst-case time complexity for finding all maximal cliques. Technical Report of the University of Electro-Communications, UEC-TR-C5(2)
12. Tomita E, Tanaka A, Takahashi H (2006) The worst-case time complexity for generating all maximal cliques and computational experiments. Theor Comput Sci 363 (Special issue on COCOON 2004, Jeju Island. Lecture notes in computer science, 3106): 28–42
13. Tomita E, Akutsu T, Matsunaga T (2011) Efficient algorithms for finding maximum and maximal cliques: Effective tools for bioinformatics. In: Laskovski AN (ed) Biomedical engineering, trends in electronics, communications and software. InTech, Rijeka, pp 625–640. Available from: http://www.intechopen.com/articles/show/title/efficient-algorithms-for-finding-maximum-and-maximal-cliques-effective-tools-for-bioinformatics
14. Tsukiyama S, Ide M, Ariyoshi H, Shirakawa I (1977) A new algorithm for generating all the maximal independent sets. SIAM J Comput 6:505–517
15. Uno T (2010) An efficient algorithm for solving pseudo clique enumeration problem. Algorithmica 56:3–16
16. Uno T, Asai T, Arimura H, Uchida Y (2003) LCM: an efficient algorithm for enumerating frequent closed item sets. In: Workshop on frequent itemset mining implementations (FIMI), Melbourne

C

Clock Synchronization

Boaz Patt-Shamir
Department of Electrical Engineering, Tel-Aviv University, Tel-Aviv, Israel

Years and Authors of Summarized Original Work

1994; Patt-Shamir, Rajsbaum

Problem Definition

Background and Overview

Coordinating processors located in different places is one of the fundamental problems in distributed computing. In his seminal work, Lamport [4, 5] studied the model where the only source of coordination is message exchange between the processors; the time that elapses between successive steps at the same processor, as well as the time spent by a message in transit, may be arbitrarily large or small. Lamport observed that in this model, called the *asynchronous model*, temporal concepts such as "past" and "future" are derivatives of causal dependence, a notion with a simple algorithmic interpretation. The work of Patt-Shamir and Rajsbaum [10] can be viewed as extending Lamport's qualitative treatment with quantitative concepts. For example, a statement like "event a happened before event b" may be refined to a statement like "event a happened at least 2 time units and at most 5 time units before event b". This is in contrast to most previous theoretical work, which focused on the linear-programming aspects of clock synchronization (see below).

The basic idea in [10] is as follows. First, the framework is extended to allow for upper

and lower bounds on the time that elapses between pairs of events, using the system's *real-time specification*. The notion of real-time specification is a very natural one. For example, most processors have local clocks, whose rate of progress is typically bounded with respect to real time (these bounds are usually referred to as the clock's "drift bounds"). Another example is send and receive events of a given message: It is always true that the receive event occurs before the send event, and in many cases, tighter lower and upper bounds are available. Having defined real-time specification, [10] proceeds to show how to combine these local bounds global bounds in the best possible way using simple graph-theoretic concepts. This allows one to derive optimal protocols that say, for example, what is the current reading of a remote clock. If that remote clock is the standard clock, then the result is optimal clock synchronization in the common sense (this concept is called "external synchronization" below).

Formal Model

The system consists of a fixed set of interconnected *processors*. Each processor has a *local clock*. An *execution* of the system is a sequence of events, where each event is either a *send* event, a *receive* event, or an *internal* event. Regarding communication, it is only assumed that each receive event of a message m has a unique corresponding send event of m. This means that messages may be arbitrarily lost, duplicated or reordered, but not corrupted. Each event e occurs at a single specified processor, and has two real numbers associated with it: its *local time*, denoted $\mathrm{LT}(e)$, and its *real time*, denoted $\mathrm{RT}(e)$. The local time of an event models the reading of the local clock when that event occurs, and the local processor may use this value, e.g., for calculations, or by sending it over to another processor. By contrast, the real time of an event is not observable by processors: it is an abstract concept that exists only in the analysis.

Finally, the real-time properties of the system are modeled by a pair of functions that map each pair of events to $\mathbb{R} \cup \{-\infty, \infty\}$: given two events e and e', $L(e, e') = \ell$ means that $\mathrm{RT}(e') - \mathrm{RT}(e) \geq \ell$, and $H(e, e') = h$ means that $\mathrm{RT}(e') - \mathrm{RT}(e) \leq h$, i.e., that the number of (real) time units since the occurrence of event e until the occurrence of e' is at least ℓ and at most h. Without loss of generality, it is assumed that $L(e, e') = -H(e', e)$ for all events e, e' (just use the smaller of them). Henceforth, only the upper bounds function H is used to represent the real-time specification.

Some special cases of real time properties are particularly important. In a completely asynchronous system, $H(e', e) = 0$ if either e occurs before e' in the same processor, or if e and e' are the send and receive events, respectively, of the same message. (For simplicity, it is assumed that two ordered events may have the same real time of occurrence.) In all other cases $H(e, e') = \infty$. On the other extreme of the model spectrum, there is the *drift-free* clocks model, where all local clocks run at exactly the rate of real time. Formally, in this case $H(e, e') = \mathrm{LT}(e') - \mathrm{LT}(e)$ for any two events e and e' occurring at the same processor. Obviously, it may be the case that only some of the clocks in the system are drift-free.

Algorithms

In this work, message generation and delivery is completely decoupled from message information. Formally, messages are assumed to be generated by some "send module", and delivered by the "communication system". The task of algorithms is to add contents in messages and state variables in each node. (The idea of decoupling synchronization information from message generation was introduced in [1].) The algorithm only has local information, i.e., contents of the local state variables and the local clock, as well as the contents of the incoming message, if we are dealing with a receive event. It is also assumed that the real time specification is known to the algorithm. The conjunction of the events, their and their local times (but not their real times) is called as the *view* of the given execution. Algorithms, therefore, can only use as input the view of an execution and its real time specification.

Problem Statement

The simplest variant of clock synchronization is *external synchronization*, where one of the processors, called the source, has a drift-free clock, and the task of all processors is to maintain the tightest possible estimate on the current reading of the source clock. This formulation corresponds to the Newtonian model, where the processors reside in a well-defined time coordinate system, and the source clock is reading the standard time. Formally, in external synchronization each processor v has two output variables Δ_v and ε_v; the estimate of v of the source time at a given state is $LT_v + \Delta_v$, where LT_v is the current local time at v. The algorithm is required to guarantee that the difference between the source time and it estimate is at most ε_v (note that Δ_v, as well as ε_v, may change dynamically during the execution). The performance of the algorithm is judged by the value of the ε_v variables: the smaller, the better.

In another variant of the problem, called *internal synchronization*, there is no distinguished processor, and the requirement is essentially that all clocks will have values which are close to each other. Defining this variant is not as straightforward, because trivial solutions (e.g., "set all clocks to 0 all the time") must be disqualified.

Key Results

The key construct used in [10] is the *synchronization graph* of an execution, defined by combining the concepts of local times and real-time specification as follows.

Definition 1 Let β be a view of an execution of the system, and let H be a real time specification for β. The *synchronization graph* generated by β and H is a directed weighted graph $\Gamma_{\beta H} = (V, E, w)$, where V is the set of events in β, and for each ordered pair of events p q in β such that $H(p,q) < \infty$, there is a directed edge $(p,q) \in E$. The *weight* of an edge (p, q) is $w(p,q) \stackrel{\text{def}}{=} H(p,q) - \mathrm{LT}(p) + \mathrm{LT}(q)$.

The natural concept of *distance* from an event p to an event q in a synchronization graph Γ, denoted $d_\Gamma(p,q)$, is defined by the length of the shortest weight path from p to q, or infinity if q is not reachable from p. Since weights may be negative, one has to prove that the concept is well defined: indeed, it is shown that if $\Gamma_{\beta H}$ is derived from an execution with view β that satisfies real time specification H, then $\Gamma_{\beta H}$ does not contain directed cycles of negative weight.

The main algorithmic result concerning synchronization graphs is summarized in the following theorem.

Theorem 1 *Let α be an execution with view β. Then α satisfies the real time specification H if and only if* $\mathrm{RT}(p) - \mathrm{RT}(q) \leq d_\Gamma(p,q) + \mathrm{LT}(p) - \mathrm{LT}(q)$ *for any two events p and q in $\Gamma_{\beta H}$.*

Note that all quantities in the r.h.s. of the inequality are available to the synchronization algorithm, which can therefore determine upper bounds on the real time that elapses between events. Moreover, these bounds are the best possible, as implied by the next theorem.

Theorem 2 *Let $\Gamma_{\beta H} = (V, E, w)$ be a synchronization graph obtained from a view β satisfying real time specification H. Then for any given event $p_0 \in V$, and for any finite number $N > 0$, there exist executions α_0 and α_1 with view β, both satisfying H, and such that the following real time assignments hold.*

- *In α_0, for all $q \in V$ with $d_\Gamma(q, p_0) < \infty$,$\mathrm{RT}_{\alpha_0}(q) = \mathrm{LT}(q) + d_\Gamma(q, p_0)$, and for all $q \in V$ with $d_\Gamma(q, p_0) = \infty$,$\mathrm{RT}_{\alpha_0}(q) > \mathrm{LT}(q) + N$.*
- *In α_1, for all $q \in V$ with $d_\Gamma(p_0, q) < \infty$,$\mathrm{RT}_{\alpha_1}(q) = \mathrm{LT}(q) - d_\Gamma(p_0, q)$, and for all $q \in V$ with $d_\Gamma(p_0, q) = \infty$,$\mathrm{RT}_{\alpha_1}(q) < \mathrm{LT}(q) - N$.*

From the algorithmic viewpoint, one important drawback of results of Theorems 1 and 2 is that they depend on the view of an execution, which may grow without bound. Is it really necessary?

The last general result in [10] answers this question in the affirmative. Specifically, it is shown that in some variant of the *branching program* computational model, the space complexity of any synchronization algorithm that works with arbitrary real time specifications cannot be bounded by a function of the system size. The result is proved by considering multiple scenarios on a simple system of four processors on a line.

Later Developments

Based on the concept of synchronization graph, Ostrovsky and Patt-Shamir present a refined general optimal algorithm for clock synchronization [9]. The idea in [9] is to discard parts of the synchronization graphs that are no longer relevant. Roughly speaking, the complexity of the algorithm is bounded by a polynomial in the system size and the ratio of processors speeds.

Much theoretical work was invested in the internal synchronization variant of the problem. For example, Lundelius and Lynch [7] proved that in a system of n processors with full connectivity, if message delays can take arbitrary values in [0, 1] and local clocks are drift-free, then the best synchronization that can be guaranteed is $1 - \frac{1}{n}$. Helpern et al. [3] extended their result to general graphs using linear-programming techniques. This work, in turn, was extended by Attiya et al. [1] to analyze any given execution (rather than only the worst case for a given topology), but the analysis is performed off-line and in a centralized fashion. The work of Patt-Shamir and Rajsbaum [10] extended the "per execution" viewpoint to on-line distributed algorithms, and shifted the focus of the problem to external synchronization.

Recently, Fan and Lynch [2] proved that in a line of n processors whose clocks may drift, no algorithm can guarantee that the difference between the clock readings of all pairs of neighbors is $o(\log n / \log \log n)$.

Clock synchronization is very useful in practice. See, for example, Liskov [6] for some motivation. It is worth noting that the Internet provides a protocol for external clock synchronization called NTP [8].

Applications

Theorem 1 immediately gives rise to an algorithm for clock synchronization: every processor maintains a representation of the synchronization graph portion known to it. This can be done using a full information protocol: In each outgoing message this graph is sent, and whenever a message arrives, the graph is extended to include the new information from the graph in the arriving message. By Theorem 2, the synchronization graph obtained this way represents at any point in time all information available required for optimal synchronization. For example, consider external synchronization. Directly from definitions it follows that all events associated with a drift-free clock (such as events in the source node) are at distance 0 from each other in the synchronization graph, and can therefore be considered, for distance computations, as a single node s. Now, assuming that the source clock actually shows real time, it is easy to see that for any event p,

$$\mathrm{RT}(p) \in [\mathrm{LT}(p) - d(s, p), \mathrm{LT}(p) + d(p, s)],$$

and furthermore, no better bounds can be obtained by any correct algorithm.

The general algorithm described above (maintaining the complete synchronization graph) can be used also to obtain optimal results for internal synchronization; details are omitted.

An interesting special case is where all clocks are drift free. In this case, the size of the synchronization graph remains fixed: similarly to a source node in external synchronization, all events occurring at the same processor can be mapped to a single node; parallel edges can be replaced by a single new edge whose weight is minimal among all old edges. This way one can obtain a particularly efficient distributed algorithm solving external clock synchronization, based on the distributed Bellman–Ford algorithm for distance computation.

Finally, note that the asynchronous model may also be viewed as a special case of this general theory, where an event p "happens before" an event q if and only if $d(p, q) \leq 0$.

Open Problems

One central issue in clock synchronization is faulty executions, where the real time specification is violated. Synchronization graphs detect any detectable error: views which do not have an execution that conforms with the real time specification will result in synchronization graphs with negative cycles. However, it is desirable to overcome such faults, say by removing from the synchronization graph some edges so as to break all negative-weight cycles. The natural objective in this case is to remove the least number of edges. This problem is APX-hard as it generalizes the Feedback Arc Set problem. Unfortunately, no non-trivial approximation algorithms for it are known.

Cross-References

▸ Causal Order, Logical Clocks, State Machine Replication

Recommended Reading

1. Attiya H, Herzberg A, Rajsbaum S (1996) Optimal clock synchronization under different delay assumptions. SIAM J Comput 25(2):369–389
2. Fan R, Lynch NA (2006) Gradient clock synchronization. Distrib Comput 18(4):255–266
3. Halpern JY, Megiddo N, Munshi AA (1985) Optimal precision in the presence of uncertainty. J Complex 1:170–196
4. Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21(7):558–565
5. Lamport L (1986) The mutual exclusion problem. Part I: a theory of interprocess communication. J ACM 33(2):313–326
6. Liskov B (1993) Practical uses of synchronized clocks in distributed systems. Distrib Comput 6:211–219. Invited talk at the 9th annual ACM symposium on principles of distributed computing, Quebec City, 22–24 Aug 1990
7. Lundelius J, Lynch N (1988) A new fault-tolerant algorithm for clock synchronization. Inf Comput 77:1–36
8. Mills DL (2006) Computer network time synchronization: the network time protocol. CRC, Boca Raton
9. Ostrovsky R, Patt-Shamir B (1999) Optimal and efficient clock synchronization under drifting clocks. In: Proceedings of the 18th annual symposium on principles of distributed computing, Atlanta, May 1999, pp 3–12
10. Patt-Shamir B, Rajsbaum S (1994) A theory of clock synchronization. In: Proceedings of the 26th annual ACM symposium on theory of computing, Montreal, 23–25 May 1994, pp 810–819

C

Closest String and Substring Problems

Lusheng Wang[1], Ming Li[2], and Bin Ma[2,3]
[1]Department of Computer Science, City University of Hong Kong, Hong Kong, Hong Kong
[2]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada
[3]Department of Computer Science, University of Western Ontario, London, ON, Canada

Keywords

Approximation algorithm; Fixed-parameter algorithms

Years and Authors of Summarized Original Work

2000; Li, Ma, Wang
2003; Deng, et al.
2008; Marx
2009; Ma, Sun
2011; Chen, Wang
2012; Chen, Ma, Wang

Problem Definition

The problem of finding a center string that is "close" to every given string arises and has applications in computational molecular biology [4,5,9–11,18,19] and coding theory [1,6,7].

This problem has two versions: The first problem comes from coding theory when we are looking for a code not too far away from a given set of codes.

Problem 1 (The closest string problem) Input: a set of strings $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, each of length m.

Output: the smallest d and a string s of length m which is within Hamming distance d to each $s_i \in \mathcal{S}$.

The second problem is much more elusive than the closest string problem. The problem is formulated from applications in finding conserved regions, genetic drug target identification, and genetic probes in molecular biology.

Problem 2 (The closest substring problem) Input: an integer L and a set of strings $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, each of length m.

Output: the smallest d and a string s, of length L, which is within Hamming distance d away from a length L substring t_i of s_i for $i = 1, 2, \ldots, n$.

The following results on approximation algorithms are from [12–15].

Theorem 1 *There is a polynomial time approximation scheme for the closest string problem.*

Theorem 2 *There is a polynomial time approximation scheme for the closest substring problem.*

A faster approximation algorithm for the closest string problem was given in [16].

Lots of results have been obtained in terms of parameterized complexity and fixed-parameter algorithms. In 2005, Marx showed that the closest substring problem is W[1]-hard even if both d and n are parameters [17]. Two algorithms for the closest substring problem have been developed [17] for the cases where d and n are small. The running times for the two algorithms are $f(d) \cdot m^{O(\log d)}$ and $g(d,n) \cdot n^{O(\log \log n)}$ for some functions f and g, respectively. The first fixed-parameter algorithm for closest string problem has a running time complexity $O(nd^{d+1})$ [8]. Ma and Sun designed a fixed-parameter algorithm with running time $O(nm + nd \cdot (16|\Sigma|)^d)$ for the closest string problem [16]. Extending the algorithm for the closest string problem, an $O(nL + nd \times 2^{4d}|\Sigma|^d \times m^{\lceil \log d \rceil + 1})$ time algorithm was given for the closest substring problem [16].

Since then, a series of improved algorithms have been obtained. Wang and Zhu gave an $O(nL + nd \cdot (2^{3.25}(|\Sigma| - 1))^d)$ algorithm for the closest string problem [20]. Chen and Wang gave an algorithm with running times $O(nL + nd \cdot 47.21^d)$ for protein with $|\Sigma| = 20$ and $O(nL + nd \cdot 13.92^d)$ for DNA with $|\Sigma| = 4$, respectively [2]. They also developed a software package for the (L,d) motif model. Currently the fastest fixed-parameter algorithm for the closest string problem was given by Chen, Ma, and Wang. They developed a three-string approach and the running time of the algorithm is $O(nL + nd^3 \cdot d^6.731)$ for binary strings [3].

Results for other measures with applications in computational biology can be found in [5, 9, 18, 19].

Applications

Many problems in molecular biology involve finding similar regions common to each sequence in a given set of DNA, RNA, or protein sequences. These problems find applications in locating binding sites and finding conserved regions in unaligned sequences [5, 9, 18, 19], genetic drug target identification [10], designing genetic probes [10], universal PCR primer design [5, 10], and, outside computational biology, in coding theory [1, 6, 7]. Such problems may be considered to be various generalizations of the common substring problem, allowing errors. Many measures have been proposed for finding such regions common to every given string. A popular and one of the most fundamental

measures is the Hamming distance. Moreover, two popular objective functions are used in these areas. One is the total sum of distances between the center string (common substring) and each of the given strings. The other is the maximum distance between the center string and a given string. For more details, see [10].

A More General Problem

The *distinguishing substring selection* problem has as input two sets of strings, $\mathcal{B}$ and $\mathcal{G}$. It is required to find a substring of unspecified length (denoted by L) such that it is, informally, close to a substring of every string in $\mathcal{B}$ and far away from every length L substring of strings in $\mathcal{G}$. However, we can go through all the possible length L substrings of strings in $\mathcal{G}$, and we may assume that every string in $\mathcal{G}$ has the same length L since $\mathcal{G}$ can be reconstructed to contain all substrings of length L in each of the good strings.

The problem is formally defined as follows: Given a set $\mathcal{B} = \{s_1, s_2, \ldots, s_n\}$ of n_1 (bad) strings of length at least L, and a set $\mathcal{G} = \{g_1, g_2, \ldots g_{n_2}\}$ of n_2 (good) strings of length exactly L, as well as two integers d_b and d_g ($d_b \leq d_g$), the distinguishing substring selection problem (DSSP) is to find a string s such that for each string, there exists a length L substring t_i of s_i with $d(s, t_i) \leq d_b$ and for any string $g_i \in \mathcal{G}$, $d(s, g_i) \geq d_g$. Here $d(,)$ represents the Hamming distance between two strings. If all strings in $\mathcal{B}$ are also of the same length L, the problem is called the distinguishing string problem (DSP).

The distinguishing string problem was first proposed in [10] for generic drug target design. The following results are from [4].

Theorem 3 *There is a polynomial time approximation scheme for the distinguishing substring selection problem. That is, for any constant $\epsilon > 0$, the algorithm finds a string s of length L such that for every $s_i \in \mathcal{B}$, there is a length L substring t_i of s_i with $d(t_i, s) \leq (1 + \epsilon)d_b$ and for every substring u_i of length L of every $g_i \in \mathcal{G}$, $d(u_i, s) \geq (1 - \epsilon)d_g$, if a solution to the original pair ($d_b \leq d_g$) exists. Since there are a polynomial number of such pairs (db,dg), we can exhaust all the possibilities in polynomial time to find a good approximation required by the corresponding application problems.*

Open Problems

The PTASs designed here use linear programming and randomized rounding technique to solve some cases for the problem. Thus, the running time complexity of the algorithms for both the closest string and closest substring is very high. An interesting open problem is to design more efficient PTASs for both problems.

Recommended Reading

1. Ben-Dor A, Lancia G, Perone J, Ravi R (1997) Banishing bias from consensus sequences. In: Proceedings of the 8th annual symposium on combinatorial pattern matching conference, Aarhus, pp 247–261
2. Chen Z, Wang L (2011) Fast exact algorithms for the closest string and substring problems with application to the planted (L, d)-motif model. IEEE/ACM Trans Comput Biol Bioinform 8(5):1400–1410
3. Chen Z-Z, Ma B, Wang L (2012) A three-string approach to the closest string problem. J Comput Syst Sci 78(1):164–178
4. Deng X, Li G, Li Z, Ma B, Wang L (2003) Genetic design of drugs without side-effects. SIAM J Comput 32(4):1073–1090
5. Dopazo J, Rodríguez A, Sáiz JC, Sobrino F (1993) Design of primers for PCR amplification of highly variable genomes. CABIOS 9:123–125
6. Frances M, Litman A (1997) On covering problems of codes. Theor Comput Syst 30:113–119
7. Gasieniec L, Jansson J, Lingas A (1999) Efficient approximation algorithms for the hamming center problem. In: Proceedings of the 10th ACM-SIAM symposium on discrete algorithms, Baltimore, pp 135–S906
8. Gramm J, Niedermeier R, Rossmanith P 2003 Fixed-parameter algorithms for closest string and related problems. Algorithmica 37(1):25–42
9. Hertz G, Stormo G (1995) Identification of consensus patterns in unaligned DNA and protein sequences: a large-deviation statistical basis for penalizing gaps. In: Proceedings of the 3rd international conference on bioinformatics and genome research, Tallahassee, pp 201–216

10. Lanctot K, Li M, Ma B, Wang S, Zhang L (1999) Distinguishing string selection problems. In: Proceedings of the 10th ACM-SIAM symposium on discrete algorithms, Baltimore, pp 633–642
11. Lawrence C, Reilly A (1990) An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. Proteins 7:41–51
12. Li M, Ma B, Wang L (2002) Finding similar regions in many sequences. J Comput Syst Sci 65(1):73–96
13. Li M, Ma B, Wang L (1999) Finding similar regions in many strings. In: Proceedings of the thirty-first annual ACM symposium on theory of computing, Atlanta, pp 473–482
14. Li M, Ma B, Wang L (2002) On the closest string and substring problems. J ACM 49(2):157–171
15. Ma B (2000) A polynomial time approximation scheme for the closest substring problem. In: Proceedings of the 11th annual symposium on combinatorial pattern matching, Montreal, pp 99–107
16. Ma B, Sun X (2009) More efficient algorithms for closest string and substring problems. SIAM J Comput 39(4):1432–1443
17. Marx D (2008) Closest substring problems with small distances. SIAM J Comput 38(4):1382–1410
18. Stormo G (1990) Consensus patterns in DNA. In: Doolittle RF (ed) Molecular evolution: computer analysis of protein and nucleic acid sequences. Methods Enzymol 183:211–221
19. Stormo G, Hartzell GW III (1991) Identifying protein-binding sites from unaligned DNA fragments. Proc Natl Acad Sci USA 88:5699–5703
20. Wang L, Zhu B (2009) Efficient algorithms for the closest string and distinguishing string selection problems. In: Proceedings of 3rd international workshop on frontiers in algorithms, Hefei. Lecture notes in computer science, vol 5598, pp 261–270

Closest Substring

Jens Gramm
WSI Institute of Theoretical Computer Science, Tübingen University, Tübingen, Germany

Keywords

Common approximate substring

Years and Authors of Summarized Original Work

2005; Marx

Problem Definition

CLOSEST SUBSTRING is a core problem in the field of consensus string analysis with, in particular, applications in computational biology. Its decision version is defined as follows.

CLOSEST SUBSTRING
Input: k strings $s_1, s_2, \ldots, s_k$ over alphabet Σ and non-negative integers d and L.
Question: Is there a string s of length L and, for all $i = 1, \ldots, k$, a length-L substring s_i' of s_i such that $d_H(s, s_i') \leq d$?
Here $d_H(s, s_i')$ denotes the Hamming distance between s and s_i', i.e., the number of positions in which s and s_i' differ. Following the notation used in [7], m is used to denote the average length of the input strings and n to denote the total size of the problem input.

The optimization version of CLOSEST SUBSTRING asks for the minimum value of the distance parameter d for which the input strings still allow a solution.

Key Results

The classical complexity of CLOSEST SUBSTRING is given by

Theorem 1 ([4, 5]) CLOSEST SUBSTRING *is NP-complete, and remains so for the special case of the* CLOSEST STRING *problem, where the requested solution string s has to be of same length as the input strings.* CLOSEST STRING *is NP-complete even for the further restriction to a binary alphabet.*

The following theorem gives the central statement concerning the problem's approximability:

Theorem 2 ([6]) CLOSEST SUBSTRING *(as well as* CLOSEST STRING*) admit polynomial time approximation schemes (PTAS's), where the objective function is the minimum Hamming distance d.*

In its randomized version, the PTAS cited by Theorem 2 computes, with high probability,

a solution with Hamming distance $(1+\epsilon)d_{\text{opt}}$ for an optimum value d_{opt} in $(k^2m)^{O(\log|\Sigma|/\epsilon^4)}$ running time. With additional overhead, this randomized PTAS can be derandomized. A straightforward and efficient factor-2 approximation for CLOSEST STRING is obtained by trying all length-L substrings of one of the input strings.

The following two statements address the problem's parametrized complexity, with respect to both obvious problem parameters d and k:

Theorem 3 ([3]) CLOSEST SUBSTRING *is W[1]-hard with respect to the parameter k, even for binary alphabet.*

Theorem 4 ([7]) CLOSEST SUBSTRING *is W[1]-hard with respect to the parameter d, even for binary alphabet.*

For non-binary alphabet the statement of Theorem 3 has been shown independently by Evans et al. [2]. Theorem 3 and Theorem 4 show that an exact algorithm for CLOSEST SUBSTRING with polynomial running time is unlikely for a constant value of d as well as for a constant value of k, i.e., such an algorithm does not exist unless 3-SAT can be solved in subexponential time.

Theorem 4 also allows additional insights into the problem's approximability: In the PTAS for CLOSEST SUBSTRING, the exponent of the polynomial bounding the running time depends on the approximation factor. These are not "efficient" PTAS's (EPTAS's), i.e., PTAS's with a $f(\epsilon)\cdot n^c$ running time for some function f and some constant c, and therefore are probably not useful in practice. Theorem 4 implies that most likely the PTAS with the $n^{O(1/\epsilon^4)}$ running time presented in [6] cannot be improved to an EPTAS. More precisely, there is no $f(\epsilon)\cdot n^{o(\log 1/\epsilon)}$ time PTAS for CLOSEST SUBSTRING unless 3-SAT can be solved in subexponential time. Moreover, the proof of Theorem 4 also yields.

Theorem 5 ([7]) *There are no $f(d,k)\cdot n^{o(\log d)}$ time and no $g(d,k)\cdot n^{o(\log\log k)}$ exact algorithms solving* CLOSEST SUBSTRING *for some functions f and g unless 3-SAT can be solved in subexponential time.*

For unbounded alphabet the bounds have been strengthened by showing that Closest Substring has no PTAS with running time $f(\epsilon)\cdot n^{o(1/\epsilon)}$ for any function f unless 3-SAT can be solved in subexponential time [10]. The following statements provide exact algorithms for CLOSEST SUBSTRING with small fixed values of d and k, matching the bounds given in Theorem 5:

Theorem 6 ([7]) CLOSEST SUBSTRING *can be solved in time $f(d)\cdot n^{O(\log d)}$ for some function f, where, more precisely, $f(d)=|\Sigma|^{d(\log d+2)}$.*

Theorem 7 ([7]) CLOSEST SUBSTRING *can be solved in time $g(d,k)\cdot n^{O(\log\log k)}$ for some function g, where, more precisely, $g(d,k)=(|\Sigma|d)^{O(kd)}$.*

With regard to problem parameter L, CLOSEST SUBSTRING can be trivially solved in $O(|\Sigma|^L\cdot n)$ time by trying all possible strings over alphabet Σ.

Applications

An application of CLOSEST SUBSTRING lies in the analysis of biological sequences. In motif discovery, a goal is to search "signals" common to a set of selected strings representing DNA or protein sequences. One way to represent these signals are approximately preserved substrings occurring in each of the input strings. Employing Hamming distance as a biologically meaningful distance measure results in the problem formulation of CLOSEST SUBSTRING.

For example, Sagot [9] studies motif discovery by solving CLOSEST SUBSTRING (and generalizations thereof) using suffix trees; this approach has a worst-case running time of $O(k^2m\cdot L^d\cdot|\Sigma|^d)$. In the context of motif discovery, also heuristics applicable to CLOSEST SUBSTRING were proposed, e.g., Pevzner and Sze [8] present an algorithm called WINNOWER and Buhler and Tompa [1] use a technique called random projections.

Open Problems

It is open [7] whether the $n^{O(1/\epsilon^4)}$ running time of the approximation scheme presented in [6] can be improved to $n^{O(\log 1/\epsilon)}$, matching the bound derived from Theorem 4.

Cross-References

The following problems are close relatives of CLOSEST SUBSTRING:

- ▶ Closest String and Substring Problems is the special case of CLOSEST SUBSTRING, where the requested solution string s has to be of same length as the input strings.
- Distinguishing Substring Selection is the generalization of CLOSEST SUBSTRING, where a second set of input strings and an additional integer d' are given and where the requested solution string s has – in addition to the requirements posed by CLOSEST SUBSTRING – Hamming distance at least d' with every length-L substring from the second set of strings.
- Consensus Patterns is the problem obtained by replacing, in the definition of CLOSEST SUBSTRING, the maximum of Hamming distances by the sum of Hamming distances. The resulting modified question of CONSENSUS PATTERNS is: Is there a string s of length L with

$$\sum_{i=1,\ldots,m} d_H(s, s_i') \leq d?$$

CONSENSUS PATTERNS is the special case of SUBSTRING PARSIMONY in which the phylogenetic tree provided in the definition of SUBSTRING PARSIMONY is a star phylogeny.

Recommended Reading

1. Buhler J, Tompa M (2002) Finding motifs using random projections. J Comput Biol 9(2):225–242
2. Evans PA, Smith AD, Wareham HT (2003) On the complexity of finding common approximate substrings. Theor Comput Sci 306(1–3):407–430
3. Fellows MR, Gramm J, Niedermeier R (2006) On the parameterized intractability of motif search problems. Combinatorica 26(2):141–167
4. Frances M, Litman A (1997) On covering problems of codes. Theor Comput Syst 30:113–119
5. Lanctot JK, Li M, Ma B, Wang S, Zhang L (2003) Distinguishing string search problems. Inf Comput 185:41–55
6. Li M, Ma B, Wang L (2002) On the closest string and substring problems. J ACM 49(2):157–171
7. Marx D (2005) The closest substring problem with small distances. In: Proceedings of the 46th FOCS. IEEE Press, pp 63–72
8. Pevzner PA, Sze SH (2000) Combinatorial approaches to finding subtle signals in DNA sequences. In: Proceedings of 8th ISMB. AAAI Press, pp 269–278
9. Sagot MF (1998) Spelling approximate repeated or common motifs using a suffix tree. In: Proceedings of the 3rd LATIN. LNCS, vol 1380. Springer, pp 111–127
10. Wang J, Huang M, Cheng J (2007) A lower bound on approximation algorithms for the closest substring problem. In: Proceedings of the COCOA 2007. LNCS, vol 4616, pp 291–300

Clustered Graph Drawing

Fabrizio Frati
School of Information Technologies, The University of Sydney, Sydney, NSW, Australia
Engineering Department, Roma Tre University, Rome, Italy

Keywords

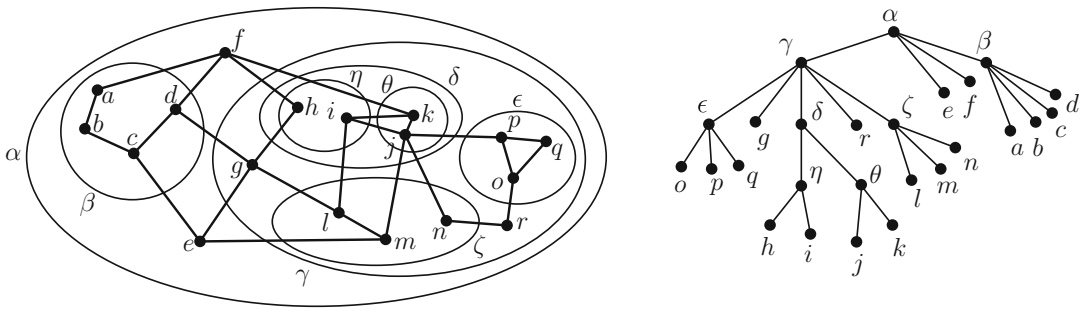
Clustered graph; Convex drawings; Graph drawing; Planarity testing; Straight-line drawings

Years and Authors of Summarized Original Work

1995; Feng, Cohen, Eades

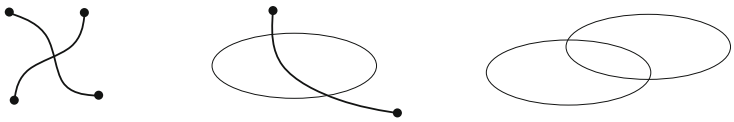
Problem Definition

A *clustered graph* $C(G, T)$ consists of a graph G, called *underlying graph*, and of a rooted tree T, called *inclusion tree*. The leaves of T are the vertices of G; each internal node μ of T



Clustered Graph Drawing, Fig. 1 A clustered graph $C(G, T)$ (*left*) and its inclusion tree (*right*)

Clustered Graph Drawing, Fig. 2 Types of crossings in a drawing of a clustered graph



represents a *cluster*, that is, the set of vertices of G that are the leaves of the subtree of T rooted at μ. Figure 1 shows a clustered graph and its inclusion tree.

Clustered graphs are widely used in applications where it is needed at the same time to represent relationships between entities and to group entities with semantic affinities. For example, in the Internet network, the routers and the links between them are the vertices and edges of a graph, respectively; geographically close routers are grouped into areas that are hence associated with clusters of vertices. In turn, areas are grouped into autonomous systems that are hence associated with clusters of vertices.

Visualizing clustered graphs is a difficult problem, due to the simultaneous need for a readable drawing of the underlying structure and of the clustering relationship. As for the visualization of graphs, the most important aesthetic criterion for the readability of a drawing of a clustered graph is the *planarity*, whose definition needs a refinement in the context of clustered graphs, in order to deal with the clustering structure.

In a *drawing* of a clustered graph $C(G, T)$, vertices and edges of G are drawn as points and open curves, respectively, and each cluster μ is represented by a simple closed region R_μ containing all and only the vertices of μ. A drawing of C can have three types of crossings. *Edge-edge crossings* are crossings between edges of G (see Fig. 2, left). Consider an edge e of G and a cluster μ in T. If e intersects the boundary of R_μ more than once, we have an *edge-region crossing* (see Fig. 2, middle). Finally, consider two clusters μ and ν in T; if the boundary of R_μ intersects the boundary of R_ν, we have a *region-region crossing* (see Fig. 2, right). A drawing of a clustered graph is *c-planar* (short for *clustered planar*) if it does not have any edge-edge, edge-region, or region-region crossing. A clustered graph is *c-planar* if it admits a c-planar drawing. A drawing of a clustered graph is *straight line* if each edge is represented by a straight-line segment; also, it is *convex* if each cluster is represented by a convex region.

The notion of c-planarity was first introduced by Feng, Cohen, and Eades in 1995 [10, 11]. The graph drawing community has subsequently adopted this definition as a standard, and the topological and geometric properties of c-planar drawings of clustered graphs have been investigated in tens of papers. The two main questions raised by Feng, Cohen, and Eades were the following.

Problem 1 (C-Planarity Testing)

QUESTION: What's the time complexity of testing the c-planarity of a clustered graph?

Problem 2 (Straight-Line Convex C-Planar Drawability)

QUESTION: Does every c-planar-clustered graph admit a straight-line convex c-planar drawing?

Key Results

Almost 20 years after the publication of the seminal papers by Feng et al. [10, 11], a solution for Problem 1 remains an elusive goal, arguably the most intriguing and well-studied algorithmic problem in the graph drawing research area (see, e.g., [3, 4, 7, 12, 13, 15–17]).

Polynomial-time algorithms have been presented to test the c-planarity of a large number of classes of clustered graphs. A particular attention has been devoted to *c-connected* clustered graphs that are clustered graphs $C(G, T)$ such that each cluster $\mu \in T$ induces a connected component G_μ of G. The following theorem reveals the importance of c-connected clustered graphs.

Theorem 1 (Feng, Cohen, and Eades [11]) *A clustered graph is c-planar if and only if it is a subgraph of a c-planar c-connected clustered graph.*

Feng, Cohen, and Eades provided in [11] a nice and simple quadratic-time testing algorithm, which is described in the following.

Theorem 2 (Feng, Cohen, and Eades [11]) *The c-planarity of an n-vertex c-connected clustered graph can be tested in $O(n^2)$ time.*

The starting point of Feng et al. result is a characterization of c-planar drawings.

Theorem 3 (Feng, Cohen, and Eades [11]) *A drawing of a c-connected clustered graph $C(G, T)$ is c-planar if and only if it is planar, and, for each cluster μ, all the vertices and edges of $G - G_\mu$ are in the outer face of the drawing of G_μ.*

The algorithm of Feng et al. [11] performs a bottom-up traversal of T.

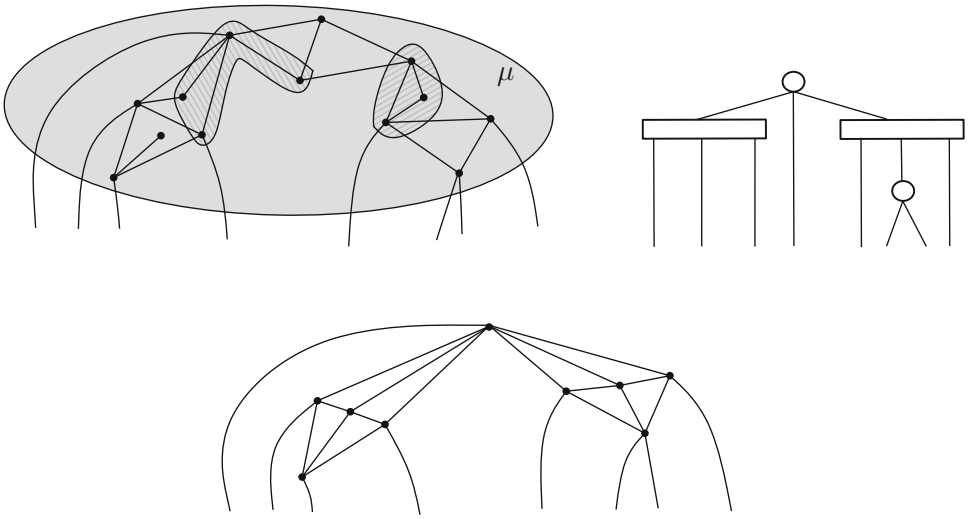
When a node $\mu \in T$ is considered, the algorithm tests whether a drawing of G_μ exists such that (P1) for each descendant ν of μ, all the vertices and edges of $G_\mu - G_\nu$ are in the outer face of the drawing of G_ν in Γ_μ and (P2) all the vertices of G_μ having neighbors in $G - G_\mu$ are incident to the outer face of G_μ in Γ_μ. Feng et al. show how a PQ-tree P_μ [2] can be used to efficiently represent all the (possibly exponentially many) orderings in which the edges incident to μ can cross the boundary of R_μ in any planar drawing Γ_μ of G_μ satisfying properties P1 and P2 (see Fig. 3, left and middle).

PQ-tree P_μ can be easily computed for each leaf $\mu \in T$. Consider an internal node $\mu \in T$ and assume that PQ-trees $P_{\mu_1}, \ldots, P_{\mu_k}$ have been associated to the children $\mu_1, \ldots, \mu_k$ of μ. *Representative graphs* $H_{\mu_1}, \ldots, H_{\mu_k}$ are constructed from $P_{\mu_1}, \ldots, P_{\mu_k}$; the embeddings of H_{μ_i} are in bijection with the embeddings of G_{μ_i} satisfying properties P1 and P2 (see Fig. 3, left and right). Then, a graph G'_μ is constructed composed of $H_{\mu_1}, \ldots, H_{\mu_k}$, of a dummy vertex v_μ, and of length-2 paths connecting v_μ with every vertex of $H_{\mu_1}, \ldots, H_{\mu_k}$ that has a neighbor in $G - G_\mu$. Feng et al. argue that the embeddings of G_μ satisfying properties P1 and P2 are in bijection with the embeddings of G'_μ in which v_μ is incident to the outer face. Hence, a planarity testing for G'_μ is performed. This allows to determine P_μ, thus allowing the visit of T to go on.

If no planarity test fails, the algorithm completes the visit of T. Top-down traversing T and fixing an embedding for the PQ-tree associated to each node of T determines a c-planar drawing of $C(G, T)$.

Involved linear-time algorithms to test the c-planarity of c-connected clustered graphs are known nowadays [5, 6]. The algorithm in [5] relies on a structural characterization of the c-planarity of a c-connected clustered graph $C(G, T)$ based on the decomposition of G into triconnected components. The characterization allows one to test in linear time the c-planarity of $C(G, T)$ via a bottom-up visit of the SPQR-tree [8] of G, which is a data structure efficiently representing the planar embeddings of G.

Problem 1 is fundamental for the graph drawing research area. However, no less importance has to be attributed to the task of designing



Clustered Graph Drawing, Fig. 3 *Left*: Graph G_μ and the edges incident to μ. *Middle*: The PQ-tree representing the possible orderings in which the edges incident to μ can cross the boundary of R_μ in a planar drawing of G_μ satisfying properties P1 and P2. *Right*: The representative graph H_μ for G_μ

algorithms for constructing *geometric representations* of clustered graphs. The milestones in this research direction have been established by Feng et al., who provided in [10] a full answer to Problem 2.

Theorem 4 (Feng, Cohen, and Eades [10]) *Every c-planar clustered graph admits a straight-line convex c-planar drawing.*

The proof of Theorem 4 relies on a positive answer for the following question: Does every *planar hierarchical graph* admit a *planar straight-line hierarchical drawing*? A *hierarchical graph* is a graph with an assignment of its vertices to k layers $l_1, \ldots, l_k$. A hierarchical drawing maps each vertex assigned to layer l_i to a point on the horizontal line $y = i$ and each edge to a y-monotone curve between the corresponding endpoints. A hierarchical graph is *planar* if it admits a planar hierarchical drawing. Feng et al. [10] showed an algorithm to construct a planar straight-line hierarchical drawing of any planar hierarchical graph H. Their algorithm splits H into some subgraphs, inductively constructs planar straight-line hierarchical drawings of such subgraphs, and glues these drawings together to obtain a planar straight-line hierarchical drawing of H.

Feng et al. also showed how the result on hierarchical graphs leads to a proof of Theorem 4, namely:

1. Starting from any c-planar clustered graph $C(G, T)$, construct a hierarchical graph H by assigning the vertices of G to n layers, in the same order as in an st-numbering of G in which vertices of the same cluster are numbered consecutively.
2. Construct a planar straight-line hierarchical drawing Γ_H of H.
3. Construct a straight-line convex c-planar drawing of $C(G, T)$ starting from Γ_H by drawing each cluster as a convex region slightly surrounding the convex hull of its vertices.

Angelini, Frati, and Kaufmann [1] recently strengthened Theorem 4 by proving that every c-planar clustered graph admits a straight-line c-planar drawing in which each cluster is represented by a scaled copy of an arbitrary convex shape.

Hong and Nagamochi [14] studied straight-line convex c-planar drawings in which the faces of the underlying graph are delimited by convex polygons. They proved that a c-connected

clustered graph admits such a drawing if and only if it is c-planar, completely connected (i.e., for every cluster μ, both G_μ and $G - G_\mu$ are connected), and internally triconnected (i.e., for every separation pair $\{u, v\}$, vertices u and v are incident to the outer face of G and each connected component of $G - \{u, v\}$ contains vertices incident to the outer face of G).

The drawings constructed by the algorithm in [10] use real coordinates. Hence, when displaying these drawings on a screen with a finite resolution rule, exponential area might be required for the visualization. This drawback is however unavoidable. Namely, Feng et al. proved in [10] that there exist clustered graphs requiring exponential area in *any* straight-line convex c-planar drawing with a finite resolution rule. This harshly differentiates c-planar clustered graphs from planar graphs, as straight-line convex planar drawings of planar graphs can be constructed in polynomial area.

Theorem 5 (Feng, Cohen, and Eades [10]) *There exist n-vertex c-planar clustered graphs requiring $2^{\Omega(n)}$ area in any straight-line convex c-planar drawing.*

The proof of Theorem 5 adapts techniques introduced by Di Battista et al. [9] to prove area lower bounds for *straight-line upward planar drawings* of *directed graphs*.

Open Problems

After almost 20 years since it was first posed by Feng, Cohen, and Eades, Problem 1 still represents a terrific challenge for researchers working in graph drawing.

A key result of Feng, Cohen, and Eades [11] shows that testing the c-planarity of a clustered graph $C(G, T)$ is a polynomial-time solvable problem if $C(G, T)$ is c-connected – see Theorem 2. Moreover, a clustered graph is c-planar if and only if it is a subgraph of a c-planar c-connected clustered graph – see Theorem 1. Hence, the core of testing the c-planarity of a non-c-connected clustered graph $C(G, T)$ is an augmentation problem, asking whether $C(G, T)$ can be augmented to a c-connected c-planar clustered graph $C'(G', T)$ by inserting edges in G.

This augmentation problem seems far from being solved. A particular attention [3,4,7,15,16] has been devoted to the case in which a planar embedding for G is prescribed as part of the input. In this case, edges might only be inserted inside faces of G, in order to guarantee the planarity of G'. Thus, the problem becomes equivalent to the one selecting a set S of edges into a set M of topological embedded multigraphs, where each cluster μ defines a multigraph M_μ in M consisting of all the edges that can be inserted inside faces of G in order to connect distinct connected components of G_μ. Then the edges in S are those that are selected to augment G to G' – hence no two edges in S are allowed to cross. Even in this prescribed-embedding version, only partial results are known. For example, polynomial-time algorithms to test the c-planarity of $C(G, T)$ are known if the faces of G have at most five incident vertices [7], or if each cluster induces at most two connected components [15], or if each cluster has at most two incident vertices on each face of G [3].

Cross-References

- Convex Graph Drawing
- Upward Graph Drawing

Recommended Reading

1. Angelini P, Frati F, Kaufmann M (2011) Straight-line rectangular drawings of clustered graphs. Discret Comput Geom 45(1):88–140
2. Booth KS, Lueker GS (1976) Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J Comput Syst Sci 13(3):335–379
3. Chimani M, Di Battista G, Frati F, Klein K (2014) Advances on testing c-planarity of embedded flat clustered graphs. In: Graph drawing (GD '14), Würzburg, pp 416–427
4. Cortese PF, Di Battista G, Patrignani M, Pizzonia M (2005) Clustering cycles into cycles of

clusters. J Graph Algorithms Appl 9(3):391–413. doi:10.7155/jgaa.00115
5. Cortese PF, Di Battista G, Frati F, Patrignani M, Pizzonia M (2008) C-planarity of c-connected clustered graphs. J Graph Algorithms Appl 12(2):225–262
6. Dahlhaus E (1998) A linear time algorithm to recognize clustered graphs and its parallelization. In: Lucchesi CL, Moura AV (eds) Latin American theoretical informatics (LATIN '98), Campinas. LNCS, vol 1380. Springer, pp 239–248
7. Di Battista G, Frati F (2009) Efficient c-planarity testing for embedded flat clustered graphs with small faces. J Graph Algorithms Appl 13(3):349–378. Special issue from GD '07
8. Di Battista G, Tamassia R (1996) On-line planarity testing. SIAM J Comput 25:956–997
9. Di Battista G, Tamassia R, Tollis IG (1992) Area requirement and symmetry display of planar upward drawings. Discret Comput Geom 7: 381–401
10. Feng Q, Cohen RF, Eades P (1995) How to draw a planar clustered graph. In: Du D, Li M (eds) Computing and combinatorics conference (COCOON '95), Xi'an. LNCS, vol 959. Springer, pp 21–30
11. Feng Q, Cohen RF, Eades P (1995) Planarity for clustered graphs. In: Spirakis P (ed) European symposium on algorithms (ESA '95), Corfu. LNCS, vol 979. Springer, pp 213–226
12. Goodrich MT, Lueker GS, Sun JZ (2006) C-planarity of extrovert clustered graphs. In: Healy P, Nikolov N (eds) International symposium on graph drawing (GD '05), Limerick. LNCS, vol 3843. Springer, pp 211–222
13. Gutwenger C, Jünger M, Leipert S, Mutzel P, Percan M, Weiskircher R (2002) Advances in c-planarity testing of clustered graphs. In: Goodrich MT, Kobourov SG (eds) International symposium on graph drawing (GD '02), Irvine. LNCS, vol 2528. Springer, pp 220–235
14. Hong SH, Nagamochi H (2010) Convex drawings of hierarchical planar graphs and clustered planar graphs. J Discret Algorithms 8(3): 282–295
15. Jelínek V, Jelínková E, Kratochvíl J, Lidický B (2009) Clustered planarity: embedded clustered graphs with two-component clusters. In: Tollis IG, Patrignani M (eds) Graph drawing (GD '08), Heraklion. LNCS, vol 5417, pp 121–132. doi:10.1007/978-3-642-00219-9_13
16. Jelínková E, Kára J, Kratochvíl J, Pergel M, Suchý O, Vyskocil T (2009) Clustered planarity: small clusters in cycles and Eulerian graphs. J Graph Algorithms Appl 13(3):379–422
17. Schaefer M (2013) Toward a theory of planarity: Hanani-Tutte and planarity variants. J Graph Algorithms Appl 17(4):367–440

Clustering Under Stability Assumptions

Pranjal Awasthi
Department of Computer Science, Princeton University, Princeton, NJ, USA
Department of Electrical Engineering, Indian Institute of Technology Madras, Chennai, Tamilnadu, India

Keywords

Approximation stability; Clustering; k-median; k-means; Non-worst-case analysis; Separability

Years and Authors of Summarized Original Work

2006; Ostrovsky, Rabani, Schulman, Swami
2009; Balcan, Blum, Gupta
2010; Bilu, Linial
2010; Awasthi, Blum, Sheffet
2010; Kumar, Kannan
2011; Awasthi, Blum, Sheffet
2012; Balcan, Liang

Problem Definition

The problem of clustering consists of partitioning a set of objects such as images, text documents, etc. into groups of related items. The information available to the clustering algorithm consists of pairwise similarity information between objects. One of the most popular approaches to clustering is to map the objects into data points in a metric space, define an objective function over the data points, and find a partitioning which achieves the optimal solution, or an approximately optimal solution to the given objective function. In this entry, we will focus on two of the most widely studied objective functions for clustering: the k-median objective and the k-means objective.

In k-median clustering, the input is a set P of n points in a metric space (X, d), where $d(\cdot)$ is

the distance function. The objective is to find k center points $c_1, c_2, \cdots c_k$. The clustering is then formed by assigning each data point to the closest center point. If a point x is assigned to center $c(x)$, then the cost incurred is $d(x, c(x))$. The goal is to find center points and a partitioning of the data so as to minimize the total cost $\Phi = \min_{c_1,c_2,\ldots c_k} \sum_x \min_i d(x, c_i)$. This objective is closely related to the well-studied facility location problem [1, 9] in theoretical computer science.

Similarly, for the k-means objective, the goal is to find k center points. However, the cost incurred by a point x is the distance squared to its center. Hence, the goal is to minimize $\Phi = \min_{c_1,c_2,\ldots c_k} \sum_x \min_i d^2(x, c_i)$. A special case of the k-means objective which is of particular interest is the Euclidean k-means problem where the points are in $\Re^m$ and the distance function is the squared Euclidean distance. Again, the goal is to choose k center points and assign each point to the closest center while minimizing the total cost. However, unlike k-median and k-means in metric spaces, the center points do not necessarily have to belong to the data set P and can be arbitrarily chosen from $\Re^m$. Unfortunately, optimizing both these objectives turns out to be NP-hard. Hence, a lot of the work in the theoretical computer science community focuses on designing good approximation algorithms for these problems[1, 8–10, 12] with formal guarantees on worst-case instances.

However, in most practical scenarios, the clustering instances which one encounters are not worst case but instead have additional structure/stability associated with them. In such cases, it is natural to ask if one can abstract out this structure in the form of a stability notion, formally study it, and exploit this additional structure in order to obtain optimal or nearly optimal solutions and bypass NP-hardness which only applies to worst-case instances. This modern take on clustering research has, in recent years, produced new insights and deeper understanding of what we know about clustering. In this entry, we will survey some key results on clustering under stability assumptions.

Key Results

ϵ-separability:

This notion of stability was proposed by Ostrovsky et al. [15]. The motivation comes from the fact that in practice, when solving a clustering instance, one typically has to decide how many clusters to partition the data into, i.e., the value of k. If the k-means objective is the underlying criteria being used to judge the quality of a clustering, and the optimal $(k-1)$-means clustering is comparable to the optimal k-means clustering; then, one can in principle also use $(k-1)$ clusters to describe the data set. Hence, the particular clustering instance is not well behaved or not stable. In fact this particular method is a very popular heuristic to find out the number of hidden clusters in the data set suggesting that real-world instances have this property.

Definition 1 (ϵ-Separability) Given an instance of Euclidean k-means clustering, let $\mathrm{OPT}(k)$ denote the cost of the optimal k-means solution. We can also decompose $\mathrm{OPT}(k)$ as $\mathrm{OPT} = \sum_{i=1}^{k} \mathrm{OPT}_i$, where OPT_i denotes the 1-means cost of cluster C_i, i.e., $\sum_{x \in C_i} d(x, c_i)^2$. Such an instance is called *ϵ-separable* if it satisfies $\mathrm{OPT}(k-1) > \frac{1}{\epsilon^2}\mathrm{OPT}(k)$.

It was shown by Ostrovsky et al. [15] that one can design much better approximation algorithms for such instances. The algorithm is based on over sampling $O(k)$ candidate centers using a distance weighted sampling scheme, followed by a greedy deletion strategy to reduce the k centers without incurring too much increase in the k-means cost.

Theorem 1 ([15]) *There is a polynomial time algorithm which given any ϵ-separable Euclidean k-means instance, outputs a clustering of cost at most $\frac{\mathrm{OPT}}{1-\rho}$ with probability $1 - O((\rho)^{1/4})$ where $\rho = \Theta(\epsilon^2)$.*

$(1+\alpha, \epsilon)$-Approximation-Stability:

Balcan et al. [5] introduced and analyzed a class of approximation stable instances for which one can find near optimal clusterings in polynomial time. The motivation comes from the fact that for many problems of interest to machine learning, there is an unknown underlying correct "target"

clustering. In such cases, the implicit hope when pursuing an objective-based clustering approach (k-means or k-median) is that approximately optimizing the objective function will in fact produce a clustering of low clustering error, i.e., a clustering that is point wise close to the target clustering. Balcan et al. showed that by making this implicit assumption explicit, one can efficiently compute a low-error clustering even in cases when the approximation problem of the objective function is NP-hard!

Definition 2 (**$(1+\alpha,\epsilon)$-approximation-stability**) Let P be a set of n points residing in a metric space (M,d). Given an objective function Φ (such as k-median, k-means), we say that instance (M,P) *satisfies* $(1+\alpha,\epsilon)$-*approximation-stability for* Φ if all clusterings $\mathcal{C}$ with $\Phi(\mathcal{C}) \leq (1+\alpha)\cdot \mathrm{OPT}(k)$ are point-wise ϵ-close to the target clustering $\mathcal{T}$ for (M,P).

Here, the term "target" clustering refers to the ground truth clustering of P which one is trying to approximate. The distance between any two k clusterings $\mathcal{C}$ and $\mathcal{C}^*$ of n points is measured as $\mathrm{dist}(\mathcal{C},\mathcal{C}^*) = \min_{\sigma\in S_k}\frac{1}{n}\sum_{i=1}^{k}|C_i \setminus C^*_{\sigma(i)}|$. Interestingly, this approximation stability condition implies a lot of structure about the problem instance which could be exploited algorithmically. For example, one can show the following:

Theorem 2 (**[5]**) *If the given instance* (M,P) *satisfies* $(1+\alpha,\epsilon)$*-approximation-stability for the* k*-median or the* k*-means objective, then we can efficiently produce a clustering that is* $O(\epsilon+\epsilon/\alpha)$*-close to the target clustering* $\mathcal{T}$.

As mentioned above, this theorem is valid even for values of α for which getting a $(1+\alpha)$-approximation to k-median and k-means is NP-hard! The algorithm first creates a graph over data points by connecting points which are within a certain distance threshold. The next step involves iteratively peeling off connected components in the graph and simultaneously de-noising the instance.

Related Notions

The notion of ϵ-separability and $(1+\alpha,\epsilon)$-approximation-stability are related to each other. For example, Theorem 5.1 in [15] shows that ϵ-separability implies that any near-optimal solution to k-means is $O(\epsilon^2)$-close to the k-means optimal clustering. However, the converse is not necessarily the case; an instance could satisfy approximation-stability without being ϵ-separated. In [6], Balcan et al. present a specific example of points in Euclidean space with $\alpha = 1$. In fact, when k is much larger than $1/\epsilon$, the difference between the two properties can be more substantial.

The notion of separability and approximation stability was generalized in [2] where the authors study a notion of stability called α-*weak deletion* stability. A clustering instance is stable under this notion if in the optimal clustering merging any two clusters into one increases the cost by a multiplicative factor of $(1+\alpha)$. This definition captures both ϵ-separability and approximation stability in the case of large cluster sizes. Remarkably, [2] show that for such instances of k-median and Euclidean k-means, one can design a $(1+\epsilon)$ approximation algorithm for any constant $\epsilon > 0$. This leads to immediate improvements over the works of [5] (for the case of large clusters) and of [15]. However, the run time of the resulting algorithm depends polynomially in n and k and exponentially in the parameters $1/\alpha$ and $1/\epsilon$, so the simpler algorithms of [2] and [5] are more suitable for scenarios where one expects the stronger properties to hold.

Kumar and Kannan [11] study a separation condition motivated by the k-means objective and the kind of instances produced by Gaussian and related mixture models. They consider the setting of points in Euclidean space and show that if the projection of any data point onto the line joining the mean of its cluster in the target clustering to the mean of any other cluster of the target is $\Omega(k)$ standard deviations closer to its own mean than the other mean, then they can recover the target clusters in polynomial time. This condition was further analyzed and reduced by work of [3].

Bilu and Linial [7] study clustering instances which are perturbation resilient. An instance is c-perturbation resilient if it has the property that the optimal solution to the objective remains optimal even after bounded perturbations (up to factor c) to the input weight matrix. They give an algorithm for maxcut (which can be viewed

as a 2-clustering problem) under the assumption that the optimal solution is stable to (roughly) $O(n^{2/3})$-factor multiplicative perturbations to the edge weights. This was improved by [14]. Awasthi et al. [3] study perturbation resilience for center-based clustering objectives such as k-median and k-means and give an algorithm that finds the optimal solution when the input is stable to only factor-3 perturbations. This factor is improved to $1 + \sqrt{2}$ by [4], who also design algorithms under a relaxed (c, ϵ)-stability to perturbation condition in which the optimal solution need not be identical on the c-perturbed instance, but may change on an ϵ fraction of the points (in this case, the algorithms require $c = 2 + \sqrt{3}$).

For the k-median objective, (c, ϵ)-approximation-stability with respect to $\mathcal{C}^*$ implies (c, ϵ)-stability to perturbations because an optimal solution in a c-perturbed instance is guaranteed to be a c-approximation on the original instance. Similarly, for k-means, (c, ϵ)-stability to perturbations is implied by (c^2, ϵ)-approximation-stability. However, as noted above, the values of c known to lead to efficient clustering in the case of stability to perturbations are larger than for approximation-stability, where any constant $c > 1$ suffices.

Open Problems

The algorithm proposed in [15] for ϵ-separability is a variant of the popular Lloyd's heuristic for k-means [13]. Hence, the result can also be viewed as a characterization of when such heuristics work in practice. It would be interesting to establish weaker sufficient conditions for such heuristics. For instance, is it possible that weak-deletion stability is sufficient for a version of the Lloyd's heuristic to converge to the optimal clustering? Another open direction concerns the notion of perturbation resilience. Can one reduce the perturbation factor c needed for efficient clustering? Alternatively, if one cannot find the *optimal* clustering for small values of c, can one still find a near-optimal clustering, of approximation ratio better than what is possible on worst-case instances?

In a different direction, one can also consider relaxations of the perturbation-resilience condition. For example, Balcan et al. [4] also consider instances that are "mostly resilient" to c-perturbations; under any c-perturbation of the underlying metric, no more than an ϵ-fraction of the points gets mislabeled under the optimal solution. For sufficiently large constant c and sufficiently small constant ϵ, they present algorithms that get good approximations to the objective under this condition. A different kind of relaxation would be to consider a notion of *resilience to perturbations on average*: a clustering instance whose optimal clustering is likely not to change, assuming the perturbation is *random* from a suitable distribution. Can this weaker notion be used to still achieve positive guarantees?

Finally, the notion of stability can also shed light on practically interesting instances of many other important problems. Can stability assumptions, preferably ones of a mild nature, allow us to bypass NP-hardness results of other problems? One particularly intriguing direction is the problem of Sparsest-Cut, for which no PTAS or constant-approximation algorithm is known, yet a powerful heuristics based on spectral techniques work remarkably well in practice.

Recommended Reading

1. Arya V, Garg N, Khandekar R, Meyerson A, Munagala K, Pandit V (2004) Local search heuristics for k-median and facility location problems. SIAM J Comput 33(3):544–562
2. Awasthi P, Blum A, Sheffet O (2010) Stability yields a PTAS for k-median and k-means clustering. In: Proceedings of the 2010 IEEE 51st annual symposium on foundations of computer science, Las Vegas
3. Awasthi P, Blum A, Sheffet O (2012) Center-based clustering under perturbation stability. Inf Process Lett 112(1–2):49–54
4. Balcan M-F, Liang Y (2012) Clustering under perturbation resilience. In: Proceedings of the 39th international colloquium on automata, languages and programming, Warwick
5. Balcan M-F, Blum A, Gupta A (2009) Approximate clustering without the approximation. In: Proceedings of the ACM-SIAM symposium on discrete algorithms, New York
6. Balcan M-F, Blum A, Gupta A (2013) Clustering under approximation stability. J ACM 60:1–34

7. Bilu Y, Linial N (2010) Are stable instances easy? In: Proceedings of the first symposium on innovations in computer science, Beijing
8. Charikar M, Guha S, Tardos E, Shmoy DB (1999) A constant-factor approximation algorithm for the k-median problem. In: Proceedings of the thirty-first annual ACM symposium on theory of computing, Atlanta
9. Jain K, Mahdian M, Saberi A (2002) A new greedy approach for facility location problems. In: Proceedings of the 34th annual ACM symposium on theory of computing, Montreal
10. Kanungo T, Mount DM, Netanyahu NS, Piatko CD, Silverman R, Wu AY (2002) A local search approximation algorithm for k-means clustering. In: Proceedings of the eighteenth annual symposium on computational geometry. ACM, New York
11. Kumar A, Kannan R (2010) Clustering with spectral norm and the k-means algorithm. In: Proceedings of the 51st annual IEEE symposium on foundations of computer science, Las Vegas
12. Kumar A, Sabharwal Y, Sen S (2004) A simple linear time $(1+\epsilon)$-approximation algorithm for k-means clustering in any dimensions. In: Proceedings of the 45th annual IEEE symposium on foundations of computer science, Washington, DC
13. Lloyd SP (1982) Least squares quantization in PCM. IEEE Trans Inf Theory 28(2):129–137
14. Makarychev K, Makarychev Y, Vijayaraghavan A (2014) Bilu-linial stable instances of max cut and minimum multiway cut. In: SODA, Portland
15. Ostrovsky R, Rabani Y, Schulman L, Swamy C (2006) The effectiveness of lloyd-type methods for the k-means problem. In: Proceedings of the 47th annual IEEE symposium on foundations of computer science, Berkeley

Color Coding

Noga Alon[1], Raphael Yuster[2], and Uri Zwick[3]
[1]Department of Mathematics and Computer Science, Tel-Aviv University, Tel-Aviv, Israel
[2]Department of Mathematics, University of Haifa, Haifa, Israel
[3]Department of Mathematics and Computer Science, Tel-Aviv University, Tel-Aviv, Israel

Keywords

Finding small subgraphs within large graphs

Years and Authors of Summarized Original Work

1995; Alon, Yuster, Zwick

Problem Definition

Color coding [2] is a novel method used for solving, in polynomial time, various subcases of the generally NP-Hard *subgraph isomorphism* problem. The input for the subgraph isomorphism problem is an ordered pair of (possibly directed) graphs (G, H). The output is either a mapping showing that H is isomorphic to a (possibly induced) subgraph of G, or **false** if no such subgraph exists. The subgraph isomorphism problem includes, as special cases, the HAMILTON-PATH, CLIQUE, and INDEPENDENT SET problems, as well as many others. The problem is also interesting when H is *fixed*. The goal, in this case, is to design algorithms whose running times are significantly better than the running time of the naïve algorithm.

Method Description

The color coding method is a randomized method. The vertices of the graph $G = (V, E)$ in which a subgraph isomorphic to $H = (V_H, E_H)$ is sought are randomly colored by $k = |V_H|$ colors. If $|V_H| = O(\log |V|)$, then with a small probability, but only polynomially small (i.e., one over a polynomial), all the vertices of a subgraph of G which is isomorphic to H, if there is such a subgraph, will be colored by distinct colors. Such a subgraph is called *color coded*. The color coding method exploits the fact that, in many cases, it is easier to detect color coded subgraphs than uncolored ones.

Perhaps the simplest interesting subcases of the subgraph isomorphism problem are the following: Given a directed or undirected graph $G = (V, E)$ and a number k, does G contain a simple (directed) path of length k? Does G contain a simple (directed) cycle of length *exactly* k? The following describes a $2^{O(k)} \cdot |E|$

time algorithm that receives as input the graph $G = (V, E)$, a coloring $c: V \to \{1, \ldots, k\}$ and a vertex $s \in V$, and finds a colorful path of length $k-1$ that starts at s, if one exists. To find a colorful path of length $k-1$ in G that starts somewhere, just add a new vertex s' to V, color it with a new color 0 and connect it with edges to all the vertices of V. Now look for a colorful path of length k that starts at s'.

A colorful path of length $k-1$ that starts at some specified vertex s is found using a dynamic programming approach. Suppose one is already given, for each vertex $v \in V$, the possible sets of colors on colorful paths of length i that connect s and v. Note that there is no need to record all colorful paths connecting s and v. Instead, record the color sets appearing on such paths. For each vertex v there is a collection of at most $\binom{k}{i}$ color sets. Now, inspect every subset C that belongs to the collection of v, and every edge $(v, u) \in E$. If $c(u) \notin C$, add the set $C \cup \{c(u)\}$ to the collection of u that corresponds to colorful paths of length $i+1$. The graph G contains a colorful path of length $k-1$ with respect to the coloring c if and only if the final collection, that corresponding to paths of length $k-1$, of at least one vertex is non-empty. The number of operations performed by the algorithm outlined is at most $O(\sum_{i=0}^{k} i\binom{k}{i} \cdot |E|)$ which is clearly $O(k2^k \cdot |E|)$.

Derandomization

The randomized algorithms obtained using the color coding method are derandomized with only a small loss in efficiency. All that is needed to derandomize them is a family of colorings of $G = (V, E)$ so that every subset of k vertices of G is assigned distinct colors by at least one of these colorings. Such a family is also called a family of *perfect hash functions* from $\{1, 2, \ldots, |V|\}$ to $\{1, 2, \ldots, k\}$. Such a family is explicitly constructed by combining the methods of [1, 9, 12, 16]. For a derandomization technique yielding a constant factor improvement see [5].

Key Results

Lemma 1 *Let $G = (V, E)$ be a directed or undirected graph and let $c: V \to \{1, \ldots, k\}$ be a coloring of its vertices with k colors. A colorful path of length $k-1$ in G, if one exists, can be found in $2^{O(k)} \cdot |E|$ worst-case time.*

Lemma 2 *Let $G = (V, E)$ be a directed or undirected graph and let $c: V \to \{1, \ldots, k\}$ be a coloring of its vertices with k colors. All pairs of vertices connected by colorful paths of length $k-1$ in G can be found in either $2^{O(k)} \cdot |V||E|$ or $2^{O(k)} \cdot |V|^{\omega}$ worst-case time (here $\omega < 2.376$ denotes the matrix multiplication exponent).*

Using the above lemmata the following results are obtained.

Theorem 3 *A simple directed or undirected path of length $k-1$ in a (directed or undirected) graph $G = (V, E)$ that contains such a path can be found in $2^{O(k)} \cdot |V|$ expected time in the undirected case and in $2^{O(k)} \cdot |E|$ expected time in the directed case.*

Theorem 4 *A simple directed or undirected cycle of size k in a (directed or undirected) graph $G = (V, E)$ that contains such a cycle can be found in either $2^{O(k)} \cdot |V||E|$ or $2^{O(k)} \cdot |V|^{\omega}$ expected time.*

A cycle of length k in minor-closed families of graphs can be found, using color coding, even faster (for planar graphs, a slightly faster algorithm appears in [6]).

Theorem 5 *Let* C *be a non-trivial minor-closed family of graphs and let $k \geq 3$ be a fixed integer. Then, there exists a randomized algorithm that given a graph$G = (V, E)$ from* C*, finds a C_k (a simple cycle of size k) in G, if one exists, in $O(|V|)$ expected time.*

As mentioned above, all these theorems can be derandomized at the price of a $\log |V|$ factor. The algorithms are also easily to parallelize.

Applications

The initial goal was to obtain efficient algorithms for finding simple paths and cycles in graphs. The color coding method turned out, however, to have a much wider range of applicability. The linear time (i.e., $2^{O(k)} \cdot |E|$ for directed graphs and $2^{O(k)} \cdot |V|$ for undirected graphs) bounds for simple paths apply in fact to any *forest* on k vertices. The $2^{O(k)} \cdot |V|^{\omega}$ bound for simple cycles applies in fact to any *series-parallel* graph on k vertices. More generally, if $G = (V, E)$ contains a subgraph isomorphic to a graph $H = (V_H, E_H)$ whose *tree-width* is at most t, then such a subgraph can be found in $2^{O(k)} \cdot |V|^{t+1}$ expected time, where $k = |V_H|$. This improves an algorithm of Plehn and Voigt [14] that has a running time of $k^{O(k)} \cdot |V|^{t+1}$. As a very special case, it follows that the LOG PATH problem is in P. This resolves in the affirmative a conjecture of Papadimitriou and Yannakakis [13]. The exponential dependence on k in the above bounds is probably unavoidable as the problem is NP-complete if k is part of the input.

The color coding method has been a fruitful method in the study of parametrized algorithms and parametrized complexity [7, 8]. Recently, the method has found interesting applications in computational biology, specifically in detecting signaling pathways within protein interaction networks, see [10, 17, 18, 19].

Open Problems

Several problems, listed below, remain open.

- Is there a polynomial time (deterministic or randomized) algorithm for deciding if a given graph $G = (V, E)$ contains a path of length, say, $\log^2 |V|$? (This is unlikely, as it will imply the existence of an algorithm that decides in time $2^{O(\sqrt{n})}$ whether a given graph on n vertices is Hamiltonian.)
- Can the $\log |V|$ factor appearing in the derandomization be omitted?
- Is the problem of deciding whether a given graph $G = (V, E)$ contains a triangle as difficult as the Boolean multiplication of two $|V| \times |V|$ matrices?

Experimental Results

Results of running the basic algorithm on biological data have been reported in [17, 19].

Cross-References

- ▸ Approximation Schemes for Planar Graph Problems
- ▸ Graph Isomorphism
- ▸ Treewidth of Graphs

Recommended Reading

1. Alon N, Goldreich O, Håstad J, Peralta R (1992) Simple constructions of almost k-wise independent random variables. Random Struct Algorithms 3(3):289–304
2. Alon N, Yuster R, Zwick U (1995) Color coding. J ACM 42:844–856
3. Alon N, Yuster R, Zwick U (1997) Finding and counting given length cycles. Algorithmica 17(3):209–223
4. Björklund A, Husfeldt T (2003) Finding a path of superlogarithmic length. SIAM J Comput 32(6):1395–1402
5. Chen J, Lu S, Sze S, Zhang F (2007) Improved algorithms for path, matching, and packing problems. In: Proceedings of the 18th ACM-SIAM symposium on discrete algorithms (SODA), pp 298–307
6. Eppstein D (1999) Subgraph isomorphism in planar graphs and related problems. J Graph Algorithms Appl 3(3):1–27
7. Fellows MR (2003) New directions and new challenges in algorithm design and complexity, parameterized. In: Lecture notes in computer science, vol 2748, pp 505–519
8. Flum J, Grohe M (2004) The parameterized complexity of counting problems. SIAM J Comput 33(4):892–922
9. Fredman ML, Komlós J, Szemerédi E (1984) Storing a sparse table with $O(1)$ worst case access time. J ACM 31:538–544

10. Hüffner F, Wernicke S, Zichner T (2007) Algorithm engineering for color coding to facilitate signaling pathway detection. In: Proceedings of the 5th Asia-Pacific bioinformatics conference (APBC), pp 277–286
11. Monien B (1985) How to find long paths efficiently. Ann Discret Math 25:239–254
12. Naor J, Naor M (1993) Small-bias probability spaces: efficient constructions and applications. SIAM J Comput 22(4):838–856
13. Papadimitriou CH, Yannakakis M (1996) On limited nondeterminism and the complexity of the V-C dimension. J Comput Syst Sci 53(2):161–170
14. Plehn J, Voigt B (1990) Finding minimally weighted subgraphs. Lect Notes Comput Sci 484: 18–29
15. Robertson N, Seymour P (1986) Graph minors. II. Algorithmic aspects of tree-width. J Algorithms 7:309–322
16. Schmidt JP, Siegel A (1990) The spatial complexity of oblivious *k*-probe hash functions. SIAM J Comput 19(5):775–786
17. Scott J, Ideker T, Karp RM, Sharan R (2006) Efficient algorithms for detecting signaling pathways in protein interaction networks. J Comput Biol 13(2):133–144
18. Sharan R, Ideker T (2006) Modeling cellular machinery through biological network comparison. Nat Biotechnol 24:427–433
19. Shlomi T, Segal D, Ruppin E, Sharan R (2006) QPath: a method for querying pathways in a protein-protein interaction network. BMC Bioinform 7:199

Colouring Non-sparse Random Intersection Graphs

Christoforos L. Raptopoulos
Computer Science Department, University of Geneva, Geneva, Switzerland
Computer Technology Institute and Press "Diophantus", Patras, Greece
Research Academic Computer Technology Institute, Greece and Computer Engineering and Informatics Department, University of Patras, Patras, Greece

Keywords

Martingales; Probabilistic method; Proper coloring; Random intersection graphs; Random hypergraphs

Years and Authors of Summarized Original Work

2009; Nikoletseas, Raptopoulos, Spirakis

Problem Definition

A *proper coloring* of a graph $G = (V, E)$ is an assignment of colors to all vertices in V in such a way that no two adjacent vertices have the same color. A *k-coloring* of G is a coloring that uses k colors. The minimum number of colors that can be used to properly color G is the (*vertex*) *chromatic number* of G and is denoted by $\chi(G)$.

Deciding whether a given graph admits a k-coloring for a given $k \geq 3$ is well known to be NP complete. In particular, it is NP hard to compute the chromatic number [5]. The best known approximation algorithm computes a coloring of size at most within a factor $O\left(\frac{n(\log\log n)^2}{(\log n)^3}\right)$ of the chromatic number [6]. Furthermore, for any constant $\epsilon > 0$, it is NP hard to approximate the chromatic number within a factor $n^{1-\epsilon}$ [14].

The intractability of the vertex coloring problem for arbitrary graphs leads researchers to the study of the problem for appropriately generated random graphs. In the current entry, we consider coloring random instances of the *random intersection graphs model*, which is defined as follows:

Definition 1 (Random Intersection Graph – $\mathcal{G}_{n,m,p}$ [9, 13]) Consider a universe $\mathcal{M} = \{1, 2, \ldots, m\}$ of elements and a set of n vertices V. Assign independently to each vertex $v \in V$ a subset S_v of $\mathcal{M}$, choosing each element $i \in \mathcal{M}$ independently with probability p, and draw an edge between two vertices $v \neq u$ if and only if $S_v \cap S_u \neq \emptyset$. The resulting graph is an instance $G_{n,m,p}$ of the random intersection graphs model.

We will say that a property holds in $\mathcal{G}_{n,m,p}$ *with high probability (whp)* if the probability that a random instance of the $\mathcal{G}_{n,m,p}$ model has the property is at least $1 - o(1)$.

In this model, we will refer to the elements in the universe $\mathcal{M}$ as *labels*. We also denote by L_i the set of vertices that have chosen label $i \in M$.

Given $G_{n,m,p}$, we will refer to $\{L_i, i \in \mathcal{M}\}$ as its *label representation*. Consider the bipartite graph with vertex set $V \cup \mathcal{M}$ and edge set $\{(v,i) : i \in S_v\} = \{(v,i) : v \in L_i\}$. We will refer to this graph as the *bipartite random graph* $B_{n,m,p}$ *associated to* $G_{n,m,p}$. Notice that the associated bipartite graph is uniquely defined by the label representation.

It follows from the definition of the model that the edges in $G_{n,m,p}$ are not independent. This dependence becomes stronger as the number of labels decreases. In fact, the authors in [3] prove the equivalence (measured in terms of total variation distance) of the random intersection graphs model $\mathcal{G}_{n,m,p}$ and the Erdős-Rényi random graphs model $\mathcal{G}_{n,\hat{p}}$, for $\hat{p} = 1 - (1 - p^2)^m$, when $m = n^\alpha, \alpha > 6$. This bound on the number of labels was improved in [12], by showing equivalence of sharp threshold functions among the two models for $\alpha \geq 3$. We note that $1 - (1 - p^2)^m$ is in fact the (unconditioned) probability that a specific edge exists in $G_{n,m,p}$. In view of this equivalence, in this entry, we consider the interesting range of values $m = n^\alpha, \alpha < 1$, where random intersection graphs seem to differ the most from Erdős-Rényi random graphs.

In [1] the authors propose algorithms that whp probability color sparse instances of $G_{n,m,p}$. In particular, for $m = n^\alpha, \alpha > 0$ and $p = o\left(\sqrt{\frac{1}{nm}}\right)$, they show that $G_{n,m,p}$ can be colored optimally. Also, in the case where $m = n^\alpha, \alpha < 1$ and $p = o\left(\frac{1}{m \ln n}\right)$, they show that $\chi(G_{n,m,p}) \sim np$ whp. To do this, they prove that $G_{n,m,p}$ is chordal whp (or equivalently, the associated bipartite graph does not contain cycles), and so a perfect elimination scheme can be used to find a coloring in polynomial time. The range of values we consider here is different than the one needed for the algorithms in [1] to work. In particular, we study coloring $G_{n,m,p}$ for the wider range $mp \leq (1-\alpha)\ln n$, as well as the denser range $mp \geq \ln^2 n$. We have to note also that the proof techniques used in [1] cannot be used in the range we consider, since the properties that they examine do not hold in our case. Therefore, a completely different approach is required.

Key Results

In this entry, we initially considered the problem of properly coloring almost all vertices in $G_{n,m,p}$. In particular, we proved the following:

Theorem 1 *Let $m = n^\alpha, \alpha < 1$ and $mp \leq \beta \ln n$, for any constant $\beta < 1-\alpha$. Then a random instance of the random intersection graphs model $G_{n,m,p}$ contains a subset of at least $n - o(n)$ vertices that can be colored using np colors, with probability at least $1 - e^{-n^{0.99}}$.*

Note that the range of values of m, p considered in the above Theorem is quite wider than the one studied in [1]. For the proof, we combine ideas from [4] (see also [7]) and [10]. In particular, we define a Doob martingale as follows: Let $v_1, v_2, \ldots, v_n$ be an arbitrary ordering of the vertices of $G_{n,m,p}$. For $i = 1, 2, \ldots, n$, let B_i be the subgraph of the associated bipartite graph for $G_{n,m,p}$ (namely, $B_{n,m,p}$) induced by $\cup_{j=1}^{i} v_j \bigcup \mathcal{M}$. We denote by H_i the intersection graph whose bipartite graph has vertex set $V \bigcup \mathcal{M}$ and edge set that is exactly as B_i between $\cup_{j=1}^{i} v_j$ and $\mathcal{M}$, whereas every other edge (i.e., the ones between $\cup_{j=i+1}^{n} v_j$ and $\mathcal{M}$) appears independently with probability p.

Let also X denote the size of the largest np-colorable subset of vertices in $G_{n,m,p}$, and let X_i denote the expectation of the largest np-colorable subset in H_i. Notice that X_i is a random variable depending on the overlap between $G_{n,m,p}$ and H_i. Obviously, $X = X_n$ and setting $X_0 = E[X]$, we have $|X_i - X_{i+1}| \leq 1$, for all $i = 1, 2, \ldots, n$. It is straightforward to verify that the sequence $X_0, X_1, \ldots, X_n$ is a Doob martingale, and thus we can use Azuma's inequality to prove concentration of X_n around its mean value. However, the exact value of $E[X]$ is unknown. Nevertheless, we could prove a lower bound on $E[X]$ by providing a lower bound on the probability that X takes sufficiently large values. In particular, we showed that, for any positive constant $\epsilon > 0$, the probability that X takes values at least $(1-\epsilon)n$ is larger than the upper bound given by Azuma's inequality, implying that $E[X] \geq n - o(n)$.

It is worth noting here that the proof of Theorem 1 can also be used to prove that $\Theta(np)$

colors are enough to color $n - o(n)$ vertices, even in the case where $mp = \beta \ln n$, for any constant $\beta > 0$. However, finding the exact constant multiplying np is technically more difficult. Finally, note that Theorem 1 does not provide any direct information for the chromatic number of $G_{n,m,p}$, because the vertices that remain uncolored could induce a clique in $G_{n,m,p}$ in the worst case.

An Efficient Algorithm

Following our existential result of Theorem 1, we also proposed and analyzed an algorithm `CliqueColor` for finding a proper coloring of a random instance of $\mathcal{G}_{n,m,p}$, for any $mp \geq \ln^2 n$, where $m = n^\alpha, \alpha < 1$. The algorithm uses information of the label sets assigned to the vertices of $G_{n,m,p}$, and it runs in $O\left(\frac{n^2mp^2}{\ln n}\right)$ time whp (i.e., polynomial in n and m). In the algorithm, every vertex initially chooses independently uniformly at random a preference in colors from a set $\mathcal{C}$, denoted by $shade(\cdot)$, and every label l chooses a preference in the colors of the vertices in L_l, denoted by $c_l(\cdot)$. Subsequently, the algorithm visits every label clique and fixes the color (according to preference lists) for as many vertices as possible without causing collisions with already-colored vertices. Finally, it finds a proper coloring to the remaining uncolored vertices, using a new set of colors $\mathcal{C}'$. Algorithm `CliqueColor` is described below:

It is evident that algorithm `CliqueColor` always finds a proper coloring of $G_{n,m,p}$, but its efficiency depends on the number of colors included in $\mathcal{C}$ and $\mathcal{C}'$. The main idea is that if we have enough colors in the initial color set $\mathcal{C}$, then the subgraph $\mathcal{H}$ containing uncolored vertices will have sufficiently small maximum degree $\Delta(\mathcal{H})$, so that we can easily color it using $\Delta(\mathcal{H})$ extra colors. More specifically, we prove the following:

Theorem 2 (Efficiency) *Let $m = n^\alpha, \alpha < 1$ and $mp \geq \ln^2 n, p = o\left(\frac{1}{\sqrt{m}}\right)$. Then algorithm `CliqueColor` succeeds in finding a proper $\Theta\left(\frac{nmp^2}{\ln n}\right)$-coloring using for $G_{n,m,p}$ in polynomial time whp.*

Algorithm `CliqueColor`:

Input: An instance $G_{n,m,p}$ of $\mathcal{G}_{n,m,p}$ and its associated bipartite $B_{n,m,p}$.
Output: A proper coloring of $G_{n,m,p}$.

1. for every $v \in V$ choose a color denoted by $shade(v)$ independently, uniformly at random among those in $\mathcal{C}$;
2. for every $l \in \mathcal{M}$, choose a coloring of the vertices in L_l such that, for every color in $\{c \in \mathcal{C} : \exists v \in L_l$ with $shade(v) = c\}$, there is exactly one vertex in the set $\{u \in L_l : shade(u) = c\}$ having $c_l(u) = c$, while the rest remain uncolored;
3. set $U = \emptyset$ and $C = \emptyset$;
4. **for** $l = 1$ **to** m **do** {
5. color every vertex in $L_l \setminus (U \cup C)$ according to $c_l(\cdot)$ iff there is no collision with the color of a vertex in $L_l \cap C$;
6. include every vertex in L_l colored that way in C and the rest in U; } **enddo**
7. let $\mathcal{H}$ denote the (intersection) subgraph of $G_{n,m,p}$ induced by the vertices in U and let $\Delta(\mathcal{H})$ be its maximum degree;
8. give a proper $\Delta(\mathcal{H})$-coloring of $\mathcal{H}$ using a new set of colors $\mathcal{C}'$ of cardinality $\Delta(\mathcal{H})$;
9. **output** a coloring of $G_{n,m,p}$ using $|\mathcal{C} \cup \mathcal{C}'|$ colors;

It is worth noting that the number of colors used by algorithm `CliqueColor` in the case $mp \geq \ln^2 n, p = O\left(\frac{1}{\sqrt[4]{m}}\right)$ and $m = n^\alpha, \alpha < 1$ is of the correct order of magnitude (i.e., it is optimal up to constant factors). Indeed, by the concentration of the values of $|S_v|$ around mp for any vertex v with high probability, we can use the results of [11] on the independence number of the *uniform random intersection graphs model* $\mathcal{G}_{n,m,\lambda}$, with $\lambda \sim mp$, to provide a lower bound on the chromatic number. Indeed, it can be easily verified that the independence number of $G_{n,m,\lambda}$ for $\lambda = mp \geq \ln^2 n$ is at most $\Theta\left(\frac{\ln n}{mp^2}\right)$, which implies that the chromatic number of $G_{n,m,\lambda}$ (hence also of the $G_{n,m,p}$ because of the concentration of the values of $|S_v|$) is at least $\Omega\left(\frac{nmp^2}{\ln n}\right)$.

Coloring Random Hypergraphs

The model of random intersection graphs $\mathcal{G}_{n,m,p}$ could also be thought of as generating random hypergraphs. The hypergraphs generated have vertex set V and edge set $\mathcal{M}$. There is a huge amount of literature concerning coloring hypergraphs. However, the question about coloring

there seems to be different from the one we considered here. More specifically, a proper coloring of a hypergraph is any assignment of colors to the vertices, so that no monochromatic edge exists. This of course implies that fewer colors than the chromatic number (studied in this entry) are needed in order to achieve this goal.

In $\mathcal{G}_{n,m,p}$, the problem of finding a coloring such that no label is monochromatic seems to be quite easier when p is not too small. The proof of the following Theorem is based on the method of conditional expectations (see [2, 8]).

Theorem 3 *Let $G_{n,m,p}$ be a random instance of the model $\mathcal{G}_{n,m,p}$, for $p = \omega\left(\frac{\ln m}{n}\right)$ and $m = n^{\alpha}$, for any fixed $\alpha > 0$. Then with high probability, there is a polynomial time algorithm that finds a k-coloring of the vertices such that no label is monochromatic, for any fixed integer $k \geq 2$.*

Applications

Graph coloring enjoys many practical applications as well as theoretical challenges. Beside the classical types of problems, different limitations can also be set on the graph, or on the way a color is assigned, or even on the color itself. Some of the many applications of graph coloring include modeling scheduling problems, register allocation, pattern matching, etc.

Random intersection graphs are relevant to and capture quite nicely social networking. Indeed, a social network is a structure made of nodes (individuals or organizations) tied by one or more specific types of interdependency, such as values, visions, financial exchange, friends, conflicts, web links, etc. Social network analysis views social relationships in terms of nodes and ties. Nodes are the individual actors within the networks and ties are the relationships between the actors. Other applications include oblivious resource sharing in a (general) distributed setting, efficient and secure communication in sensor networks, interactions of mobile agents traversing the web, etc. Even epidemiological phenomena (like spread of disease) tend to be more accurately captured by this "interaction-sensitive" random graphs model.

Open Problems

In [1], the authors present (among other results) an algorithm for coloring $G_{n,m,p}$ in the case where $m = n^{\alpha}, \alpha < 1$ and $mp = o\left(\frac{1}{\log n}\right)$. In contrast, we presented algorithm `CliqueColor`, which finds a proper coloring of $G_{n,m,p}$ using $\Theta(\chi(G_{n,m,p}))$ whp, in the case $m = n^{\alpha}, \alpha < 1$ and $mp \geq \ln^2 n$. It remains open whether we can construct efficient algorithms (both in terms of the running time and the number of colors used) for finding proper colorings of $G_{n,m,p}$ for the range of values $\Omega\left(\frac{1}{\log n}\right) \leq mp \leq \ln^2 n$.

Recommended Reading

1. Behrisch M, Taraz A, Ueckerdt M (2009) Colouring random intersection graphs and complex networks. SIAM J Discret Math 23(1):288–299
2. Erdős P, Selfridge J (1973) On a combinatorial game. J Comb Theory (A) 14:298–301
3. Fill JA, Sheinerman ER, Singer-Cohen KB (2000) Random intersection graphs when $m = \omega(n)$: an equivalence theorem relating the evolution of the $G(n, m, p)$ and $G(n, p)$ models. Random Struct Algorithms 16(2):156–176
4. Frieze A (1990) On the independence number of random graphs. Discret Math 81:171–175
5. Garey MR, Johnson DS, Stockmeyer L (1974) Some simplified NP-complete problems. In: Proceedings of the sixth annual ACM symposium on theory of computing, Seattle, pp 47–63. doi:10.1145/800119.803884
6. Halldórsson MM (1993) A still better performance guarantee for approximate graph coloring. Inf Process Lett 45:19–23. doi:10.1016/0020-0190(93)90246-6
7. Łuczak L (2005) The chromatic number of random graphs. Combinatorica 11(1):45–54
8. Molloy M, Reed B (2002) Graph colouring and the probabilistic method. Springer, Berlin/Heidelberg
9. Karoński M, Scheinerman ER, Singer-Cohen KB (1999) On random intersection graphs: the subgraph problem. Comb Probab Comput J 8:131–159
10. Nikoletseas SE, Raptopoulos CL, Spirakis PG (2008) Large independent sets in general random intersection graphs. Theor Comput Sci (TCS) J Spec Issue Glob Comput 406(3):215–224

11. Nikoletseas SE, Raptopoulos CL, Spirakis PG (2009) Combinatorial properties for efficient communication in distributed networks with local interactions. In: Proceedings of the 23rd IEEE international parallel and distributed processing symposium (IPDPS), Rome, pp 1–11
12. Rybarczyk K (2011) Equivalence of a random intersection graph and $G(n, p)$. Random Struct Algorithms 38(1–2):205–234
13. Singer-Cohen KB (1995) Random intersection graphs. PhD thesis, John Hopkins University
14. Zuckerman D (2007) Linear degree extractors and the inapproximability of max clique and chromatic number. Theory Comput 3:103–128. doi:10.4086/toc.2007.v003a006

Combinatorial Gray Code

Frank Ruskey
Department of Computer Science, University of Victoria, Victoria, BC, Canada

Keywords

Combinatorial object; Gray code; Hamilton cycle; Loopless algorithm

Years and Authors of Summarized Original Work

1989; Savage
1993; Pruesse, Ruskey
2012; Ruskey, Sawada, Williams
2012; Sawada, Williams

Problem Definition

In the field of *combinatorial generation*, the goal is to have fast elegant algorithms and code for exhaustively listing the elements of various combinatorial classes such as permutations, combinations, partitions, trees, graphs, and so on. Often it is desirable that successive objects in the listing satisfy some closeness condition, particularly conditions where successive objects differ only by a constant amount. Such listings are called *combinatorial Gray codes*; thus the study of combinatorial Gray codes is an important subfield of combinatorial generation.

There are a variety of applications of combinatorial objects where there is an inherent *closeness operation* that takes one object to another object and vice versa. There is a natural *closeness graph* $G = (V, E)$ that is associated with this setting. The vertices, V, of this graph are the combinatorial objects and edges, E, are between objects that are close. A combinatorial Gray code then becomes a Hamilton path in G and a *Gray cycle* is a Hamilton cycle in G.

The term *Combinatorial Gray Code* seems to have first appeared in print in Joichi, White, and Williamson [4]. An excellent survey up to 1997 was provided by Savage [10], and many examples of combinatorial Gray codes may be found in Knuth [5]. There are literally thousands of papers and patents on Gray codes in general, and although fewer about combinatorial Gray codes, this article can only scratch the surface and will focus on fundamental concepts, generalized settings, and recent results.

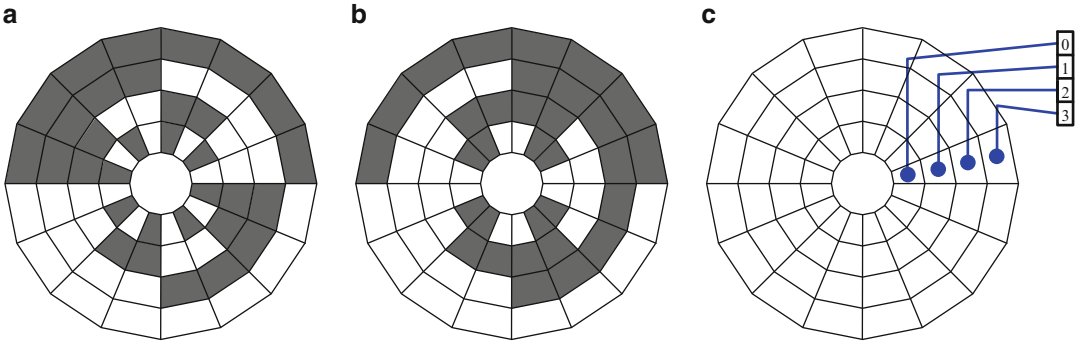
The Binary Reflected Gray Codes

The binary reflected Gray code (BRGC) is a well-known circular listing of the bit strings of a fixed length in which successive bit strings differ by a single bit. (The word *bit string* is used in this article instead of "binary string.") Let B_n be the Gray code list of all bit strings of length n. The list is defined by the following simple recursion:

$$B_0 = \varepsilon, \quad \text{and for } n > 0 \quad B_{n+1} = 0B_n, 1B_n^R. \tag{1}$$

In this definition, ε is the empty string, the comma represents concatenation of lists, the symbol x preceding a list indicates that an x is to be prepended to every string in list, and the superscript R indicates that the list is reversed. For example, $B_1 = 0, 1$, $B_1 = 00, 01, 11, 10$, and $B_3 = 000, 001, 011, 010, 110, 111, 101, 100$.

Figure 1 illustrates one of the most useful applications of the BRGC. Imagine a rotating "shaft," like a photocopier drum or a shutoff valve



Combinatorial Gray Code, Fig. 1 Applications of the binary reflected Gray code (**a**, **b**, **c**: see explanation in the text)

on a pipeline, in which the amount of rotation is to be determined by reading n "bits" from a sensor (c). If the sensor position is between two adjacent binary strings, then the result is indeterminate; so, for example, if the sensor fell between 0110 and 1100, then it could return either of these, but also 0100 or 1110 could be returned. If the bits are encoded around the shaft as a sequence of bit strings in lexicographic order, as shown in (a), and the sensor falls between 0000 and 1111, then any bit string might be returned! This problem is entirely mitigated by using a Gray code, such as the BRGC (b), since then only one of the two bit strings that lay under the sensor will be returned.

The term "Gray" comes from Frank Gray and engineer at Bell Labs who was issued a patent that uses the BRGC (Pulse code communication, March 17, 1953. U.S. patent no. 2,632,058). However, the BRGC was known much earlier; in particular it occurs in solutions to the Chinese rings puzzle and was used to list diagrams in the I Ching. See Knuth [5] for further historical background.

Surprisingly, new things are yet being proved about the BRGC; e.g., Williams [15] shows that the BRGC is generated by the following iterative "greedy" rule: Starting with the all 0s bit string, flip the rightmost bit that yields a bit string that has not already been generated. There are many other useful Gray code listings of bit strings known, for example, where the bit flips are equally distributed over the indexing positions or where all bit strings of density k (i.e., having k 1s) are listed before any of those of density $k+2$.

Combinations as Represented by Bit Strings

Many other simple Gray codes can be described using rules in the style of (1). For example, $B_{n,k} = 0B_{n-1,k}, 1B^R_{n-1,k-1}$ gives a Gray code of all bit strings of length n with exactly k 1s. It has the property that successive bit strings differ by the transposition of two (possibly nonadjacent) bits. Observe also that this is the list obtained from the BRGC by deleting all bit strings that do not contain k 1s.

A more restrictive Gray code for combinations is that of Eades and McKay [1]. Here the recursive construction ($1 < k < n$) is $E_{n,k} = 0E_{n-1,k-1}, 10E^R_{n-2,k-1}, 11E_{n-2,k-2}$. This Gray code has the nice property that successive bit strings differ by a transposition of two bits *and* only 0s lie between the two transposed bits. It is worth noting that there is no Gray code for combinations by transpositions in which the transposed bits are always adjacent (unless n is even and k is odd).

Generating Permutations via Plain Changes

The second most well-known combinatorial Gray code lists all $n!$ permutations of $\{1, 2, \ldots, n\}$ in one-line notation in such a way that successive permutations differ by the transposition of two adjacent elements. In its algorithmic form it is usually attributed to Johnson [3] and Trotter [13] although it has been used for centuries by campanologists [5], who refer to it as *plain changes*. Given such a list L_{n-1} for $\{1, 2, \ldots, n-1\}$, the list L_n can be created by successively sweeping

the n back-and-forth through each permutation in L_{n-1}. For example, if L_3 is 123, 132, 312, 321, 231, 213, then the first 8 permutations of L_4 are 1234, 1243, 1423, 4123, 4132, 1432, 1342, 1324. The "weave" below illustrates plane changes for $n = 5$, where the permutations occur as columns and the leftmost column corresponds to the permutation 12345, read top to bottom.

Hamiltonicity

In our BRGC example above, the closeness operation is the flipping of a bit, and the closeness graph is the hypercube. In a Gray code for permutations, a natural closeness operation is the transposing of two adjacent elements; in this case the closeness graph is sometimes called the permutohedron.

Sometimes it happens that the closeness graph G has no Hamilton path. If it is not connected, then there is no hope of finding a Gray code, but if it is connected, then there are several approaches that have proved successful in finding a Gray code in a related graph; the result is a weaker Gray code that permits 2 or 3 applications of the closeness operation. One approach is to consider the prism $G \times e$ [7]. If G is bipartite and Hamiltonian, then the Gray code consists of every other vertex along the Hamilton cycle. As a last resort the result of Sekanina below [12] can be used. An implementation of Sekanina's proof, called prepostorder, is given by Knuth [5].

Theorem 1 (Sekanina) *If G is connected, then G^3 is Hamiltonian.*

At this point the reader may be wondering: "What distinguishes the study of Gray codes from the study of Hamiltonicity of graphs?" The underlying motivation for most combinatorial Gray codes is algorithmic; proving the existence of the Hamilton cycle is only the first step; one then tries to get an efficient implementation according to the criteria discussed in the next section (e.g., the recent "middle-level" result gives an existence proof via a complex construction – but it lacks the nice construction that one would strive for when thinking about a Gray code). Another difference is that the graphs studied usually have a very specific underlying structure; often this structure is recursive and is exploited in the development of efficient algorithms.

The Representation Issue

Many combinatorial structures exhibit a chameleonic nature. The canonical example of this is the objects counted by the Catalan numbers, C_n, of which there are hundreds. For example, they count the well-formed parentheses strings with n left and n right parentheses but also count the binary trees with n nodes. The most natural ways of representing these are very different; well-formed parentheses are naturally represented by bit strings of length $2n$, whereas binary trees are naturally represented using a linked structure with two pointers per cell. Furthermore, the natural closeness conditions are also different; for parentheses, a swap of two parentheses is natural, and for binary trees, the class rotation operation is most natural. When discussing Gray codes it is imperative to know precisely what representation is being used (and then what closeness operation is being used).

Algorithmic Issues

For the vast majority of combinatorial Gray codes, *space* is the main enemy. The first task of the algorithm designer is to make sure that their algorithm uses an amount of space that is a small polynomial of the input size; algorithms that rely on sublists of the objects being listed are doomed even though many Hamiltonicity proofs naïvely lead to such algorithms. For example, an efficient algorithm for generating the BRGC cannot directly use (1) since that would require exponential space.

CAT Algorithms, Loopless Algorithms

From the global point of view, the best possible algorithms are those that output the objects (V)

in time that is proportional to the number of objects ($|V|$). Such algorithms are said to be CAT, standing for constant amortized time.

From a more local point of view, the best possible algorithms are those that output successive objects so that the amount of work between successive objects is constant. Such algorithms are said to be *loopless*, a term introduced by Ehrlich [2]. Both the BRGC and the plain changes algorithm for permutations mentioned above can be implemented as loopless algorithms.

Note that in both of these definitions we ignore the time that it would take to actually output the objects; what is being measured is the amount of data structure change that is occurring. This measure is the correct one to use because in many applications it is only the part of the data structure that changes that is really needed by the application (e.g., to update an objective function).

Key Results

Below are listed some combinatorial Gray codes, focusing on those that are representative, are very general, or are recent breakthroughs.

Numerical Partitions [5, 9]

Objects: All numerical partitions of an integer n.

Representation: Sequence of positive integers $a_1 \geq a_2 \geq \cdots$ such that $a_1 + a_2 \cdots = n$.

Closeness operation: Two partitions a and a' are close if there are two indices i and j such that $a'_i = a_i + 1$ and $a'_j = a_j - 1$.

Efficiency: CAT.

Comments: Results have been extended to the case where all parts are distinct and where the number of parts or the largest part is fixed.

Spanning Trees

Objects: Spanning trees of connected unlabeled graphs.

Representation: List of edges in the graph.

Closeness operation: Successive trees differ by a single edge replacement.

Efficiency: CAT.

References: University of Victoria MSc thesis of Malcolm Smith. Knuth 4A, [5], pages 468–469. Knuth has an implementation of the Smith algorithm on his website mentioned below; see the programs *SPAN.

Basic Words of Antimatroids [7]

Objects: Let $\mathcal{A}$ be a set system over $[n] := \{1, 2, \ldots, n\}$ that (a) is closed under union ($S \cup T \in \mathcal{A}$ for all $S, T \in \mathcal{A}$) and (b) is accessible (for all $S \in \mathcal{A}$ with $S \neq \emptyset$, there is an $x \in S$ such that $S \setminus \{x\} \in \mathcal{A}$). Such a set system $\mathcal{A}$ is an *antimatroid*. Repeated application of (b) starting with the set $[n]$ and ending with $\emptyset$ gives a permutation of $[n]$ called a *basic word*.

Representation: A permutation of $[n]$ in one-line notation.

Closeness operation: Successive permutations differ by the transposition of one or two adjacent elements.

Efficiency: CAT if there is an $O(1)$ "oracle" for determining whether the closeness operation applied to a basic word gives another basic word of $\mathcal{A}$; loopless if the antimatroid is the set of ideals of a poset.

Important special cases: Linear extensions of posets (partially ordered sets), convex shellings of finite point sets, and perfect elimination orderings of chordal graphs. Linear extensions have as special cases permutations (of a multiset), k-ary trees, standard young tableau, alternating permutations, etc.

Additional notes: If G is the cover graph of an antimatroid $\mathcal{A}$ where the sets are ordered by set inclusion, then the prism of G is Hamiltonian. Thus there is a Gray code for the elements of $\mathcal{A}$. No CAT algorithm is known for this Gray code, even in the case where the antimatroid consists of the ideals of a poset.

Words in a Bubble Language [8, 11]

Objects: A language L over alphabet $\{0, 1\}$ is a *bubble language* if it is closed under the operation of changing the rightmost 01 to a 10.

Representation: Bit strings of length n (i.e., elements of $\{0, 1\}^n$).

Closeness operation: Two $01 \leftrightarrow 10$ swaps, or (equivalently) one rotation of a prefix.

Important special cases: Combinations, well-formed parentheses, necklaces, and prefix normal words.
Efficiency: CAT if there is an $O(1)$ oracle for determining membership under the operation.

Permutations via σ and τ [16]

Objects: Permutations of $[n]$.
Representation: One-line notation.
Closeness operation: Successive permutations differ by the rotation $\sigma = (1\ 2\ \cdots\ n)$ or the transposition $\tau = (1\ 2)$ as applied to the indices of the representation.
Efficiency: CAT, but has a loopless implementation if only the successive σ or τ generators are output or if the permutation is represented using linked lists.
Comments: This is known as Wilf's (directed) $\sigma - \tau$ problem.

Middle Two Levels of the Boolean Lattice [6]

Objects: All subsets S of $\mathcal{B}_{2n+1}$ of density n or $n+1$.
Representation: Characteristic bit strings ($b_i = 1$ if and only if $i \in S$).
Closeness operation: Bit strings differ by a transposition of adjacent bits.
Efficiency: Unknown. A good open problem.
Comments: This is famously known as the "middle two-level problem."

URLs to Code and Data Sets

Don Knuth maintains a web page which contains some implementations of combinatorial Gray codes at http://www-cs-faculty.stanford.edu/~uno/programs.html. See, in particular, the programs GRAYSPAN, SPSPAN, GRAYSPSPAN, KODA-RUSKEY, and SPIDERS.

Jeorg Arndt maintains a website (http://www.jjj.de/fxt/) and book with many C programs for generating combinatorial objects, some of which are combinatorial Gray codes. The book may be freely downloaded. Chapter 14, entitled "Gray codes for strings with restrictions", is devoted to combinatorial Gray codes, but they can also be found in other chapters; e.g., 15.2 and 15.3 both contain Gray codes for well-formed parentheses strings.

The "Combinatorial Object Server" at http://www.theory.cs.uvic.ca/~cos/ allows you to produce small lists of combinatorial objects, often in various Gray code orders.

Cross-References

Entries relevant to combinatorial generation (not necessarily Gray codes):

- Enumeration of Non-crossing Geometric Graphs
- Enumeration of Paths, Cycles, and Spanning Trees
- Geometric Object Enumeration
- Permutation Enumeration
- Tree Enumeration

Recommended Reading

1. Eades P, McKay B (1984) An algorithm for generating subsets of fixed size with a strong minimal change property. Inf Process Lett 19:131–133
2. Ehrlich G (1973) Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. J ACM 20:500–513
3. Johnson S (1963) Generation of permutations by adjacent transposition. Math Comput 17:282–285
4. Joichi JT, White DE, Williamson SG (1980) Combinatorial Gray codes. SIAM J Comput 9(1):130–141
5. Knuth DE (2011) The art of computer programming. Combinatorial algorithms, vol 4A, Part 1. Addison-Wesley, Upper Saddle River, xvi+883pp. ISBN 0-201-03804-8
6. Mütze T (2014) Proof of the middle levels conjecture. arXiv:1404.4442
7. Pruesse G, Ruskey F (1993) Gray codes from antimatroids. Order 10:239–252
8. Ruskey F, Sawada J, Williams A (2012) Binary bubble languages. J Comb Theory Ser A 119(1):155–169
9. Savage C (1989) Gray code sequences of partitions. J Algorithms 10:577–595
10. Savage C (1997) A survey of combinatorial Gray codes. SIAM Rev 39:605–629
11. Sawada J, Williams A (2012) Efficient oracles for generating binary bubble languages. Electron J Comb 19:Paper 42
12. Sekanina M (1960) Spisy Přírodovědecké. Fakulty University v Brně 412:137–140

13. Trotter H (1962) Algorithm 115: Perm. Commun ACM 5:434–435
14. Wilf HS (1989) Combinatorial algorithms: an update. In: CBMS-NSF regional conference series in applied mathematics. SIAM. http://epubs.siam.org/doi/book/10.1137/1.9781611970166
15. Williams A (2013) The greedy Gray code algorithm. In: Algorithms and data structures symposium (WADS 2013), London, Canada. LNCS, vol 8037, pp 525–536
16. Williams A (2014) Hamiltonicity of the Cayley digraph on the symmetric group generated by $\sigma = (1\ 2\ \cdots\ n)$ and $\tau = (1\ 2)$. arXiv:1307.2549

Combinatorial Optimization and Verification in Self-Assembly

Robert Schweller
Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, TX, USA

Keywords

Algorithmic self-assembly; Tile Assembly Model; Tile complexity; Two-Handed Assembly; Self-assembly

Years and Authors of Summarized Original Work

2002; Adleman, Cheng, Goel, Huang, Kempe, Moisset, Rothemund
2013; Cannon, Demaine, Demaine, Eisenstat, Patitz, Schweller, Summers, Winslow

Problem Definition

Tile Assembly Models

Two of the most studied tile self-assembly models in the literature are the abstract Tile Assembly Model (aTAM) [7] and the Two-Handed Tile Assembly Model (2HAM) [4]. Both models constitute a mathematical model of self-assembly in which system components are four-sided Wang tiles with glue types assigned to each tile edge. Any pair of glue types are assigned some nonnegative interaction strength denoting how strongly the pair of glues bind. The models differ in their rules for growth in that the aTAM allows singleton tiles to attach one at a time to a growing seed, whereas the 2HAM permits any two previously built assemblies to combine given enough affinity for attachment.

In more detail, an aTAM system is an ordered triplet (T, τ, σ) consisting of a set of tiles T, a positive integer threshold parameter τ called the system's *temperature*, and a special tile $\sigma \in T$ denoted as the *seed* tile. Assembly proceeds by attaching copies of tiles from T to a growing seed assembly whenever the placement of a tile on the 2D grid achieves a total strength of attachment from abutting edges, determined by the sum of pairwise glue interactions, that meets or exceeds the temperature parameter τ. An additional twist that is often considered is the ability to specify a relative concentration distribution on the tiles in T. The growth from the initial seed then proceeds randomly with higher concentrated tile types attaching more quickly than lower concentrated types. Even when the final assembly is deterministic, adjusting concentration profiles may substantially alter the expected time to reach the unique terminal state.

The Two-Handed Tile Assembly Model (2HAM) [4] is similar to the aTAM, but removes the concept of a seed tile. Instead, a 2HAM system (T, τ) *produces* a new assembly whenever any two previously built (and potentially large) assemblies may translate together into a new stable assembly based on glue interactions and temperature. The distinction between the 2HAM and the aTAM is that the 2HAM allows large assemblies to grow independently and attach as large, pre-built assemblies, while the aTAM grows through the step-by-step attachment of singleton tiles to a growing seed.

A typical goal in tile self-assembly is to design an *efficient* tile system that uniquely assembles a target shape. Two primary efficiency metrics are (1) the number of distinct tile types used to self-assemble the target shape and (2) the expected time the system takes to self-assemble the target shape. Toward minimizing the number of tiles used to build a shape, the Minimum Tile Set

Problem is considered. Toward the goal of minimizing assembly time, the problem of selecting an optimal concentration distribution over the tiles of a given set is considered in the Tile Concentration Problem. Finally, the computational problem of simply verifying whether a given system correctly and uniquely self-assembles a target shape is considered in the Unique Assembly Verification Problem. Formally, the problems are as follows:

Problem 1 (The Minimum Tile Set Problem [2]) Given a shape, find the tile system with the minimum number of tile types that uniquely self-assembles into this shape.

Problem 2 (The Tile Concentration Problem [2]) Given a shape and a tile system that uniquely produces the given shape, assign concentrations to each tile type so that the expected assembly time for the shape is minimized.

Problem 3 (The Unique Assembly Verification Problem [2, 4]) Given a tile system and an assembly, determine if the tile system uniquely self-assembles into the assembly.

Key Results

Minimum Tile Set Problem

The NP-completeness of the Minimum Tile Set Problem within the aTAM is proven in [2] by a reduction from 3CNF-SAT. The proof is notable in that the polynomial time reduction relies on the polynomial time solution of the Minimum Tile Set Problem for tree shapes, which the authors show is polynomial time solvable. The authors also show that the Minimum Tile Set Problem is polynomial time solvable for $n \times n$ squares by noting that since the optimal solution has at most $O(\log n)$ tile types [7], a brute force search of candidate tile sets finishes in polynomial time as long as the temperatures of the systems under consideration are all a fixed constant. Extending the polynomial time solution to find the minimum tile system over any temperature is achieved in [5].

Theorem 1 *The Minimum Tile Set Problem is NP-complete within the aTAM. For the restricted classes of shapes consisting of squares and trees, the Minimum Tile Set Problem is polynomial time solvable.*

Concentration Optimization

The next result provides an approximation algorithm for the Tile Concentration Problem for a restricted class of aTAM tile system called *partial order* systems. Partial order systems are systems in which a unique assembly is constructed, and for any pair of adjacent tiles in the final assembly which have positive bonding strength, there is a strict order in which the two tiles are placed with respect to each other for all possible assembly sequences. For such systems, a $O(\log n)$-approximation algorithm is presented [2].

Theorem 2 *For any partial order aTAM system (T, τ, σ) that uniquely self-assembles a size-n assembly, there exists a polynomial time $O(\log n)$-approximation algorithm for the Tile Concentration Problem.*

Assembly Verification

The next result provides an important distinction in verification complexity between the aTAM and the 2HAM. In [2] a straightforward quadratic time algorithm for assembly verification is presented. In contrast, the problem is shown to be co-NP-complete in [4] through a reduction from 3CNF-SAT. The hardness holds for a 3D generalization of the 2HAM, but requires only 1 step into the third dimension. To achieve this reduction, the exponentially many candidate 3CNF-SAT solutions are engineered into the order in which the system might grow while maintaining that these candidate paths all collapse into a single polynomial-sized final assembly in the case that no satisfying solution exists. This reduction fundamentally relies on the third dimension and thus leaves open the complexity of 2D verification in the 2HAM.

Theorem 3 *The Unique Assembly Verification Problem is co-NP-complete for the 3D 2HAM and solvable in polynomial time $O(|A|^2+|A||T|)$ in the aTAM.*

Open Problems

A few open problems in this area are as follows. The Minimum Tile Set Problem has an efficient solution for squares which stems from a logarithmic upper bound on the complexity of assembling such shapes. This holds more generally for thick rectangles, but this ceases to be true when the width of the rectangle becomes sufficiently thin [3]. The complexity of the Minimum Tile Set Problem is open for this class of simple geometric shapes. For the Tile Concentration Problem, an exact solution is conjectured to be #P-hard for partial order systems [2], but this has not been proven. More generally, little is known about the Tile Concentration Problem for non-partial order systems. Another direction within the scope of minimizing assembly time is to consider optimizing over the tiles used, as well as the concentration distribution over the tile set. Some work along these lines has been done with respect to the fast assembly of $n \times n$ squares [1] and the fast implementation of basic arithmetic primitives in self-assembly [6]. In the case of the Unique Assembly Verification Problem, the complexity of the problem for the 2HAM in 2D is still unknown. For the aTAM, it is an open question as to whether the quadratic run time of verification can be improved.

Cross-References

- Active Self-Assembly and Molecular Robotics with Nubots
- Combinatorial Optimization and Verification in Self-Assembly
- Intrinsic Universality in Self-Assembly
- Patterned Self-Assembly Tile Set Synthesis
- Randomized Self-Assembly
- Robustness in Self-Assembly
- Self-Assembly at Temperature 1
- Self-Assembly of Fractals
- Self-Assembly of Squares and Scaled Shapes
- Self-Assembly with General Shaped Tiles
- Temperature Programming in Self-Assembly

Recommended Reading

1. Adleman L, Cheng Q, Goel A, Huang M-D (2001) Running time and program size for self-assembled squares. In: Proceedings of the thirty-third annual ACM symposium on theory of computing. ACM, New York, pp 740–748
2. Adleman LM, Cheng Q, Goel A, Huang M-DA, Kempe D, de Espanés PM, Rothemund PWK (2002) Combinatorial optimization problems in self-assembly. In: Proceedings of the thirty-fourth annual ACM symposium on theory of computing (STOC '02), Montreal. ACM, New York, pp 23–32
3. Aggarwal G, Cheng Q, Goldwasser MH, Kao M-Y, de Espanés PM, Schweller RT (2005) Complexities for generalized models of self-assembly. SIAM J Comput 34:1493–1515
4. Cannon S, Demaine ED, Demaine ML, Eisenstat S, Patitz MJ, Schweller R, Summers SM, Winslow A (2013) Two hands are better than one (up to constant factors): self-assembly in the 2HAM vs. aTAM. In: Portier N, Wilke T (eds) STACS. LIPIcs, vol 20, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, Wadern, pp 172–184
5. Chen H-L, Doty D, Seki S (2011) Program size and temperature in self-assembly. In: ISAAC 2011: proceedings of the 22nd international symposium on algorithms and computation. Lecture notes in computer science, vol 7074. Springer, Berlin/New York, pp 445–453
6. Keenan A, Schweller R, Sherman M, Zhong X (2014) Fast arithmetic in algorithmic self-assembly. In: Unconventional computation and natural computation. Lecture notes in computer science. Springer, Cham, pp 242–253
7. Rothemund PWK, Winfree E (2000) The program-size complexity of self-assembled squares (extended abstract). In: Proceedings of the 32nd ACM symposium on theory of computing, STOC'00. ACM, New York, pp 459–468

Communication Complexity

Amit Chakrabarti
Department of Computer Science, Dartmouth College, Hanover, NH, USA

Keywords

Circuits; Communication; Compression; Data streams; Data structures; Direct sum; Fingerprinting; Information theory; Lower bounds; Randomization

Years and Authors of Summarized Original Work

1977; Yao
1979; Yao
1986; Babai, Frankl, Simon
1991; Newman
1992; Razborov
1995; Feder, Kushilevitz, Naor, Nisan
1997; Kushilevitz, Nisan
1998; Miltersen, Nisan, Safra, Wigderson
2001; Chakrabarti, Shi, Wirth, Yao
2004; Bar-Yossef, Jayram, Kumar, Sivakumar
2009; Lee, Shraibman
2010; Barak, Braverman, Chen, Rao
2010; Jain, Klauck
2011; Braverman, Rao
2012; Braverman
2012; Chakrabarti, Regev
2014; Brody, Chakrabarti, Kondapally, Woodruff, Yaroslavtsev
2014; Ganor, Kol, Raz
2014; Lovett

Problem Definition

Two players – Alice and Bob – are playing a game in which their shared goal is to compute a function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ efficiently. The game starts with Alice holding a value $x \in \mathcal{X}$ and Bob holding $y \in \mathcal{Y}$. They then communicate by sending each other messages according to a predetermined *protocol*, at the end of which they must both arrive at some output $z \in \mathcal{Z}$. The protocol is deemed *correct* if $z = f(x, y)$ for all inputs (x, y). Each message from Alice (resp. Bob) is an arbitrary binary-string-valued function of x (resp. y) and all previous messages received during the protocol's execution. The *cost* of the protocol is the maximum total length of all such messages, over all possible inputs, and is the basic measure of efficiency of the protocol. The central goals in communication complexity [23] are (i) to design protocols with low cost for given problems of interest, and (ii) to prove lower bounds on the cost that must be paid to solve a given problem. The minimum possible such cost is a natural measure of complexity of the function f and is denoted $\mathrm{D}(f)$.

Notably, the "message functions" in the above definition are not required to be efficiently computable. Thus, communication complexity focuses on certain basic information theoretic aspects of computation, abstracting away messier and potentially unmanageable lower-level details. Arguably, it is this aspect of communication complexity that has made it such a successful paradigm for proving lower bounds in a wide range of areas in computer science.

Most work in communication complexity focuses on *randomized* protocols, wherein random coin tosses (equivalently, a single random binary string) may be used to determine the messages sent. These coin tosses may be performed either in private by each player or in public: the resulting protocols are called private coin and public coin, respectively. A randomized protocol is said to compute f with error bounded by $\varepsilon \geq 0$ if, for all inputs (x, y), its output on (x, y) differs from $f(x, y)$ with probability at most ε. With this notion in place, one can then define the ε-error randomized communication complexity of f, denoted $\mathrm{R}_\varepsilon(f)$, analogously to the deterministic one. By convention, this notation assumes private coins; the analogous public-coin variant is denoted $\mathrm{R}^{\mathrm{pub}}_\varepsilon(f)$. Further, when f has Boolean output, it is convenient to put $\mathrm{R}(f) = \mathrm{R}_{1/3}(f)$. Clearly, one always has $\mathrm{R}^{\mathrm{pub}}(f) \leq \mathrm{R}(f) \leq \mathrm{D}(f)$.

Consider a probability distribution μ on the input domain $\mathcal{X} \times \mathcal{Y}$. A protocol's error under μ is the probability that it errs when given a random input $(X, Y) \sim \mu$. The ε-error μ-*distributional complexity* of f, denoted $\mathrm{D}^\mu_\varepsilon(f)$, is then the minimum cost of a deterministic protocol for f whose error, under μ, is at most ε; an easy averaging argument shows that the restriction of determinism incurs no loss of generality. The fundamental minimax principle of Yao [22] says that $\mathrm{R}_\varepsilon(f) = \max_\mu \mathrm{D}^\mu_\varepsilon(f)$. In particular, exhibiting a lower bound on $\mathrm{D}^\mu_\varepsilon(f)$ for a wisely chosen μ lower bounds $\mathrm{R}_\varepsilon(f)$; this is a key lower-bounding technique in the area.

Let Π be a protocol that uses a public random string R as well as private random strings: R_A for Alice, R_B for Bob. Let $\Pi(x, y, R, R_A, R_B)$ denote the *transcript* of conversation between Alice and Bob, on input (x, y). The internal and external information costs of Π with respect to the distribution μ are then defined as follows:

$$\begin{aligned}\mathrm{icost}(\Pi) &= \mathrm{I}(\Pi(X, Y, R, R_A, R_B) : X \mid Y, R) \\ &\quad + \mathrm{I}(\Pi(X, Y, R, R_A, R_B) : Y \mid X, R), \\ \mathrm{icost}^{\mathrm{ext}}(\Pi) &= \mathrm{I}(\Pi(X, Y, R, R_A, R_B) : X, Y \mid R),\end{aligned}$$

where $(X, Y) \sim \mu$ and I denotes mutual information. These definitions capture the amount of *information* learned by each player about the other's input (in the internal case) and by an external observer about the total input (in the external case) when Π is run on a random μ-distributed input. It is elementary to show that these quantities lower bound the actual communication cost of the protocol. Therefore, the corresponding *information complexity* [3,9] measures – denoted $\mathrm{IC}^{\mu}_{\varepsilon}(f)$ and $\mathrm{IC}^{\mu,\mathrm{ext}}_{\varepsilon}(f)$ – defined as the infima of these costs over all worst-case ε-error protocols for f, naturally lower bound $\mathrm{R}_{\varepsilon}(f)$. This is another important lower-bounding technique.

Key Results

In a number of basic communication problems, Alice's and Bob's inputs are n-bit strings, denoted x and y, respectively, and the goal is to compute a Boolean function $f(x, y)$. We shall denote the ith bit of x as x_i. The bound $\mathrm{D}(f) \leq n+1$ is then trivial, because Alice can always just sent Bob her input, for a cost of n.

The textbook by Kushilevitz and Nisan [14] gives a thorough introduction to the subject and contains detailed proofs of several of the results summarized below. We first present results about specific communication problems, then move on to more abstract results about general problems, and close with a few applications of these results and ideas.

Problem-Specific Results

Equality Testing

This problem is defined by the *equality* function, given by $\mathrm{EQ}_n(x, y) = 1$ if $x = y$ and $\mathrm{EQ}_n(x, y) = 0$ otherwise. This can be solved nontrivially by a randomized protocol wherein Alice sends Bob a *fingerprint* of x, which Bob can compare with the corresponding fingerprint of y generated using the same random seed. Using public coins, a random n-bit string r can be used as a seed to generate the fingerprint $\langle x, r\rangle = \sum_{i=1}^{n} x_i r_i \bmod 2$. One can readily check that this yields $\mathrm{R}^{\mathrm{pub}}(\mathrm{EQ}_n) = O(1)$ and, more generally, $\mathrm{R}^{\mathrm{pub}}_{\varepsilon}(\mathrm{EQ}_n) = O(\log(1/\varepsilon))$. In the private coin setting, one can use a different kind of fingerprinting, e.g., by treating the bits of x as the coefficients of a degree-n polynomial and evaluating it at a random element of $\mathbb{F}_q$ (for a large enough prime q) to obtain a fingerprint. This idea leads to the bound $\mathrm{R}(\mathrm{EQ}_n) = O(\log n)$.

Randomization is essential for the above results: it can be shown that $\mathrm{D}(\mathrm{EQ}_n) \geq n$. The argument relies on the fundamental *rectangle property* of deterministic protocols, which states that the set of inputs (x, y) that lead to the same transcript must form a combinatorial rectangle inside the input space $\mathcal{X} \times \mathcal{Y}$. This can be proved by induction on the length of the transcript. This rectangle property then implies that if $u \neq v \in \{0, 1\}^n$, then a correct protocol for EQ_n cannot have the same transcript on inputs (u, u) and (v, v); otherwise, it would have the same transcript of (u, v) as well, and therefore err, because $\mathrm{EQ}_n(u, u) \neq \mathrm{EQ}_n(u, v)$. It follows that the protocol must have at least 2^n distinct transcripts, whence one of them must have length at least n.

It can also be shown that the upper bounds above are optimal [14]. The lower bound $\mathrm{R}(\mathrm{EQ}_n) = \Omega(\log n)$ is a consequence of the more general result that $\mathrm{D}(f) = 2^{O(\mathrm{R}(f))}$ for every Boolean function f. The lower bound $\mathrm{R}^{\mathrm{pub}}_{\varepsilon}(\mathrm{EQ}_n) = \Omega(\log(1/\varepsilon))$ follows from Yao's minimax principle and a version of the above rectangle-property argument.

More refined results can be obtained by considering the expected (rather than worst case) cost

of an r-round protocol, i.e., a protocol in which a total of r are sent: in this case, $\mathrm{R}_{\varepsilon}^{\mathrm{pub},(r)}(\mathrm{EQ}_n) = \Theta(\log\log\cdots\log(\min\{n, \log(1/\varepsilon)\}))$ with the outer chain of logarithms iterated $(r-1)$ times. This is tight [7]. Another, incomparable, result is that EQ_n has a zero-error randomized protocol with information cost only $O(1)$, regardless of the joint distribution from which the inputs are drawn [5].

Comparison

This problem is defined by the *greater-than* function, given by $\mathrm{GT}_n(x, y) = 1$ if $x > y$ and $\mathrm{GT}_n(x, y) = 0$ otherwise, where we treat x and y as integers written in binary. Like EQ_n, it has no nontrivial deterministic protocol, for much the same reason. As before, this implies $\mathrm{R}(\mathrm{GT}_n) = \Omega(\log n)$.

In fact the tight bound $\mathrm{R}(\mathrm{GT}_n) = \Theta(\log n)$ holds, but the proof requires a subtle argument. Binary search based on equality testing on substrings of x and y allows one to zoom in, in $O(\log n)$ rounds, on the most significant bit position where x differs from y. If each equality test is allowed $O(1/\log n)$ probability of error, a straightforward union bound gives an $O(1)$ overall error rate, but this uses $\Theta(\log\log n)$ communication per round. The improvement to an overall $O(\log n)$ bound is obtained by preceding each binary search step with an extra "sanity check" equality test on prefixes of x and y and backtracking to the previous level of the binary search if the check fails: this allows one to use only $O(1)$ communication per round.

The bounded-round complexity of GT_n is also fairly well understood. Replacing the binary search above with an $n^{1/r}$-ary search leads to the r-round bound $\mathrm{R}_{\varepsilon}^{(r)}(\mathrm{GT}_n) = O(n^{1/r}\log n)$. A lower bound of $\Omega(n^{1/r}/r^2)$ can be proven by carefully analyzing information cost.

Indexing and Bipartite Pointer Jumping

The *indexing* or *index* problem is defined by the Boolean function $\mathrm{INDEX}_n(x, k) = x_k$, where $x \in \{0,1\}^n$ as usual, but Bob's input $k \in [n]$, where $[n] = \{1, 2, \ldots, n\}$. Straightforward information-theoretic arguments show that the one-round complexity $\mathrm{R}^{(1)}(\mathrm{INDEX}_n) = \Omega(n)$, where the single message must go from Alice to Bob. Without this restriction, clearly $\mathrm{R}(\mathrm{INDEX}_n) = O(\log n)$. A more delicate result [17] is that in a $(1/3)$-error protocol for INDEX_n, for any $b \in [1, \log n]$, either Bob must send b bits or Alice must send $n/2^{O(b)}$ bits; an easy 2-round protocol shows that this trade-off is optimal. Even more delicate results, involving information cost, are known, and these are useful in certain applications (see below).

The indexing problem illustrates that interaction can improve communication cost exponentially. This can be generalized to show that $r+1$ rounds can be exponentially more powerful than r rounds. For this, one considers the *bipartite pointer jumping* problem, where Alice and Bob receive functions $f, g : [n] \to [n]$, respectively, and Bob also receives $y \in [n]$. Their goal is to compute $\mathrm{PJ}_{r,n}(f, g, y) = h_r(\cdots h_2(h_1(y))\cdots) \bmod 2$, where $h_i = f$ for odd i and $h_i = g$ for even i. Notice that $\mathrm{PJ}_{1,n}$ is essentially the same as INDEX_n and that $\mathrm{R}^{(r+1)}(\mathrm{PJ}_{r,n}) = O(r\log n)$. Suitably generalizing the information-theoretic arguments for INDEX_n shows that $\mathrm{R}^{(r)}(\mathrm{PJ}_{r,n}) = \Omega(n/r^2)$.

Inner Product Parity

The Boolean function $\mathrm{IP}_n(x, y) = \langle x, y\rangle$, which is the parity of the inner product $\sum_{i=1}^{n} x_i y_i$, is the most basic *very* hard communication problem: solving it to error $\frac{1}{2} - \delta$ (for constant δ) requires $n - O(\log(1/\delta))$ communication. This is proved by considering the distributional complexity $\mathrm{D}_{\varepsilon}^{\mu}(\mathrm{IP}_n)$, where μ is the uniform distribution and lower bounding it using the *discrepancy method*. Observe that a deterministic protocol Π with cost C induces a partition of the input domain into 2^C combinatorial rectangles, on each of which Π has the same transcript, hence the same output. If Π has error at most $\frac{1}{2} - \delta$ under μ, then the μ-discrepancies of these rectangles – i.e., the differences between the μ-measures of the 0s and 1s within them – must sum up to at least 2δ. Letting $\mathrm{disc}^{\mu}(f)$ denote the maximum over all rectangles R in $\mathcal{X}\times\mathcal{Y}$ of the μ- discrepancy of R, we then obtain $2^C\,\mathrm{disc}^{\mu}(f) \geq 2\delta$, which allows us to lower bound C if we can upper bound the discrepancy $\mathrm{disc}^{\mu}(f)$.

For the function IP, the matrix $(\text{IP}(x, y))_{x\in\{0,1\}^n, y\in\{0,1\}^n}$ is easily seen to be a Hadamard matrix, whose spectrum is well understood. With a bit of matrix analysis, this enables the discrepancy of IP under a uniform μ to be computed very accurately. This in turn yields the claimed communication complexity lower bound.

Set Disjointness

The problem of determining whether Alice's set $x \subseteq [n]$ is disjoint from Bob's set $y \subseteq [n]$, denoted $\text{DISJ}_n(x, y)$, is, along with its natural generalizations, the most studied and widely useful problem in communication complexity. It is easy to prove, from first principles, the strong lower bound $\text{D}(\text{DISJ}_n(x, y)) = n - o(n)$. Obtaining a similarly strong lower bound for randomized complexity turns out to be quite a challenge, one that has driven a number of theoretical innovations.

The discrepancy method outlined above is provably very weak at lower bounding $\text{R}(\text{DISJ}_n)$. Instead, one considers a refinement called the *corruption* technique: it consists of showing that "large" rectangles in the matrix for DISJ_n cannot come close to consisting purely of 1-inputs (i.e., disjoint pairs (x, y)) but must be corrupted by a "significant" fraction of 0-inputs. On the other hand, a sufficiently low-cost communication protocol for DISJ_n would imply that at least one such large rectangle must exist. The tension between these two facts gives rise to a lower bound on $\text{D}^{\mu}_{\varepsilon}(\text{DISJ}_n)$, where μ and ε figure in the quantification of "large" and "significant" above. Following this outline, Babai et al. [2] proved an $\Omega(\sqrt{n})$ lower bound using the uniform distribution. This was then improved by using a certain non-product input distribution, i.e., one where the inputs x and y are correlated – a provably necessary complication – to the optimal $\Omega(n)$, initially via a complicated Kolmogorov complexity technique, but eventually via elementary (though ingenious) combinatorics by Razborov [20]. Subsequently, Bar-Yossef et al. [4] re-proved the $\Omega(n)$ bound via a novel notion of *conditional information complexity*, a proof that has since been reworked to use the more natural internal information complexity [5].

Disjointness is also interesting and natural as a multiparty problem, where each of t players holds a subset of $[n]$ and they wish to determine if these are disjoint. An important result with many applications (see below) is that under a promise that the sets are pairwise disjoint except perhaps at one element, this requires $\Omega(n/t)$ communication, even if the players communicate via broadcast; this bound is essentially tight. Without this promise, and with only private message channels between the t players, disjointness requires $\Omega(tn)$ communication.

Gap Hamming Distance

This problem is defined by the partial Boolean function on $\{0,1\}^n \times \{0,1\}^n$ given by $\text{GHD}_n(x, y) = 0$ if $\|x - y\|_1 \leq \frac{1}{2}n - \sqrt{n}$; $\text{GHD}_n(x, y) = 1$ if $\|x - y\| \geq \frac{1}{2}n + \sqrt{n}$; and $\text{GHD}_n(x, y) = \star$ otherwise. Correctness and error probability for protocols for GHD_n are based only on inputs not mapped to $\star$. After several efforts giving special-case lower bounds, it was eventually proved [8] that $\text{R}(\text{GHD}_n) = \Omega(n)$ and in particular $\text{D}^{\mu}_{\varepsilon}(\text{GHD}_n) = \Omega(n)$ with μ being uniform and $\varepsilon = \Theta(1)$ being sufficiently small. This bound provably does not follow from the corruption method because of the presence of large barely-corrupted rectangles in the matrix for GHD_n; instead it was proved using the so-called *smooth corruption* technique [12].

General Complexity-Theoretic Results

There is a vast literature on general results connecting various notions of complexity for communication problems. As before, we survey some highlights. Throughout, we consider a general function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$.

Determinism vs. Public vs. Private Randomness

A private-coin protocol can be deterministically simulated by direct estimation of the probability of generating each possible transcript. This leads to the relation $\text{R}(f) = \Omega(\log \text{D}(f))$. This separation is the best possible, as witnessed by EQ_n. Further, a public-coin protocol can be restricted to draw its random string – no matter

how long – from a fixed set S consisting of $O(n)$ strings, for a constant additive increase in error probability [18]. This implies that it can be simulated using a private coin (which is used only to draw a random element of S) at an additional communication cost of $\log n + O(1)$. Therefore, $\mathrm{R}(f) \leq \mathrm{R}^{\mathrm{pub}}(f) + \log n + O(1)$. Again the EQ_n function shows that this separation is the best possible.

The Log-Rank Conjecture and Further Matrix Analysis

The rectangle property of communication protocols readily implies that $\mathrm{D}(f) \geq \log_2 \mathrm{rk}\, f$, where $\mathrm{rk}\, f$ is the rank of the matrix $(f(x,y))_{\{0,1\}^n \times \{0,1\}^n}$. It has long been conjectured that $\mathrm{D}(f) = \mathrm{poly}(\log \mathrm{rk}\, f)$. This famous conjecture remains wide open; the best-known relevant upper bound is $\mathrm{D}(f) \leq O(\sqrt{\mathrm{rk}\, f} \log \mathrm{rk}\, f)$ [16].

Other, more sophisticated, matrix analysis techniques can be used to establish lower bounds on $\mathrm{R}(f)$ by going through the approximate rank and factorization norms. The survey by Lee and Shraibman [15] has a strong focus on such techniques and provides a comprehensive coverage of results.

Direct Sum, Direct Product, and Amortization

Does the complexity of f grow n-fold, or even $\Omega(n)$-fold, if we have to compute f on n independent instances? This is called a direct sum question. Attempts to answer it and its variants, for general, as well as specific functions f have spurred a number of developments. Let f^k denote the (non-Boolean) function that, on input $((x^{(1)}, \ldots, x^{(k)}), (y^{(1)}, \ldots, y^{(k)}))$, outputs $(f(x^{(1)}, y^{(1)}), \ldots, f(x^{(k)}, y^{(k)}))$. Let $\mathrm{R}^k_\varepsilon(f^k)$ denote the cost of the best randomized protocol that computes each entry of the k-tuple of of values of f^k up to error ε. This is in contrast to the usual $\mathrm{R}_\varepsilon(f^k)$, which is concerned with getting the entire k-tuple correct except with probability ε. An alternate way of posing the direct sum question is to ask how the *amortized* randomized complexity $\overline{\mathrm{R}}_\varepsilon(f) = \lim_{k\to\infty} \mathrm{R}^k_\varepsilon(f^k)/k$ compares with $\mathrm{R}_\varepsilon(f)$.

An early result along these lines shows that $\overline{\mathrm{R}}(\mathrm{EQ}_n) = O(1)$ [10]. Recalling that $\mathrm{R}(\mathrm{EQ}_n) = \Omega(\log n)$, this shows that (in the private-coin setting) EQ_n exhibits economy of scale; we say that it does not satisfy the direct sum property. It had long been conjectured that no such economy of scale is possible in a public-coin setting; in fact information complexity rose to prominence as a technique precisely because in the *informational* setting, the direct sum property is easy to prove [9]. Thus, $\mathrm{IC}^\mu(f)$, for every distribution μ, lower bounds not just $\mathrm{R}(f)$ but also $\overline{\mathrm{R}}(f)$. More interestingly, the opposite inequality also holds, so that information and amortized randomized complexities are in fact equal; the proof uses a sophisticated *interactive protocol compression* technique [6] that should be seen as an analog of classical information theoretic results about single-message compression, e.g., via Huffman coding.

Thus, proving a general direct sum theorem for randomized communication is equivalent to compressing a protocol with information cost I down to $O(I)$ bits of communication. However, the best-known compression uses $2^{O(I)}$ bits [5], and this is optimal [11]. The proof of optimality – despite showing that a fully general direct sum theorem is impossible – is such that a slightly weakened direct sum result, such as $\mathrm{R}^k(f^k) = \Omega(k)\,\mathrm{R}(f) - O(\log n)$, remains possible and open. Meanwhile, fully general and strong direct sum theorems can be proven by restricting the model to bounded-round communication, or restricting the function to those whose complexity is captured by the smooth corruption bound.

Round Elimination

For problems that are hard only under a limitation on the number of rounds – e.g., GT_n, discussed above – strong bounded-round lower bounds are proved using the round elimination technique. Here, one shows that if an r-round protocol for f starts with a short enough first message, then this message can be eliminated altogether, and the resulting $(r-1)$-round protocol will solve a "subproblem" of f. Repeating this operation r times results in a 0-round protocol that, hopefully,

solves a nontrivial subproblem, giving a contradiction.

To make this useful, one then has to identify a reasonable notion of subproblem. It is especially useful to have this subproblem be a smaller instance of f itself. This does happen in several cases and can be illustrated by looking at GT_n: restricting n-length strings to indices in $[\ell, h]$ and forcing agreement at indices in $[1, \ell - 1]$ shows that GT_n contains $\mathrm{GT}_{h-\ell}$ as a subproblem. The proof of the aforementioned lower bound $\mathrm{R}^{(r)}(\mathrm{GT}_n) = \Omega(n^{1/r}/r^2)$ uses exactly this observation. The pointer jumping lower bound, also mentioned before, proceeds along similar lines.

Applications

Data Stream Algorithms

Consider a data stream algorithm using s bits of working memory and p passes over its input. Splitting the stream into two pieces, giving the first to Alice and the second to Bob, creates a communication problem and a $(2p - 1)$-round protocol for it, using s bits of communication per round. A generalization to multiplayer communication is immediate. These observations [1] allow us to infer several space (or pass/space trade-off) lower bounds for data stream algorithms from suitable communication lower bounds, often after a suitable reduction.

For instance, the problem of approximating the number of distinct items in a stream and its generalization to the problem of approximating frequency moments are almost fully understood, based on lower bounds for EQ_n, GHD_n, DISJ_n, and its generalization to multiple players, under the unique intersection restriction noted above. A large number of graph-theoretic problems can be shown to require $\Omega(n)$ space, n being the number of vertices, based on lower bounds for INDEX_n, PJ_n and variants, and again DISJ_n. For several other data streaming problems – e.g., approximating ℓ_∞ and cascaded norms and approximating a maximum cut or maximum matching – a reduction using an off-the-shelf communication lower bound is not known, but one can still obtain strong space lower bounds by considering a tailor-made communication problem in each case and applying the familiar lower-bounding techniques outlined above.

Data Structures

The cell-probe model of Yao is designed to capture all conceivable data structures on a modern computer: it models the query/update process as a sequence of probes into the entries (memory words) of a table containing the data structure. Focusing on static data structures for the moment, note that a t-probe algorithm using an s-word table with w-bit words directly leads to a $2t$-round communication protocol in which Alice (the querier) sends $(\log_2 s)$-bit messages and Bob (the table holder) sends w-bit messages. Lower bounds trading off t against w and s are therefore implied by suitable *asymmetric* communication lower bounds, where Alice's messages need to be much shorter than Bob's and Alice also has a correspondingly smaller input.

The study of these kinds of lower bounds was systematized by Miltersen et al. [17], who used round elimination as well as corruption-style techniques to obtain cell-probe lower bounds for set membership, predecessor search, range query, and further static data structure problems. Pătraşcu [19] derived an impressive number of cell-probe lower bounds – for problems ranging from union-find to dynamic stabbing and range reporting (in low dimension) to approximate near neighbor searching – by a tree of reductions starting from the *lopsided set disjointness* problem. This latter problem, denoted $\mathrm{LSD}_{k,n}$, gives Alice a set $x \subseteq [kn]$ with $|x| \leq k$ and Bob a set $y \subseteq [kn]$. Using information complexity techniques and a direct sum result for the basic INDEX problem, one can use the Alice/Bob trade-off result for INDEX discussed earlier to establish the nearly optimal trade-off that, for each $\delta > 0$, solving $\mathrm{LSD}_{k,n}$ to $\frac{1}{3}$ error (say) requires either Alice to send at least $\delta n \log k$ bits or Bob to send $nk^{1-O(\delta)}$ bits.

Circuit Complexity

Early work in circuit complexity had identified certain conjectured communication complexity lower bounds as a route towards strong lower bounds for circuit size and depth and related complexity measures for Boolean formulas and branching programs. Several of these conjectures remain unproven, especially ones involving the *number-on-the-forehead* (NOF) communication model, where the input is "written on the foreheads" of a large number, t, of players. The resulting high degree of input sharing allows for some rather novel nontrivial protocols, making lower bounds very hard to prove. Nevertheless, the discrepancy technique has been extended to NOF communication, and some of the technically hardest work in communication complexity has gone towards using it effectively for concrete problems, such as set disjointness [21]. While NOF lower bounds strong enough to imply circuit lower bounds remain elusive, certain other communication lower bounds, such as two-party bounds for computing *relations*, have had more success. In particular, monotone circuits for directed and undirected graph connectivity have been shown to require super-logarithmic depth, via the influential idea of Karchmer-Wigderson games [13].

Further Applications

We note in passing that there are plenty more applications of communication complexity than are possible to even outline in this short article. These touch upon such diverse areas as proof complexity; extension complexity for linear and semidefinite programming; AT^2 lower bounds in VLSI design; query complexity in the classical and quantum models; and time complexity on classical Turing machines. Kushilevitz and Nisan [14] remains the best starting point for further reading about these and more applications.

Recommended Reading

1. Alon N, Matias Y, Szegedy M (1999) The space complexity of approximating the frequency moments. J Comput Syst Sci 58(1):137–147. Preliminary version in Proceedings of the 28th annual ACM symposium on the theory of computing, 1996, pp 20–29
2. Babai L, Frankl P, Simon J (1986) Complexity classes in communication complexity theory. In: Proceedings of the 27th annual IEEE symposium on foundations of computer science, Toronto, pp 337–347
3. Barak B, Braverman M, Chen X, Rao A (2010) How to compress interactive communication. In: Proceedings of the 41st annual ACM symposium on the theory of computing, Cambridge, pp 67–76
4. Bar-Yossef Z, Jayram TS, Kumar R, Sivakumar D (2004) An information statistics approach to data stream and communication complexity. J Comput Syst Sci 68(4):702–732
5. Braverman M (2012) Interactive information complexity. In: Proceedings of the 44th annual ACM symposium on the theory of computing, New York, pp 505–524
6. Braverman M, Rao A (2011) Information equals amortized communication. In: Proceedings of the 52nd annual IEEE symposium on foundations of computer science, Palm Springs, pp 748–757
7. Brody J, Chakrabarti A, Kondapally R, Woodruff DP, Yaroslavtsev G (2014) Certifying equality with limited interaction. In: Proceedings of the 18th international workshop on randomization and approximation techniques in computer science, Barcelona, pp 545–581
8. Chakrabarti A, Regev O (2012) An optimal lower bound on the communication complexity of GAP-HAMMING-DISTANCE. SIAM J Comput 41(5):1299–1317. Preliminary version in Proceedings of the 43rd annual ACM symposium on the theory of computing, 2011, pp 51–60
9. Chakrabarti A, Shi Y, Wirth A, Yao AC (2001) Informational complexity and the direct sum problem for simultaneous message complexity. In: Proceedings of the 42nd annual IEEE symposium on foundations of computer science, Las Vegas, pp 270–278
10. Feder T, Kushilevitz E, Naor M, Nisan N (1995) Amortized communication complexity. SIAM J Comput 24(4):736–750. Preliminary version in Proceedings of the 32nd annual IEEE symposium on foundations of computer science, 1991, pp 239–248
11. Ganor A, Kol G, Raz R (2014) Exponential separation of information and communication for boolean functions. Technical report TR14-113, ECCC
12. Jain R, Klauck H (2010) The partition bound for classical communication complexity and query complexity. In: Proceedings of the 25th annual IEEE conference on computational complexity, Cambridge, pp 247–258
13. Karchmer M, Wigderson A (1990) Monotone circuits for connectivity require super-logarithmic depth. SIAM J Discrete Math 3(2):255–265. Preliminary version in Proceedings of the 20th annual ACM symposium on the theory of computing, 1988, pp 539–550
14. Kushilevitz E, Nisan N (1997) Communication complexity. Cambridge University Press, Cambridge

15. Lee T, Shraibman A (2009) Lower bounds in communication complexity. Found Trends Theor Comput Sci 3(4):263–399
16. Lovett S (2014) Communication is bounded by root of rank. In: Proceedings of the 46th annual ACM symposium on the theory of computing, New York, pp 842–846
17. Miltersen PB, Nisan N, Safra S, Wigderson A (1998) On data structures and asymmetric communication complexity. J Comput Syst Sci 57(1):37–49. Preliminary version in Proceedings of the 27th annual ACM symposium on the theory of computing, 1995, pp 103–111
18. Newman I (1991) Private vs. common random bits in communication complexity. Inf Process Lett 39(2):67–71
19. Pătraşcu M (2011) Unifying the landscape of cell-probe lower bounds. SIAM J Comput 40(3):827–847
20. Razborov A (1992) On the distributional complexity of disjointness. Theor Comput Sci 106(2):385–390. Preliminary version in Proceedings of the 17th international colloquium on automata, languages and programming, 1990, pp 249–253
21. Sherstov AA (2014) Communication lower bounds using directional derivatives. J. ACM 61:1–71. Preliminary version in Proceedings of the 45th annual ACM symposium on the theory of computing, 2013, pp 921–930
22. Yao AC (1977) Probabilistic computations: towards a unified measure of complexity. In: Proceedings of the 18th annual IEEE symposium on foundations of computer science, Providence, pp 222–227
23. Yao AC (1979) Some complexity questions related to distributive computing. In: Proceedings of the 11th annual ACM symposium on the theory of computing, Atlanta, pp 209–213

Communication in Ad Hoc Mobile Networks Using Random Walks

Ioannis Chatzigiannakis
Department of Computer Engineering and Informatics, University of Patras and Computer Technology Institute, Patras, Greece

Keywords

Data mules; Delay-tolerant networks; Disconnected ad hoc networks; Message ferrying; Message relays; Sink mobility

Years and Authors of Summarized Original Work

2003; Chatzigiannakis, Nikoletseas, Spirakis

Problem Definition

A mobile ad hoc network is a temporary dynamic interconnection network of wireless mobile nodes without any established infrastructure or centralized administration. A *basic communication problem*, in ad hoc mobile networks, is to send information from a *sender* node, A, to another designated *receiver* node, B. If mobile nodes A and B come within wireless range of each other, then they are able to communicate. However, if they do not, they can communicate if other network nodes of the network are willing to forward their packets. One way to solve this problem is the protocol of notifying every node that the sender A meets and provide it with *all the information* hoping that some of them will eventually meet the receiver B.

> Is there a more efficient technique (other than notifying every node that the sender meets, in the hope that some of them will then eventually meet the receiver) that will effectively solve the communication establishment problem without flooding the network and exhausting the battery and computational power of the nodes?

The problem of communication among mobile nodes is one of the most fundamental problems in ad hoc mobile networks and is at the core of many algorithms, such as for counting the number of nodes, electing a leader, data processing etc. For an exposition of several important problems in ad hoc mobile networks see [13]. The work of Chatzigiannakis, Nikoletseas and Spirakis [5] focuses on wireless mobile networks that are subject to highly dynamic structural changes created by mobility, channel fluctuations and device failures. These changes affect topological connectivity, occur with high frequency and may not be predictable in advance. Therefore, the environment where the nodes move (in three-dimensional space with possible obstacles) as

well as the motion that the nodes perform are *input* to any distributed algorithm.

The Motion Space

The space of possible motions of the mobile nodes is combinatorially abstracted by a *motion-graph*, i.e., the detailed geometric characteristics of the motion are neglected. Each mobile node is assumed to have a transmission range represented by a sphere *tr* centered by itself. Any other node inside *tr* can receive any message broadcast by this node. This sphere is approximated by a cube *tc* with volume $\mathrm{V}(tc)$, where $\mathrm{V}(tc) < \mathrm{V}(tr)$. The size of *tc* can be chosen in such a way that its volume $\mathrm{V}(tc)$ is the maximum that preserves $\mathrm{V}(tc) < \mathrm{V}(tr)$, and if a mobile node inside *tc* broadcasts a message, this message is received by any other node in *tc*. Given that the mobile nodes are moving in the space S, S is divided into consecutive cubes of volume $\mathrm{V}(tc)$.

Definition 1 The motion graph $G(V, E)$, $(|V| = n, |E| = m)$, which corresponds to a quantization of S is constructed in the following way: a vertex $u \in G$ represents a cube of volume $\mathrm{V}(tc)$ and an edge $(u, v) \in G$ exists if the corresponding cubes are adjacent.

The number of vertices n, actually approximates the ratio between the volume V(S) of space S, and the space occupied by the transmission range of a mobile node $\mathrm{V}(tr)$. In the extreme case where $\mathrm{V(S)} \approx \mathrm{V}(tr)$, the transmission range of the nodes approximates the space where they are moving and $n = 1$. Given the transmission range *tr*, n depends linearly on the volume of space S regardless of the choice of *tc*, and $n = O(V(\mathrm{S})/V(tr))$. The ratio $V(\mathrm{S})/V(tr)$ is the *relative motion space size* and is denoted by ρ. Since the edges of G represent neighboring polyhedra each vertex is connected with a constant number of neighbors, which yields that $m = \Theta(n)$. In this example where *tc* is a cube, G has maximum degree of six and $m \leq 6n$. Thus *motion graph* G is (usually) a *bounded degree graph* as it is derived from a regular graph of small degree by deleting parts of it corresponding to motion or communication obstacles. Let Δ be the maximum vertex degree of G.

The Motion of the Nodes-Adversaries

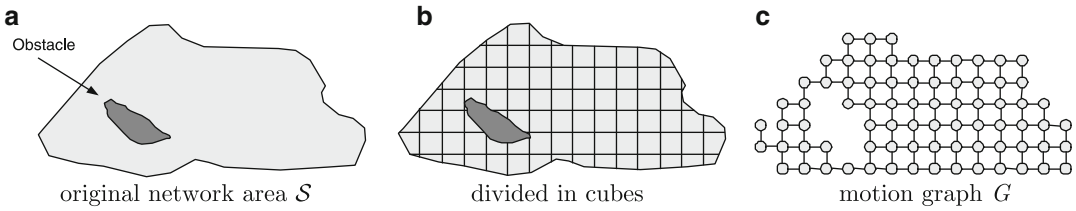
In the general case, the motions of the nodes are decided by an *oblivious adversary*: The adversary determines motion patterns in any possible way but independently of the distributed algorithm. In other words, the case where some of the nodes are deliberately trying to *maliciously affect* the protocol, e.g., avoid certain nodes, are excluded. This is a pragmatic assumption usually followed by applications. Such kind of motion adversaries are called *restricted motion adversaries*.

For purposes of studying efficiency of distributed algorithms for ad hoc networks *on the average*, the motions of the nodes are modeled by *concurrent and independent random walks*. The assumption that the mobile nodes move randomly, either according to uniformly distributed changes in their directions and velocities or according to the random waypoint mobility model by picking random destinations, has been used extensively by other research.

Key Results

The key idea is to take advantage of the mobile nodes natural movement by exchanging information whenever mobile nodes meet incidentally. It is evident, however, that if the nodes are spread in remote areas and they do not move beyond these areas, there is no way for information to reach them, unless the protocol takes special care of such situations. The work of Chatzigiannakis, Nikoletseas and Spirakis [5] proposes the idea of forcing only a small subset of the deployed nodes to move as per the needs of the protocol; they call this subset of nodes the *support* of the network. Assuming the availability of such nodes, they are used to provide a simple, correct and efficient strategy for communication between any pair of nodes of the network that avoids message flooding.

Let k nodes be a predefined set of nodes that become the nodes of the support. These



Communication in Ad Hoc Mobile Networks Using Random Walks, Fig. 1 The original network area S (**a**), how it is divided in consecutive cubes of volume $V(tc)$ (**b**) and the resulting motion graph G (**c**)

nodes move randomly and fast enough so that they visit in sufficiently short time the entire motion graph. When some node of the support is within transmission range of a sender, it notifies the sender that it may send its message(s). The messages are then stored "somewhere within the support structure". When a receiver comes within transmission range of a node of the support, the receiver is notified that a message is "waiting" for him and the message is then forwarded to the receiver.

Protocol 1 (The "Snake" Support Motion Coordination Protocol) Let $S_0, S_1, \ldots, S_{k-1}$ be the members of the support and let S_0 denote the leader node (possibly elected). The protocol forces S_0 to perform a random walk on the motion graph and each of the other nodes S_i execute the simple protocol "move where S_{i-1} was before". When S_0 is about to move, it sends a message to S_1 that states the new direction of movement. S_1 will change its direction as per instructions of S_0 and will propagate the message to S_2. In analogy, S_i will follow the orders of S_{i-1} after transmitting the new directions to S_{i+1}. Movement orders received by S_i are positioned in a queue Q_i for sequential processing. The very first move of S_i, $\forall i \in \{1, 2, \ldots, k-1\}$ is delayed by a δ period of time.

The purpose of the random walk of the head S_0 is to ensure a *cover*, within some finite time, of the whole graph G without knowledge and memory, other than local, of topology details. This memoryless motion also ensures fairness, low-overhead and inherent robustness to structural changes.

Consider the case where any sender or receiver is allowed a general, unknown motion strategy, but its strategy is provided by a restricted motion adversary. This means that each node not in the support either (a) executes a deterministic motion which either stops at a vertex or cycles forever after some initial part or (b) it executes a stochastic strategy which however is *independent* of the motion of the support. The authors in [5] prove the following correctness and efficiency results. The reader can refer to the excellent book by Aldous and Fill [1] for a nice introduction on Makrov Chains and Random Walks.

Theorem 1 *The support and the "snake" motion coordination protocol guarantee reliable communication between any sender-receiver (A, B) pair in finite time, whose expected value is bounded only by a function of the relative motion space size ρ and does not depend on the number of nodes, and is also independent of how MH_S, MH_R move, provided that the mobile nodes not in the support do not deliberately try to avoid the support.*

Theorem 2 *The expected communication time of the support and the "snake" motion coordination protocol is bounded above by $\Theta(\sqrt{mc})$ when the (optimal) support size $k = \sqrt{2mc}$ and c is $e/(e-1)u$, u being the "separation threshold time" of the random walk on G.*

Theorem 3 *By having the support's head move on a regular spanning subgraph of G, there is an absolute constant $\gamma > 0$ such that the expected meeting time of A (or B) and the support is bounded above by $\gamma n^2/k$. Thus the protocol*

guarantees a total expected communication time of $\Theta(\rho)$, independent of the total number of mobile nodes, and their movement.

The analysis assumes that the head S_0 moves according to a continuous time random walk of total rate 1 (rate of exit out of a node of G). If S_0 moves ψ *times faster* than the rest of the nodes, all the estimated times, except the inter-support time, will be divided by ψ. Thus the expected total communication time can be made to be as small as $\Theta(\gamma\rho/\sqrt{\psi})$ where γ is an absolute constant. In cases where S_0 can take advantage of the network topology, all the estimated times, except the inter-support time are improved:

Theorem 4 *When the support's head moves on a regular spanning subgraph of G the expected meeting time of A (or B) and the support cannot be less than $(n-1)^2/2m$. Since $m = \Theta(n)$, the lower bound for the expected communication time is $\Theta(n)$. In this sense, the "snake" protocol's expected communication time is optimal, for a support size which is $\Theta(n)$.*

The "on-the-average" analysis of the time-efficiency of the protocol assumes that the motion of the mobile nodes not in the support *is a random walk on the motion graph G*. The random walk of each mobile node is performed independently of the other nodes.

Theorem 5 *The expected communication time of the support and the "snake" motion coordination protocol is bounded above by the formula*

$$E(T) \leq \frac{2}{\lambda_2(G)}\Theta\left(\frac{n}{k}\right) + \Theta(k)\,.$$

The upper bound is minimized when $k = \sqrt{2n/\lambda_2(G)}$, where λ_2 is the second eigenvalue of the motion graph's adjacency matrix.

The way the support nodes move and communicate is robust, in the sense that it can tolerate failures of the support nodes. The types of failures of nodes considered are permanent, i.e., stop failures. Once such a fault happens, the support node of the fault does not participate in the ad hoc mobile network anymore. A communication protocol is β-*faults tolerant*, if it still allows the members of the network to communicate correctly, under the presence of at most β permanent faults of the nodes in the support ($\beta \geq 1$). [5] shows that:

Theorem 6 *The support and the "snake" motion coordination protocol is 1-fault tolerant.*

Applications

Ad hoc mobile networks are rapidly deployable and self-configuring networks that have important applications in many critical areas such as disaster relief, ambient intelligence, wide area sensing and surveillance. The ability to network *anywhere*, *anytime* enables teleconferencing, home networking, sensor networks, personal area networks, and embedded computing applications [13].

Related Work

The most common way to establish communication is to form paths of intermediate nodes that lie within one another's transmission range and can directly communicate with each other. The mobile nodes act as hosts and routers at the same time in order to propagate packets along these paths. This approach of maintaining a global structure with respect to the temporary network is a difficult problem. Since nodes are moving, the underlying communication graph is changing, and the nodes have to adapt quickly to such changes and reestablish their routes. Busch and Tirthapura [2] provide the first analysis of the performance of some characteristic protocols [8, 13] and show that in some cases they require $\Omega(u^2)$ time, where u is the number of nodes, to stabilize, i.e., be able to provide communication.

The work of Chatzigiannakis, Nikoletseas and Spirakis [5] focuses on networks where topological connectivity is subject to frequent, unpredictable change and studies the problem of efficient data delivery in sparse networks where network partitions can last for a significant period of time. In such cases, it is possible to

have a small team of fast moving and versatile vehicles, to implement the support. These vehicles can be cars, motorcycles, helicopters or a collection of independently controlled mobile modules, i.e., robots. This specific approach is inspired by the work of Walter, Welch and Amato [14] that study the problem of motion coordination in distributed systems consisting of such robots, which can connect, disconnect and move around.

The use of mobility to improve performance in ad hoc mobile networks has been considered in different contexts in [6, 9, 11, 15]. The primary objective has been to provide intermittent connectivity in a disconnected ad hoc network. Each solution achieves certain properties of end-to-end connectivity, such as delay and message loss among the nodes of the network. Some of them require long-range wireless transmission, other require that all nodes move pro-actively under the control of the protocol and collaborate so that they meet more often. The *key idea* of forcing only a subset of the nodes to facilitate communication is used in a similar way in [10, 15]. However, [15] focuses in cases where only one node is available. Recently, the application of mobility to the domain of wireless sensor networks has been addressed in [3, 10, 12].

Open Problems

A number of problems related to the work of Chatzigiannakis, Nikoletseas and Spirakis [5] remain open. It is clear that the size of the support, k, the shape and the way the support moves affects the performance of end-to-end connectivity. An open issue is to investigate alternative structures for the support, different motion coordination strategies and comparatively study the corresponding effects on communication times. To this end, the support idea is extended to hierarchical and highly changing motion graphs in [4]. The idea of cooperative routing based on the existence of support nodes may also improve security and trust.

An important issue for the case where the network is sparsely populated or where the rate of motion is too high is to study the performance of path construction and maintenance protocols. Some work has be done in this direction in [2] that can be also used to investigate the end-to-end communication in wireless sensor networks. It is still unknown if there exist impossibility results for distributed algorithms that attempt to maintain structural information of the implied fragile network of virtual links.

Another open research area is to analyze the properties of end-to-end communication given certain support motion strategies. There are cases where the mobile nodes interactions may behave in a similar way to the Physics paradigm of *interacting particles* and their modeling. Studies of interaction times and propagation times in various graphs are reported in [7] and are still important to further research in this direction.

Experimental Results

In [5] an experimental evaluation is conducted via simulation in order to model the different possible situations regarding the geographical area covered by an ad-hoc mobile network. A number of experiments were carried out for grid-graphs (2D, 3D), random graphs ($G_{n,p}$ model), bipartite multi-stage graphs and two-level motion graphs. All results verify the theoretical analysis and provide useful insight on how to further exploit the support idea. In [4] the model of hierarchical and highly changing ad-hoc networks is investigated. The experiments indicate that, the pattern of the "snake" algorithm's performance remains the same even in such type of networks.

URL to Code

http://ru1.cti.gr

Cross-References

▸ Mobile Agents and Exploration

Recommended Reading

1. Aldous D, Fill J (1999) Reversible markov chains and random walks on graphs. http://stat-www.berkeley.edu/users/aldous/book.html. Accessed 1999
2. Busch C, Tirthapura S (2005) Analysis of link reversal routing algorithms. SIAM J Comput 35(2):305–326
3. Chatzigiannakis I, Kinalis A, Nikoletseas S (2006) Sink mobility protocols for data collection in wireless sensor networks. In: Zomaya AY, Bononi L (eds) 4th international mobility and wireless access workshop (MOBIWAC 2006), Terromolinos, pp 52–59
4. Chatzigiannakis I, Nikoletseas S (2004) Design and analysis of an efficient communication strategy for hierarchical and highly changing ad-hoc mobile networks. J Mobile Netw Appl 9(4):319–332. Special issue on parallel processing issues in mobile computing
5. Chatzigiannakis I, Nikoletseas S, Spirakis P (2003) Distributed communication algorithms for ad hoc mobile networks. J Parallel Distrib Comput 63(1):58–74. Special issue on wireless and mobile ad-hoc networking and computing, edited by Boukerche A
6. Diggavi SN, Grossglauser M, Tse DNC (2005) Even one-dimensional mobility increases the capacity of wireless networks. IEEE Trans Inf Theory 51(11):3947–3954
7. Dimitriou T, Nikoletseas SE, Spirakis PG (2004) Analysis of the information propagation time among mobile hosts. In: Nikolaidis I, Barbeau M, Kranakis E (eds) 3rd international conference on ad-hoc, mobile, and wireless networks (ADHOCNOW 2004). Lecture notes in computer science (LNCS), vol 3158. Springer, Berlin, pp 122–134
8. Gafni E, Bertsekas DP (1981) Distributed algorithms for generating loop-free routes in networks with frequently changing topology. IEEE Trans Commun 29(1):11–18
9. Grossglauser M, Tse DNC (2002) Mobility increases the capacity of ad hoc wireless networks. IEEE/ACM Trans Netw 10(4):477–486
10. Jain S, Shah R, Brunette W, Borriello G, Roy S (2006) Exploiting mobility for energy efficient data collection in wireless sensor networks. J Mobile Netw Appl 11(3):327–339
11. Li Q, Rus D (2003) Communication in disconnected ad hoc networks using message relay. J Parallel Distrib Comput 63(1):75–86. Special issue on wireless and mobile ad-hoc networking and computing, edited by A Boukerche
12. Luo J, Panchard J, Piórkowski M, Grossglauser M, Hubaux JP (2006) Mobiroute: routing towards a mobile sink for improving lifetime in sensor networks. In: Gibbons PB, Abdelzaher T, Aspnes J, Rao R (eds) 2nd IEEE/ACM international conference on distributed computing in sensor systems (DCOSS 2005). Lecture notes in computer science (LNCS), vol 4026. Springer, Berlin, pp 480–497
13. Perkins CE (2001) Ad hoc networking. Addison-Wesley, Boston
14. Walter JE, Welch JL, Amato NM (2004) Distributed reconfiguration of metamorphic robot chains. J Distrib Comput 17(2):171–189
15. Zhao W, Ammar M, Zegura E (2004) A message ferrying approach for data delivery in sparse mobile ad hoc networks. In: Murai J, Perkins C, Tassiulas L (eds) 5th ACM international symposium on mobile ad hoc networking and computing (MobiHoc 2004). ACM, Roppongi Hills, pp 187–198

Compact Routing Schemes

Shiri Chechik
Department of Computer Science, Tel Aviv University, Tel Aviv, Israel

Keywords

Approximate shortest paths; Compact routing; Stretch factor

Years and Authors of Summarized Original Work

2013; Chechik

Problem Definition

Routing is a distributed mechanism that allows sending messages between any pair of nodes of the network. As in all distributed algorithms, a routing scheme runs locally on every processor/node of the network. Each node/processor of the network has a routing daemon running on it whose responsibility is to forward arriving messages while utilizing local information that is stored at the node itself. This local information is usually referred to as the routing table of the node.

A routing scheme involves two phases, a preprocessing phase and a routing phase. In the preprocessing phase, the algorithm assigns every node of the network a routing table and a small-size label. The label is used as the address of the node and therefore is usually expected to be of

small size – poly-logarithmic in the size of the network.

In the routing phase, some node of the networks wishes to send a message to some other nodes of the network in a distributed manner. During the routing phase, each node of the network may receive this message, and it has to decide whether this message reached its final destination, and if not, the node needs to decide to which of its neighbors this message should be forwarded next. In order to make these decisions, the node may use its own routing table and the header of the message that usually contains the label of the final destination and perhaps some additional information.

The stretch of a routing scheme is defined as the worst case ratio between the length of the path obtained by the routing scheme and the length of the shortest path between the source and the destination. There are two main objectives in designing the routing scheme. The first is to minimize the stretch of the routing scheme, and the second is to minimize the size of the routing tables. Much of the work on designing routing schemes focuses on the trade-off between these two objectives.

One extreme case is when it is allowed to use linear-size routing tables. In this case, one can store a complete routing table at all nodes, i.e., for every source node s and every potential destination node t, store at s the port number of the neighbor of s on the shortest path from s to t. In this case, the stretch is 1, i.e., the algorithm can route on exact shortest paths. However, a clear drawback is that the size of the routing tables is large, linear in the size of the network.

One may wish to use smaller routing tables at the price of a larger stretch. A routing scheme is considered to be compact if the size of the routing tables is sublinear in the number of nodes.

Key Results

This section presents a survey on compact routing schemes and a highlight of some recent new developments.

Many papers focus on the trade-off between the size of the routing tables and the stretch (e.g., [1, 2, 4, 5, 7–9]). The first trade-off was obtained by Peleg and Upfal [9]. Their scheme considered unweighted graph and achieved a bound on the total size of the routing tables.

Later, Awerbuch et al. [1] considered weighted graphs and achieved a routing scheme with a guarantee on the maximum table size. Their routing scheme uses table size of $\tilde{O}(n^{1/k})$ and was with $O(k^2 9^k)$ stretch. A better trade-off was later achieved by Awerbuch and Peleg [2].

Until very recently, the best-known trade-off was due to Thorup and Zwick [10]. They presented a routing scheme that uses routing tables of size $\tilde{O}(n^{1/k})$, a stretch of $4k - 5$, and label size of $O(k \log^2 n)$. Moreover, they showed that if a handshaking is allowed, namely, if the source node and the destination are allowed to exchange an information of size $O(\log^2 n)$ bits, then the stretch can be improved to $2k - 1$. Clearly, in many cases, it would be desirable to avoid the use of handshaking, as the overhead of establishing a handshake can be as high as sending the original message itself.

A natural question is, what is the best trade-off between routing table size and stretch one can hope for with or without a handshake? In fact, assuming the girth conjecture of Erdős [6], one can show that with table size of $O(n^{1/k})$, the best stretch possible is $2k - 1$ with or without a handshake. Hence, in the case of a handshake, Thorup and Zwick's scheme [10] is essentially optimal. However, in the case of no handshake, there is still a gap between the lower and upper bound. A main open problem in the area of compact routing schemes is on the gap between the stretch $4k - 5$ and $2k - 1$.

Recently, Chechik [3] gave the first evidence that the asymptotically optimal stretch is less than $4k$. Chechik [3] presented the first improvement to the stretch-space trade-off of compact routing scheme since the result of Thorup and Zwick [10]. More specifically, [3] presented a compact routing scheme for weighted general undirected graphs that uses tables of size $\tilde{O}(n^{1/k})$ and has stretch $c \cdot k$ for some absolute constant $c < 4$.

Open Problems

The main question that still remains unresolved is to prove or disprove the existence of a compact routing scheme that utilizes tables of size $\tilde{O}\left(n^{1/k}\right)$ and has stretch of $2k$ without the use of a handshake.

Recommended Reading

1. Awerbuch B, Bar-Noy A, Linial N, Peleg D (1990) Improved routing strategies with succinct tables. J Algorithms 11(3):307–341
2. Awerbuch B, Peleg D (1990) Sparse partitions. In: Proceedings of 31st IEEE symposium on foundations of computer science (FOCS), St. Louis, pp 503–513
3. Chechik S (2013) Compact routing schemes with improved stretch. In: 32nd ACM symposium on principles of distributed computing (PODC), Montreal, pp 33–41
4. Cowen LJ (2001) Compact routing with minimum stretch. J Algorithms 38:170–183
5. Eilam T, Gavoille C, Peleg D (2003) Compact routing schemes with low stretch factor. J Algorithms 46:97–114
6. Erdős P (1964) Extremal problems in graph theory. In: Theory of graphs and its applications. Methuen, London, pp 29–36
7. Gavoille C, Peleg D (2003) Compact and localized distributed data structures. Distrib Comput 16:111–120
8. Peleg D (2000) Distributed computing: a locality-sensitive approach. SIAM, Philadelphia
9. Peleg D, Upfal E (1989) A trade-off between space and efficiency for routing tables. J ACM 36(3):510–530
10. Thorup M, Zwick U (2001) Compact routing schemes. In: Proceedings of 13th ACM symposium on parallel algorithms and architectures (SPAA), Heraklion, pp 1–10

Competitive Auction

Tian-Ming Bu
Software Engineering Institute, East China Normal University, Shanghai, China

Keywords

Auction design; Optimal mechanism design

Years and Authors of Summarized Original Work

2001; Goldberg, Hartline, Wright
2002; Fiat, Goldberg, Hartline, Karlin

Problem Definition

This problem studies the *one round*, *sealed-bid* auction model where an auctioneer would like to sell an idiosyncratic commodity with unlimited copies to n bidders and each bidder $i \in \{1, \ldots, n\}$ will get at most one item.

First, for any i, bidder i bids a value b_i representing the price he is willing to pay for the item. They submit the bids simultaneously. After receiving the bidding vector $\mathbf{b} = (b_1, \ldots, b_n)$, the auctioneer computes and outputs the allocation vector $\mathbf{x} = (x_1, \ldots, x_n) \in \{0, 1\}^n$ and the price vector $\mathbf{p} = (p_1, \ldots, p_n)$. If for any i, $x_i = 1$, then bidder i gets the item and pays p_i for it. Otherwise, bidder i loses and pays nothing. In the auction, the auctioneer's revenue is $\sum_{i=1}^{n} \mathbf{x}\mathbf{p}^T$.

Definition 1 (Optimal Single Price Omniscient Auction $\mathcal{F}$) Given a bidding vector $\mathbf{b}$ sorted in decreasing order,

$$\mathcal{F}(\mathbf{b}) = \max_{1 \le i \le n} i \cdot b_i$$

Further,

$$\mathcal{F}^{(m)}(\mathbf{b}) = \max_{m \le i \le n} i \cdot b_i$$

Obviously, $\mathcal{F}$ maximizes the auctioneer's revenue if only uniform price is allowed.

However, in this problem, each bidder i is associated with a private value v_i representing the item's value in his opinion. So if bidder i gets the item, his payoff should be $v_i - p_i$. Otherwise, his payoff is 0. So for any bidder i, his payoff function can be formulated as $(v_i - p_i)x_i$. Furthermore, free will is allowed in the model. In other words, each bidder would bid some b_i different from his true value v_i, to maximize his payoff.

The objective of the problem is to design a *truthful* auction which could still maximize the auctioneer's revenue. An auction is *truthful* if for every bidder i, bidding his true value would maximize his payoff, regardless of the bids submitted by the other bidders [12, 13].

Definition 2 (Competitive Auctions)

INPUT: the submitted bidding vector $\mathbf{b}$.
OUTPUT: the allocation vector $\mathbf{x}$ and the price vector $\mathbf{p}$.
CONSTRAINTS:

(a) Truthful;
(b) The auctioneer's revenue is within a constant factor of the optimal single pricing for all inputs.

Key Results

Let $\mathbf{b}_{-i} = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n)$. f is any function from $\mathbf{b}_{-i}$ to the price.

Algorithm 1 Bid-independent auction: $\mathcal{A}_f(\mathbf{b})$

1: **for** $i = 1$ **to** n **do**
2: **if** $f(\mathbf{b}_{-i}) \le b_i$ **then**
3: $x_i = 1$ and $p_i = f(\mathbf{b}_i)$
4: **else**
5: $x_i = 0$
6: **end if**
7: **end for**

Theorem 1 ([6]) *An auction is truthful if and only if it is equivalent to a bid-independent auction.*

Definition 3 A truthful auction $\mathcal{A}$ is β-competitive against $\mathcal{F}^{(m)}$ if for all bidding vectors $\mathbf{b}$, the expected profit of $\mathcal{A}$ on $\mathbf{b}$ satisfies

$$\mathbf{E}(\mathcal{A}(\mathbf{b})) \ge \frac{\mathcal{F}^{(m)}(\mathbf{b})}{\beta}$$

Definition 4 (CostShare$_C$ [11]) Given bids $\mathbf{b}$, this mechanism finds the largest k such that the highest k bidders' biddings are at least $\frac{C}{k}$. Charge each of such k bidders $\frac{C}{k}$.

Algorithm 2 Sampling cost-sharing auction (SCS)

1: Partition bidding vector $\mathbf{b}$ uniformly at random into two sets $\mathbf{b}'$ and $\mathbf{b}''$.
2: Computer $\mathcal{F}' = \mathcal{F}(\mathbf{b}')$ and $\mathcal{F}'' = \mathcal{F}(\mathbf{b}'')$.
3: Running CostShare$_{\mathcal{F}''}$ on $\mathbf{b}'$ and CostShare$_{\mathcal{F}'}$ on $\mathbf{b}''$.

Theorem 2 ([6]) *SCS is 4-competitive against $\mathcal{F}^{(2)}$, and the bound is tight.*

SCS could be extended for partitioning into k parts for any k. In fact, $k = 3$ is the optimal partition.

Theorem 3 ([10]) *The random three partitioning cost sharing auction is 3.25-competitive.*

Theorem 4 ([9]) *Let $\mathcal{A}$ be any truthful randomized auction. There exists an input bidding vector $\mathbf{b}$ on which $\boldsymbol{E}(\mathcal{A}(\mathbf{b})) \le \frac{\mathcal{F}^{(2)}(\mathbf{b})}{2.42}$.*

Applications

As the Internet becomes more popular, more and more auctions are beginning to appear. Further, the items on sale in the auctions vary from antiques, paintings, and digital goods, for example, mp3, licenses, network resources, and so on. Truthful auctions can reduce the bidders' cost of investigating the competitors' strategies, since truthful auctions encourage bidders to bid their true values. On the other hand, competitive auctions can also guarantee the auctioneer's profit. So this problem is very practical and significant. These years, designing and analyzing competitive auctions under various auction models has become a hot topic [1–5, 7, 8].

Cross-References

▸ CPU Time Pricing
▸ Multiple Unit Auctions with Budget Constraint

Recommended Reading

1. Abrams Z (2006) Revenue maximization when bidders have budgets. In: Proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithms (SODA-06), Miami, pp 1074–1082
2. Bar-Yossef Z, Hildrum K, Wu F (2002) Incentive-compatible online auctions for digital goods. In: Proceedings of the 13th annual ACM-SIAM symposium on discrete mathematics (SODA-02), New York, pp 964–970
3. Borgs C, Chayes JT, Immorlica N, Mahdian M, Saberi A (2005) Multi-unit auctions with budget-constrained bidders. In: ACM conference on electronic commerce (EC-05), Vancouver, pp 44–51
4. Bu TM, Qi Q, Sun AW (2008) Unconditional competitive auctions with copy and budget constraints. Theor Comput Sci 393(1–3):1–13
5. Deshmukh K, Goldberg AV, Hartline JD, Karlin AR (2002) Truthful and competitive double auctions. In: Möhring RH, Raman R (eds) Algorithms – ESA 2002, 10th annual European symposium, Rome. Lecture notes in computer science, vol 2461. Springer, pp 361–373
6. Fiat A, Goldberg AV, Hartline JD, Karlin AR (2002) Competitive generalized auctions. In: Proceedings of the 34th annual ACM symposium on theory of computing (STOC-02), New York, pp 72–81
7. Goldberg AV, Hartline JD (2001) Competitive auctions for multiple digital goods. In: auf der Heide FM (ed) Algorithms – ESA 2001, 9th annual european symposium, Aarhus. Lecture notes in computer science, vol 2161. Springer, pp 416–427
8. Goldberg AV, Hartline JD (2003) Envy-free auctions for digital goods. In: Proceedings of the 4th ACM conference on electronic commerce (EC-03), New York, pp 29–35
9. Goldberg AV, Hartline JD, Wright A (2001) Competitive auctions and digital goods. In: Proceedings of the twelfth annual ACM-SIAM symposium on discrete algorithms (SODA-01), New York, pp 735–744
10. Hartline JD, McGrew R (2005) From optimal limited to unlimited supply auctions. In: Proceedings of the 6th ACM conference on electronic commerce (EC-05), Vancouver, pp 175–182
11. Moulin H (1999) Incremental cost sharing: characterization by coalition strategy-proofness. Soc Choice Welf 16:279–320
12. Nisan N, Ronen A (1999) Algorithmic mechanism design. In: Proceedings of the thirty-first annual ACM symposium on theory of computing (STOC-99), New York, pp 129–140
13. Parkes DC (2004) Chapter 2: iterative combinatorial auctions. PhD thesis, University of Pennsylvania

Complexity Dichotomies for Counting Graph Homomorphisms

Jin-Yi Cai[1,2], Xi Chen[3,4], and Pinyan Lu[5]
[1]Beijing University, Beijing, China
[2]Computer Sciences Department, University of Wisconsin–Madison, Madison, WI, USA
[3]Computer Science Department, Columbia University, New York, NY, USA
[4]Computer Science and Technology, Tsinghua University, Beijing, China
[5]Microsoft Research Asia, Shanghai, China

Keywords

Computational complexity; Counting complexity; Graph homomorphisms; Partition functions

Years and Authors of Summarized Original Work

2000; Dyer, Greenhill
2005; Bulatov, Grohe
2010; Goldberg, Grohe, Jerrum, Thurley
2013; Cai, Chen, Lu

Problem Definition

It is well known that if NP $\neq$ P, there is an infinite hierarchy of complexity classes between them [10]. However, for some broad classes of problems, a *complexity dichotomy* exists: every problem in the class is either in polynomial time or NP-hard. Such results include Schaefer's theorem [13], the dichotomy of Hell and Nešetřil for H-coloring [9], and some subclasses of the general constraint satisfaction problem [4]. These developments lead to the following questions: How far can we push the envelope and show dichotomies for even broader classes of problems? Given a class of problems, what is the criterion that distinguishes the tractable problems from the intractable ones? How does it help in solving the tractable problems efficiently? Now replacing NP

with #P [15], all the questions above can be asked for *counting* problems.

One family of counting problem concerns graph homomorphisms. Given two undirected graphs G and H, a graph homomorphism from G to H is a map ξ from the vertex set $V(G)$ to $V(H)$ such that (u, v) is an edge in G if and only if $(\xi(u), \xi(v))$ is an edge in H. The counting problem for graph homomorphism is to compute the number of homomorphisms from G to H. For a fixed graph H, this problem is also known as the $\#H$-coloring problem. In addition to $\#H$-coloring, a more general family of problems that has been studied intensively over the years is to count graph homomorphisms with weights. Formally, we use $\mathbf{A}$ to denote an $m \times m$ symmetric matrix with entries $(A_{i,j})$, $i, j \in [m] = \{1, \ldots, m\}$. Given any undirected graph $G = (V, E)$, we define the graph homomorphism function $Z_{\mathbf{A}}(G)$ as follows:

$$Z_{\mathbf{A}}(G) = \sum_{\xi: V \to [m]} \prod_{(u,v) \in E} A_{\xi(u),\xi(v)}. \quad (1)$$

This is also called the *partition function* from statistical physics. It is clear from the definition that $Z_{\mathbf{A}}(G)$ is exactly the number of homomorphisms from G to H, when $\mathbf{A}$ is the $\{0, 1\}$ adjacency matrix of H.

Graph homomorphism can express many natural graph properties. For example, if one takes H to be the graph over two vertices $\{0, 1\}$ with an edge $(0, 1)$ and a self-loop at 1, then the set of vertices mapped to 1 in a graph homomorphism from G to H corresponds to a VERTEX COVER of G, and the counting problem simply counts the number of vertex covers. As another example, if H is the complete graph over k vertices (without self-loops), then the problem is exactly the k-COLORING problem for G. Many additional graph invariants can be expressed as $Z_{\mathbf{A}}(G)$ for appropriate $\mathbf{A}$. Consider the Hadamard matrix

$$\mathbf{H} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad (2)$$

where we index the rows and columns by $\{0, 1\}$. In $Z_{\mathbf{H}}(G)$, every product

$$\prod_{(u,v) \in E} H_{\xi(u),\xi(v)} \in \{1, -1\}$$

and is -1 precisely when the induced subgraph of G on $\xi^{-1}(1)$ has an odd number of edges. Thus, $(2^n - Z_{\mathbf{H}}(G))/2$ is the number of induced subgraphs of G with an odd number of edges. Also expressible as $Z_{\mathbf{A}}(\cdot)$ are S-flows where S is a subset of a finite Abelian group closed under inversion [6], and a scaled version of the Tutte polynomial $\hat{T}(x, y)$ where $(x - 1)(y - 1)$ is a positive integer. In [6], Freedman, Lovász, and Schrijver characterized the graph functions that can be expressed as $Z_{\mathbf{A}}(\cdot)$.

Key Results

In [5], Dyer and Greenhill first prove a complexity dichotomy theorem for all undirected graphs H. To state it formally, we give the following definition of *block-rank*-1 matrices:

Definition 1 (Block-rank-1 matrices) A nonnegative (but not necessarily symmetric) matrix $\mathbf{A} \in \mathbb{C}^{m \times n}$ is said to be *block-rank*-1 if after separate appropriate permutations of its rows and columns, $\mathbf{A}$ becomes a block diagonal matrix and every block is of rank 1.

It is clear that a nonnegative matrix $\mathbf{A}$ is block-rank-1 iff every 2×2 submatrix of $\mathbf{A}$ with at least three positive entries is of rank 1. Here is the dichotomy theorem of Dyer and Greenhill [5]:

Theorem 1 ([5]) *Given any undirected graph H, the $\#H$-coloring problem is in polynomial time if its adjacency matrix is block-rank-1 and is #P-hard otherwise.*

For the special case when H has two vertices, the dichotomy above states that $\#H$-coloring is in polynomial time if the number of 1s in its adjacency matrix is 0, 1, 2, or 4 and is #P-hard otherwise. For the latter case, one of the diagonal entries is 0 (as H is undirected), and $\#H$-coloring

is indeed the problem of counting independent sets [16]. However, proving a dichotomy theorem for H of arbitrary size is much more challenging. Besides counting independent sets, the other starting point used in [5] is the problem of counting proper q-colorings [12]. To show that there is a reduction from one of these two problems whenever H violates the block-rank-1 criterion, Dyer and Greenhill need to define a more general counting problem with vertex weights and employ the technique of interpolation [14,16], as well as two tools often used with interpolation, stretching, and thickening.

Later in [1], Bulatov and Grohe give a sweeping complexity dichotomy theorem that generalizes the result of Dyer and Greenhill to nonnegative symmetric matrices:

Theorem 2 ([1]) *Given any symmetric and nonnegative algebraic matrix* $\mathbf{A}$*, computing* $Z_{\mathbf{A}}(\cdot)$ *is in polynomial time if* $\mathbf{A}$ *is block-rank-*1 *and is #P-hard otherwise.*

This dichotomy theorem has since played an important role in many of the new developments in the study of counting graph homomorphisms as well as counting constraint satisfaction problem because of its enormous applicability. Many #P-hardness results are built on top of this dichotomy. A proof of the dichotomy theorem with a few shortcuts can also be found in [8].

Recently in a paper with both exceptional depth and conceptual vision, Goldberg, Jerrum, Grohe, and Thurley [7] proved a complexity dichotomy for all real-valued symmetric matrices:

Theorem 3 ([7]) *Given any symmetric and real algebraic matrix* $\mathbf{A}$*, the problem of computing* $Z_{\mathbf{A}}(\cdot)$ *is either in polynomial time or #P-hard.*

The exact tractability criterion in the dichotomy above, however, is much more technical and involved. Roughly speaking, the proof of the theorem proceeds by establishing a sequence of successively more stringent properties that a tractable $\mathbf{A}$ must satisfy. Ultimately, it arrives at a point where the satisfaction of these properties together implies that the computation of $Z_{\mathbf{A}}(G)$ can be reduced to the following sum:

$$\sum_{x_1,\ldots,x_n\in\mathbb{Z}_2} (-1)^{f_G(x_1,\ldots,x_n)} \tag{3}$$

where f_G is a quadratic polynomial over $\mathbb{Z}_2$ constructed from the input graph G efficiently. This sum is known to be computable in polynomial time in n, the number of variables (e.g., see [3] and [11, Theorem 6.30]). In particular, the latter immediately implies that the following two Hadamard matrices

$$\mathbf{H}_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \text{and} \quad \mathbf{H}_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

are both tractable. This can be seen from the following polynomial view of these two matrices. If we index the rows and columns of $\mathbf{H}_2$ by $\mathbb{Z}_2$ and index the rows and columns of $\mathbf{H}_4$ by $(\mathbb{Z}_2)^2$, then their (x, y)th entry and $((x_1, x_2), (y_1, y_2))$th entry are

$$(-1)^{xy} \quad \text{and} \quad (-1)^{x_1y_2+x_2y_1},$$

respectively. From here, it is easy to reduce $Z_{\mathbf{H}_2}(\cdot)$ and $Z_{\mathbf{H}_4}(\cdot)$ to (3).

Compared with the nonnegative domain [1,5], there are a lot more interesting tractable cases over the real numbers, e.g., the two Hadamard matrices above as well as their arbitrary tensor products. It is not surprising that the potential cancelations in the sum $Z_{\mathbf{A}}(\cdot)$ may in fact be the source of efficient algorithms for computing $Z_{\mathbf{A}}(\cdot)$ itself. This motivates Cai, Chen, and Lu to continue to investigate the computational complexity of $Z_{\mathbf{A}}(\cdot)$ with $\mathbf{A}$ being a symmetric complex matrix [2], because over the complex domain, there is a significantly richer variety of possible cancelations with the roots of unit, and more interesting tractable cases are expected. This turns out to be the case, and they prove the following complexity dichotomy:

Theorem 4 ([2]) *given any symmetric and algebraic complex matrix* $\mathbf{A} \in \mathbb{C}^{m\times m}$*, the problem of*

computing $Z_{\mathbf{A}}(\cdot)$ is either in polynomial time or #P-hard.

Applications

None is reported.

Open Problems

The efficient approximation of $Z_{\mathbf{A}}(\cdot)$ remains widely open even for small nonnegative matrices. See the entry "Approximating the Partition Function of Two-Spin Systems" for the current state of the art on this. Two families of counting problems that generalize $Z_{\mathbf{A}}(\cdot)$ are counting constraint satisfaction and Holant problems. Open problems in these two areas can be found in the two entries "Complexity Dichotomies for the Counting Constraint Satisfaction Problem" and "Holant Problems."

Experimental Results

None is reported.

URLs to Code and Data Sets

None is reported.

Cross-References

- Approximating the Partition Function of Two-Spin Systems
- Holant Problems
- Holographic Algorithms

Recommended Reading

1. Bulatov A, Grohe M (2005) The complexity of partition functions. Theor Comput Sci 348(2):148–186
2. Cai JY, Chen X, Lu P (2013) Graph homomorphisms with complex values: a dichotomy theorem. SIAM J Comput 42(3):924–1029
3. Carlitz L (1969) Kloosterman sums and finite field extensions. Acta Arith 16:179–193
4. Creignou N, Khanna S, Sudan M (2001) Complexity classifications of boolean constraint satisfaction problems. SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics, Philadelphia
5. Dyer M, Greenhill C (2000) The complexity of counting graph homomorphisms. Random Struct Algorithms 17(3–4):260–289
6. Freedman M, Lovász L, Schrijver A (2007) Reflection positivity, rank connectivity, and homomorphism of graphs. J Am Math Soc 20:37–51
7. Goldberg L, Grohe M, Jerrum M, Thurley M (2010) A complexity dichotomy for partition functions with mixed signs. SIAM J Comput 39(7):3336–3402
8. Grohe M, Thurley M (2011) Counting homomorphisms and partition functions. In: Grohe M, Makowsky J (eds) Model theoretic methods in finite combinatorics. Contemporary mathematics, vol 558. American Mathematical Society, Providence
9. Hell P, Nešetřil J (1990) On the complexity of H-coloring. J Comb Theory Ser B 48(1):92–110
10. Ladner R (1975) On the structure of polynomial time reducibility. J ACM 22(1):155–171
11. Lidl R, Niederreiter H (1997) Finite fields. Encyclopedia of mathematics and its applications, vol 20. Cambridge University Press, Cambridge
12. Linial N (1986) Hard enumeration problems in geometry and combinatorics. SIAM J Algebraic Discret Methods 7:331–335
13. Schaefer T (1978) The complexity of satisfiability problems. In: Proceedings of the 10th annual ACM symposium on theory of computing, San Diego, California, pp 216–226
14. Vadhan S (2002) The complexity of counting in sparse, regular, and planar graphs. SIAM J Comput 31:398–427
15. Valiant L (1979) The complexity of computing the permanent. Theor Comput Sci 8:189–201
16. Valiant L (1979) The complexity of enumeration and reliability problems. SIAM J Comput 8:410–421

Complexity of Bimatrix Nash Equilibria

Xi Chen
Computer Science Department, Columbia University, New York, NY, USA
Computer Science and Technology, Tsinghua University, Beijing, China

Keywords

Bimatrix games; Nash Equilibria; Two-player games

Years and Authors of Summarized Original Work

2006; Chen, Deng

Problem Definition

In the middle of the last century, Nash [8] studied general noncooperative games and proved that there exists a set of mixed strategies, now commonly referred to as a Nash equilibrium, one for each player, such that no player can benefit if he/she changes his/her own strategy unilaterally. Since the development of Nash's theorem, researchers have worked on how to compute Nash equilibria efficiently. Despite much effort in the last half century, no significant progress has been made on characterizing its algorithmic complexity, though both hardness results and algorithms have been developed for various modified versions.

An exciting breakthrough, which shows that computing Nash equilibria is possibly hard, was made by Daskalakis, Goldberg, and Papadimitriou [5], for games among four players or more. The problem was proven to be complete in **PPAD** (polynomial parity argument, directed version), a complexity class introduced by Papadimitriou in [9]. The work of [5] is based on the techniques developed in [6]. This hardness result was then improved to the three-player case by Chen and Deng [1] and Daskalakis and Papadimitriou [4], independently and with different proofs. Finally, Chen and Deng [2] proved that NASH, the problem of finding a Nash equilibrium in a bimatrix game (or two-player game), is **PPAD**-complete.

A bimatrix game is a noncooperative game between two players in which the players have m and n choices of actions (or pure strategies), respectively. Such a game can be specified by two $m \times n$ matrices $\mathbf{A} = (a_{i,j})$ and $\mathbf{B} = (b_{i,j})$. If the first player chooses action i and the second player chooses action j, then their payoffs are $a_{i,j}$ and $b_{i,j}$, respectively. A mixed strategy of a player is a probability distribution over his/her choices. Let $\mathbb{P}^n$ denote the set of all probability vectors in $\mathbb{R}^n$, i.e., nonnegative vectors whose entries sum to 1. The Nash equilibrium theorem on noncooperative games, when specialized to bimatrix games, states that for every bimatrix game $\mathcal{G} = (\mathbf{A}, \mathbf{B})$, there exists a pair of mixed strategies $(\mathbf{x}^* \in \mathbb{P}^m, \mathbf{y}^* \in \mathbb{P}^n)$, called a Nash equilibrium, such that for all $\mathbf{x} \in \mathbb{P}^m$ and $\mathbf{y} \in \mathbb{P}^n$, $(\mathbf{x}^*)^{\mathrm{T}}\mathbf{A}\mathbf{y}^* \geq \mathrm{x}^T\mathbf{A}\mathbf{y}^*$ and $(\mathbf{X}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y}^* \geq (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y}$.

Computationally, one might settle with an approximate Nash equilibrium. Let $\mathbf{A}_i$ denote the ith row vector of $\mathbf{A}$ and $\mathbf{B}_i$ denote the ith column vector of $\mathbf{B}$. An ε-well-supported Nash equilibrium of game $(\mathbf{A},\mathbf{B})$ is a pair of mixed strategies $(\mathbf{x}^*, \mathbf{y}^*)$ such that

$$\mathbf{A}_i\mathbf{y}^* > \mathbf{A}_j\mathbf{y}^* + \epsilon \Rightarrow x_j^* = 0, \forall\, i, j : 1 \leq i, j \leq m;$$

$$(\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}_i > (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}_j + \epsilon \Rightarrow y_j^* = 0, \forall\, i, j : 1 \leq i, j \leq n.$$

Definition 1 (2-NASH and NASH) The input instance of problem 2-NASH is a pair $(\mathcal{G}, 0^k)$ where $\mathcal{G}$ is a bimatrix game and the output is a 2^{-k}-well-supported Nash equilibrium of $\mathcal{G}$. The input of problem NASH is a bimatrix game $\mathcal{G}$ and the output is an exact Nash equilibrium of $\mathcal{G}$.

Key Results

A binary relation $R \subset \{0,1\}^* \times \{0,1\}^*$ is *polynomially balanced* if there exists a polynomial p such that for all pairs $(x, y) \in R$, $|y| \leq p(|x|)$. It is a *polynomial-time computable relation* if for each pair (x, y), one can decide whether or not $(x, y) \in R$ in polynomial time in $|x| + |y|$. The **NP** search problem Q_R specified by R is defined as follows: given $x \in \{0,1\}^*$, if there exists y such that $(x, y) \in R$, return y; otherwise, return a special string "no."

Relation R is *total* if for every $x \in \{0,1\}^*$, there exists a y such that $(x, y) \in R$. Following [7], let **TFNP** denote the class of all **NP** search problems specified by total relations. A search problem $Q_{R_1} \in$ **TFNP** is *polynomial-time reducible* to problem $Q_{R_2} \in$ **TFNP** if there exists a pair of polynomial-time computable functions (f, g) such that for every x of R_1, if y satisfies that $(f(x), y) \in R_2$, then $(x, g(y)) \in R_1$.

Furthermore, Q_{R1} and Q_{R2} are polynomial-time equivalent if Q_{R2} is also reducible to Q_{R1}.

The complexity class **PPAD** is a subclass of **TFNP**, containing all the search problems which are polynomial-time reducible to:

Definition 2 (Problem LEAFD) The input instance of LEAFD is a pair $(M, 0^n)$, where M defines a polynomial-time Turing machine satisfying:

1. For every $v \in \{0,1\}^n$, $M(v)$ is an ordered pair (u_1, u_2) with $u_1, u_2 \in \{0,1\}^n \cup \{\text{“no”}\}$.
2. $M(0^n) = (\text{“no,”}\ 1^n)$ and the first component of $M(1^n)$ is 0^n.

This instance defines a directed graph $G = (V, E)$ with $V = \{0,1\}^n$. Edge $(u, v) \in E$ iff v is the second component of $M(u)$ and u is the first component of $M(v)$.

The output of problem LEAFD is a directed leaf of G other than 0^n. Here a vertex is called a *directed leaf* if its out-degree plus in-degree equals one.

A search problem in **PPAD** is said to be *complete* in **PPAD** (or **PPAD**-complete) if there exists a polynomial-time reduction from LEAFD to it.

Theorem ([2]) *2-Nash and Nash are* **PPAD***-complete.*

Applications

The concept of Nash equilibria has traditionally been one of the most influential tools in the study of many disciplines involved with strategies, such as political science and economic theory. The rise of the Internet and the study of its anarchical environment have made the Nash equilibrium an indispensable part of computer science. Over the past decades, the computer science community has contributed a lot to the design of efficient algorithms for related problems. This sequence of results [1–6], for the first time, provides *some evidence* that the problem of finding a Nash equilibrium is possibly hard for **P**. These results are very important to the emerging discipline, algorithmic game theory.

Open Problems

This sequence of works shows that $(r+1)$-player games are polynomial-time reducible to r-player games for every $r \geq 2$, but the reduction is carried out by first reducing $(r+1)$-player games to a fixed-point problem and then further to r-player games. Is there a natural reduction that goes directly from $(r+1)$-player games to r-player games? Such a reduction could provide a better understanding for the behavior of multiplayer games. Although many people believe that **PPAD** is hard for **P**, there is no strong evidence for this belief or intuition. The natural open problem is: can one rigorously prove that class **PPAD** is hard, under one of those generally believed assumptions in theoretical computer science, like “**NP** is not in **P**” or “one-way function exists”? Such a result would be extremely important to both computational complexity theory and algorithmic game theory.

Cross-References

▸ Matching Market Equilibrium Algorithms

Recommended Reading

1. Chen X, Deng X (2005) 3-Nash is ppad-complete. ECCC, TR05-134
2. Chen X, Deng X (2006) Settling the complexity of two-player Nash-equilibrium. In: Proceedings of the 47th annual IEEE symposium on foundations of computer science (FOCS'06), Berkeley, pp 261–272
3. Chen X, Deng X, Teng SH (2006) Computing Nash equilibria: approximation and smoothed complexity. In: Proceedings of the 47th annual IEEE symposium on foundations of computer science (FOCS'06), Berkeley, pp 603–612
4. Daskalakis C, Papadimitriou CH (2005) Three-player games are hard. ECCC, TR05-139
5. Daskalakis C, Goldberg PW, Papadimitriou CH (2006) The complexity of computing a Nash equilibrium. In: Proceedings of the 38th ACM symposium on theory of computing (STOC'06), Seattle, pp 71–78

6. Goldberg PW, Papadimitriou CH (2006) Reducibility among equilibrium problems. In: Proceedings of the 38th ACM symposium on theory of computing (STOC'06), Seattle, pp 61–70
7. Megiddo N, Papadimitriou CH (1991) On total functions, existence theorems and computational complexity. Theor Comput Sci 81:317–324
8. Nash JF (1950) Equilibrium point in n-person games. Proc Natl Acad USA 36(1):48–49
9. Papadimitriou CH (1994) On the complexity of the parity argument and other inefficient proofs of existence. J Comput Syst Sci 48: 498–532

Complexity of Core

Qizhi Fang
School of Mathematical Sciences, Ocean University of China, Qingdao, Shandong Province, China

Keywords

Balanced; Least core

Years and Authors of Summarized Original Work

2001; Fang, Zhu, Cai, Deng

Problem Definition

The core is one of the most important solution concepts in cooperative game, which is based on the coalition rationality condition: no subgroup of the players will do better if they break away from the joint decision of all players to form their own coalition. The principle behind this condition can be seen as an extension to that of the Nash Equilibrium in noncooperative games. The work of Fang, Zhu, Cai, and Deng [4] discusses the computational complexity problems related to the cores of some cooperative game models arising from combinatorial optimization problems, such as flow games and Steiner tree games.

A cooperative game with side payments is given by the pair (N, v), where $N = \{1, 2, \ldots, n\}$ is the player set and $v : 2^N \to R$ is the characteristic function. For each coalition $S \subseteq N$, the value $v(S)$ is interpreted as the profit or cost achieved by the collective action of players in S without any assistance of players in $N \backslash S$. A game is called a profit (cost) game if the characteristic function values measure the profit (cost) achieved by the coalitions. Here, the definitions are only given for profit games, symmetric statements hold for cost games.

A vector $x = \{x_1, x_2, \ldots, x_n\}$ is called an imputation if it satisfies $\sum_{i \in N} x_i = v(N)$ and $\forall i \in N : x_i \geq v(\{i\})$. The core of the game (N, v) is defined as:

$$\mathcal{C}(v) = \{x \in R^n : x(N) = v(N) \text{ and } x(S) \geq v(S),\ \forall S \subseteq N\},$$

where $x(S) = \sum_{i \in S} x_i$ for $S \subseteq N$. A game is called *balanced*, if its core is nonempty, and *totally balanced*, if every subgame (i.e., the game obtained by restricting the player set to a coalition and the characteristic function to the power set of that coalition) is balanced.

It is a challenge for the algorithmic study of the core, since there are an exponential number of constraints imposed on its definition. The following computational complexity questions have attracted much attention from researchers:

1. *Testing balancedness:* Can it be tested in polynomial time whether a given instance of the game has a nonempty core?
2. *Checking membership:* Can it be checked in polynomial time whether a given imputation belongs to the core?
3. *Finding a core member:* Is it possible to find an imputation in the core in polynomial time?

In reality, however, there is an important case in which the characteristic function value of a coalition can be evaluated via a combinatorial optimization problem, subject to constraints of resources controlled by the players of this coalition. In such circumstances, the input size of a game is the same as that of the related optimization problem, which is usually polynomial

in the number of players. Therefore, this class of games, called combinatorial optimization games, fits well into the framework of algorithm and complexity analysis. Flow games and Steiner tree games discussed in Fang et al. [4] fall within this scope.

Flow Game Let $D = (V, E; \omega; s, t)$ be a directed flow network, where V is the vertex set, E is the arc set, $\omega: E \to R^+$ is the arc capacity function, and s and t are the source and the sink of the network, respectively. Assume that each player controls one arc in the network. The value of a maximum flow can be viewed as the profit achieved by the players in cooperation. The flow game $\Gamma_f = (E, v)$ associated with the network D is defined as follows:

(i) The player set is E.
(ii) $\forall S \subseteq E$, $v(S)$ is the value of a maximum flow from s to t in the subnetwork of D consisting only of arcs belonging to S.

In Kailai and Zemel [6] and Deng et al. [1], it was shown that the flow game is totally balanced and finding a core member can be done in polynomial time.

Problem 1 (Checking membership for flow game) INSTANCE: A flow network $D = (V, E; \omega; s, t)$ and $x : E \to R^+$.

QUESTION: Is it true that $x(E) = v(E)$ and $x(S) \geq v(S)$ for all subsets $S \subset E$?

Steiner Tree Game Let $G = (V, E; \omega)$ be an edge-weighted graph with $V = \{v_0\} \cup N \cup M$, where $N, M \subseteq V \backslash \{v_0\}$ are disjoint. v_0 represents a central supplier, N represents the consumer set, M represents the switch set, and $\omega(e)$ denotes the cost of connecting the two endpoints of edge e directly. It is required to connect all the consumers in N to the central supplier v_0. The connection is not limited to using direct links between two consumers or a consumer and the central supplier; it may pass through some switches in M. The aim is to construct the cheapest connection and distribute the connection cost among the consumers fairly. Then, the associated Steiner tree game $\Gamma_s = (N, \gamma)$ is defined as follows:

(i) The player set is N.
(ii) $\forall S \subseteq N$, $\gamma(S)$ is the weight of a minimum Steiner tree on G w.r.t. the set $S \cup \{v_0\}$, that is, $\gamma(S) = \min\{\sum_{e \in E_S} \omega(e) : T_S = (V_S, E_S)$ is a subtree of G with $V_S \supseteq S \cup \{v_0\}\}$.

Different from flow games, the core of a Steiner tree game may be empty. An example with an empty core was given in Megiddo [9].

Problem 2 (Testing balancedness for a Steiner tree game) INSTANCE: An edge-weighted graph $G = (V, E; \omega)$ with $V = \{v_0\} \cup N \cup M$.

QUESTION: Does there exist a vector $x : N \to R^+$ such that $x(N) = \gamma(N)$ and $x(S) \leq \gamma(S)$ for all subsets $S \subset N$?

Problem 3 (Checking membership for a Steiner tree game) INSTANCE: An edge-weighted graph $G = (V, E; \omega)$ with $V = \{v_0\} \cup N \cup M$ and $x : N \to R^+$.

QUESTION: Is it true that $x(N) = \gamma(N)$ and $x(S) \leq \gamma(S)$ for all subsets $S \subset N$?

Key Results

Theorem 1 *It is $\mathcal{NP}$-complete to show that given a flow game $\Gamma_f = (E, v)$ defined on network $D = (V, E; \omega; s, t)$ and a vector $x : E \to R^+$ with $x(E) = v(E)$, whether there exists a coalition $S \subset N$ such that $x(S) < v(S)$. That is, checking membership of the core for flow games is co-$\mathcal{NP}$-complete.*

The proof of Theorem 1 yields directly the same conclusion for linear production games. In Owen's linear production game [10], each player j ($j \in N$) is in possession of an individual resource vector b^j. For a coalition S of players, the profit obtained by S is the optimum value of the following linear program:

$$\max \left\{ c^t y : Ay \leq \sum_{j \in S} b^j,\ y \geq 0 \right\}.$$

That is, the characteristic function value is what the coalition can achieve in the linear production

model with the resources under the control of its players. Owen showed that one imputation in the core can also be constructed through an optimal dual solution to the linear program which determines the value of N. However, in general, there exist some imputations in the core which cannot be obtained in this way.

Theorem 2 *Checking membership of the core for linear production games is co-$\mathcal{NP}$-complete.*

The problem of finding a minimum Steiner tree in a network is $\mathcal{NP}$-hard; therefore, in a Steiner tree game, the value $\gamma(S)$ of each coalition S may not be obtained in polynomial time. It implies that the complement problem of checking membership of the core for Steiner tree games may not be in $\mathcal{NP}$.

Theorem 3 *It is $\mathcal{NP}$-hard to show that given a Steiner tree game $\Gamma_s = (N, \gamma)$ defined on network $G = (V, E; \omega)$ and a vector $x : N \to R^+$ with $x(N) = \gamma(N)$, whether there exists a coalition $S \subset N$ such that $x(S) > \gamma(S)$. That is, checking membership of the core for Steiner tree games is $\mathcal{NP}$-hard.*

Theorem 4 *Testing balancedness for Steiner tree games is $\mathcal{NP}$-hard.*

Given a Steiner tree game $\Gamma_s = (N, \gamma)$ defined on network $G = (V, E; \omega)$ and a subset $S \subseteq N$, in the subgame (S, γ_S), the value $\gamma(S')$ $(S' \subseteq S)$ is the weight of a minimum Steiner tree of G w.r.t. the subset $S' \cup \{v_0\}$, where all the vertices in $N \setminus S$ are treated as switches but not consumers. It is further proved in Fang et al. [4] that determining whether a Steiner tree game is totally balanced is also $\mathcal{NP}$-hard. This is the first example of $\mathcal{NP}$-hardness for the totally balanced condition.

Theorem 5 *Testing total balancedness for Steiner tree games is $\mathcal{NP}$-hard.*

Applications

The computational complexity results on the cores of combinatorial optimization games have been as diverse as the corresponding combinatorial optimization problems. For example:

1. In matching games [2], testing balancedness, checking membership, and finding a core member can all be done in polynomial time.
2. In both flow games and minimum-cost spanning tree games [3, 4], although their cores are always nonempty and a core member can be found in polynomial time, the problem of checking membership is *co-$\mathcal{NP}$*-complete.
3. In facility location games [5], the problem of testing balancedness is in general $\mathcal{NP}$-hard; however, given the information that the core is nonempty, both finding a core member and checking membership can be solved efficiently.
4. In a game of sum of edge weight defined on a graph [1], all the problems of testing balancedness, checking membership, and finding a core member are $\mathcal{NP}$-hard.

Based on the concept of bounded rationality [3, 8], it is suggested that computational complexity be taken as an important factor in considering rationality and fairness of a solution concept. That is, the players are not willing to spend super-polynomial time to search for the most suitable solution. In the case when the solutions of a game do not exist or are difficult to compute or to check, it may not be simple to dismiss the problem as hopeless, especially when the game arises from important applications. Hence, various conceptual approaches are proposed to resolve this problem.

When the core of a game is empty, it motivates conditions ensuring nonemptiness of approximate cores. A natural way to approximate the core is the *least core*. Let (N, v) be a profit cooperative game. Given a real number ε, the ε-core is defined to contain the imputations such that $x(S) \geq v(S) - \varepsilon$ for each nonempty proper subset S of N. The *least core* is the intersection of all nonempty ε-cores. Let ε^* be the minimum value of ε such that the ε-core is empty and then the least core is the same as the ε^*-core.

The concept of the least core poses new challenges in regard to algorithmic issues. The most natural problem is how to efficiently compute the value ε^* for a given cooperative game. The catch is that the computation of ε^* requires solving

of a linear program with an exponential number of constrains. Though there are cases where this value can be computed in polynomial time [7], it is in general very hard. If the value of ε^* is considered to represent some subsidies given by the central authority to ensure the existence of the cooperation, then it is significant to give the approximate value of it even when its computation is $\mathcal{NP}$-hard.

Another possible approach is to interpret approximation as bounded rationality. For example, it would be interesting to know if there is any game with a property that for any $\varepsilon > 0$, checking membership in the ε-core can be done in polynomial time, but it is $\mathcal{NP}$-hard to tell if an imputation is in the core. In such cases, the restoration of cooperation would be a result of bounded rationality. That is to say, the players would not care an extra gain or loss of ε as the expense of another order of degree of computational resources. This methodology may be further applied to other solution concepts.

Cross-References

- General Equilibrium
- Nucleolus

Recommended Reading

1. Deng X, Papadimitriou C (1994) On the complexity of cooperative game solution concepts. Math Oper Res 19:257–266
2. Deng X, Ibaraki T, Nagamochi H (1999) Algorithmic aspects of the core of combinatorial optimization games. Math Oper Res 24:751–766
3. Faigle U, Fekete S, Hochstättler W, Kern W (1997) On the complexity of testing membership in the core of min-cost spanning tree games. Int J Game Theory 26:361–366
4. Fang Q, Zhu S, Cai M, Deng X (2001) Membership for core of LP games and other games. In: COCOON 2001. Lecture notes in computer science, vol 2108. Springer, Berlin/Heidelberg, pp 247–246
5. Goemans MX, Skutella M (2004) Cooperative facility location games. J Algorithms 50:194–214
6. Kalai E, Zemel E (1982) Generalized network problems yielding totally balanced games. Oper Res 30:998–1008
7. Kern W, Paulusma D (2003) Matching games: the least core and the nucleolus. Math Oper Res 28:294–308
8. Megiddo N (1978) Computational complexity and the game theory approach to cost allocation for a tree. Math Oper Res 3:189–196
9. Megiddo N (1978) Cost allocation for steiner trees. Networks 8:1–6
10. Owen G (1975) On the core of linear production games. Math Program 9:358–370

C

Compressed Document Retrieval on String Collections

Sharma V. Thankachan
School of CSE, Georgia Institute of Technology, Atlanta, USA

Keywords

Compressed data structures; Document retrieval; String algorithms; Top-k

Years and Authors of Summarized Original Work

2009; Hon, Shah, Vitter
2013; Belazzougui, Navarro, Valenzuela
2013; Tsur
2014; Hon, Shah, Thankachan, Vitter
2014; Navarro, Thankachan

Problem Definition

We face the following problem.

Problem 1 (Top-k document retrieval) *Let $\mathcal{D} = \{\mathsf{T}_1, \mathsf{T}_2, \ldots, \mathsf{T}_D\}$ be a collection of D documents of n characters in total, drawn from an alphabet set $\Sigma = [\sigma]$. The relevance of a document T_d with respect to a pattern P, denoted by $w(P, d)$ is a function of the set of occurrences of P in T_d. Our task is to index $\mathcal{D}$, such that whenever a pattern $P[1, p]$ and a parameter k comes as a query, the k documents with the highest $w(P, \cdot)$ values can be reported efficiently.*

Compressed Document Retrieval on String Collections, Table 1 Indexes of space $2|\mathsf{CSA}| + D\log(n/D) + O(D) + o(n)$ bits

Source	Report time per document
Hon et al. [3]	$O(t_{\mathsf{SA}} \log^{3+\epsilon} n)$
Gagie et al. [2]	$O(t_{\mathsf{SA}} \log D \log(D/k) \log^{1+\epsilon} n)$
Belazzougui et al. [1]	$O(t_{\mathsf{SA}} \log k \log(D/k) \log^{\epsilon} n)$
Hon et al. [4]	$O(t_{\mathsf{SA}} \log k \log^{\epsilon} n)$

Compressed Document Retrieval on String Collections, Table 2 Indexes of space $|\mathsf{CSA}| + D\log(n/D) + O(D) + o(n)$ bits

Source	Report time per document
Tsur [12]	$O(t_{\mathsf{SA}} \log k \log^{1+\epsilon} n)$
Navarro and Thankachan [8]	$O(t_{\mathsf{SA}} \log^2 k \log^{\epsilon} n)$

Traditionally, inverted indexes are employed for this task in Information Retrieval. However, they are not powerful enough to handle scenarios where the documents need to be treated as general strings over an arbitrary alphabet set (e.g., genome sequences in bioinformatics, text in many East-Asian languages) [5]. Hon et al. [3] proposed the first solution for Problem 1, requiring $O(n \log n)$ bits of space and $O(p + k \log k)$ query time. Later, optimal $O(p + k)$ query time indexes were proposed by Navarro and Nekrich [7], and also by Shah et al. [11]. There also exist compressed/compact space solutions, tailored to specific relevance functions (mostly term-frequency or PageRank). In this article, we briefly survey the compressed space indexes for Problem 1 for the case where the relevance function is term-frequency (i.e., $w(P, d)$ is the number of occurrences of P in T_d), which we call the *Compressed Top-k Frequent Document Retrieval* (CTFDR) problem.

Key Results

First we introduce some notations. For convenience, we append a special character \$ to every document. Then, $\mathsf{T} = \mathsf{T}_1 \circ \mathsf{T}_2 \circ \cdots \circ \mathsf{T}_D$ is the concatenation of all documents. GST, SA, and CSA are the suffix tree, suffix array, and a compressed suffix array of T, respectively. Notice that both GST and SA take $O(n \log n)$ bits of space, whereas the space of CSA ($|\mathsf{CSA}|$ bits) can be made as close as the minimum space for maintaining $\mathcal{D}$ (which is not more than $n \log \sigma$ bits) by choosing an appropriate version of CSA [6]. Using CSA, the suffix range $[sp, ep]$ of $P[1, p]$, as well as any $\mathsf{SA}[\cdot]$, can be computed in times $\mathsf{search}(p)$ and t_{SA}, respectively. Hon et al. [3] gave the first solution for the CTFDR Problem, requiring roughly $2|\mathsf{CSA}|$ bits of space, whereas the first space-optimal index was given by Tsur [12]. Various improvements on both results have been proposed and are summarized in Tables 1 and 2. Notice that the total query time is $\mathsf{search}(p)$ plus k times per reported document.

Notations and Basic Framework

The suffix tree GST of T can be considered as a generalized suffix tree of $\mathcal{D}$, where

- ℓ_i is the ith leftmost leaf in GST
- $\mathrm{doc}(\ell_i)$ is the document to which the suffix corresponding to ℓ_i belongs
- $\mathsf{Leaf}(u)$ is the set of leaves in the subtree of node u
- $\mathsf{tf}(u, d)$ is the number of leaves in $\mathsf{Leaf}(u)$ with $\mathrm{doc}(\cdot) = d$
- $\mathsf{Top}(u, k)$ is the set k of document identifiers with the highest $\mathsf{tf}(u, \cdot)$ value

From now onwards, we assume all solutions consists of a fully compressed representation of GST in $|\mathsf{CSA}| + o(n)$ bits [10], and a bitmap

$B[1,n]$, where $B[i] = 1$ iff $\mathsf{T}[i] = \$$. We use a $D\log(n/D) + O(D) + o(n)$ bits representation of B with constant time rank/select query support [9]. Therefore, $\mathrm{doc}(\ell_i)$ can be computed as 1 plus the number of 1's in $B[1, \mathsf{SA}[i]-1]$ in time $t_{\mathsf{SA}} + O(1)$. Observe that a CTFDL query (P,k) essentially asks to return the set $\mathsf{Top}(u_P, k)$, where u_P is the *locus node* of P in GST. Any superset of $\mathsf{Top}(u_P,k)$ be called as a candidate set of (P,k). The following lemma is crucial.

Lemma 1 *The set* $\mathsf{Top}(w,k) \cup \{\mathrm{doc}(\ell_i) | \ell_i \in \mathsf{Leaf}(u_P) \backslash \mathsf{Leaf}(w)\}$ *is a candidate set of* (P,k)*, where* w *is any node in the subtree of* u_P*.*

All query processing algorithms consist of the following two steps: (i) Generate a candidate set $\mathcal{C}$ of size as close to k as possible. (ii) Compute $\mathsf{tf}(u_P, d)$ of all $d \in \mathcal{C}$ and report those k document identifiers with the highest $\mathsf{tf}(u_P, \cdot)$ values as output.

An Index of Size $\approx 2|\mathsf{CSA}|$ Bits

Queries are categorized into $O(\log D)$ different types.

Definition 1 A query (P,k) is of type x if $\lceil \log k \rceil = x$.

We start with the description of a structure DS_x (of size $|\mathsf{DS}_x|$ bits) that along with GST and B can generate a candidate set of size proportional to k for any type-x query. The first step is to identify a set Mark_g of nodes in GST using the scheme described in Lemma 2 (parameter g will be fixed later). Then maintain $\mathsf{Top}(u', 2^x)$ for all $u' \in \mathsf{Mark}_g$.

Lemma 2 ([3]) *There exists a scheme to identify a set* Mark_g *of nodes in* GST *(called marked nodes) based on a grouping factor* g*, where the following conditions are satisfied: (i)* $|\mathsf{Mark}_g| = O(n/g)$*, (ii) if it exists, the highest marked node* $u' \in \mathsf{Mark}_g$ *in the subtree of any node* u *is unique, and* $\mathsf{Leaf}(u) \backslash \mathsf{Leaf}(u') \leq 2g$*. For example,* Mark_g *can be the set of all lowest common ancestor (LCA) nodes of* ℓ_i *and* ℓ_{i+g}*, where* i *is an integer multiple of* g*.*

Using DS_x, any type-x query (P,k) can be processed as follows: find the highest node u'_P in the subtree of u_P and generate the following candidate set.

$$\mathsf{Top}(u'_P, 2^x) \cup \{\mathrm{doc}(\ell_i) | \ell_i \in \mathsf{Leaf}(u_P) \backslash \mathsf{Leaf}(u'_P)\}$$

From the properties described in Lemma 2, the cardinality of this set is $O(2^x + g) = O(k+g)$ and the size of DS_x is $O((n/g)2^x \log D)$ bits. By fixing $g = 2^x \log^{2+\epsilon} n$ [3], we can bound the set cardinality by $O(k \log^{2+\epsilon} n)$ and $|\mathsf{DS}_x|$ by $O(n/\log^{1+\epsilon} n)$ bits. Therefore, we maintain DS_x for $x = 1, 2, 3, \ldots, \log D$ in $o(n)$ bits overall, and whenever a query comes, generate a candidate set of size $O(k \log^{2+\epsilon} n)$ using appropriate structures. The observation by Belazzougui et al. [1] is that $g = x2^x \log^{1+\epsilon} n$ in the above analysis yields a candidate set of even lower cardinality $O(k \log k \log^{1+\epsilon} n)$, without blowing up the space.

Later, Hon et al. [4] came up with another strategy for generating a candidate set of even smaller size, $O(k \log k \log^{\epsilon} n)$. They associate another structure DS^*_x (of space $|\mathsf{DS}^*_x|$ bits) with each DS_x. Essentially, DS^*_x maintains $\mathsf{Top}(u'', 2^x)$ of every $u'' \in \mathsf{Mark}_h$ with $h = x2^x \log^{\epsilon} n$ in an *encoded* form. Now, whenever a type-x query (P,k) comes, we first find the highest node u''_P in the subtree of u_P that belongs to Mark_h and generate the candidate set $\mathsf{Top}(u''_P, 2^x) \cup \{\mathrm{doc}(\ell_i) | \ell_i \in \mathsf{Leaf}(u_P) \backslash \mathsf{Leaf}(u''_P)\}$, whose cardinality is $O(2^x + h) = O(k \log k \log^{\epsilon} n)$.

We now describe the scheme for encoding a particular $\mathsf{Top}(u'', 2^x)$. Let u' be the highest node in the subtree of u'', that belongs to Mark_g. Then, $\mathsf{Top}(u'', 2^x) \subseteq \mathsf{Top}(u', 2^x) \cup \{\mathrm{doc}(\ell_i) | \ell_i \in \mathsf{Leaf}(u'') \backslash \mathsf{Leaf}(u')\}$. Notice that $\mathsf{Top}(u'', 2^x)$ is stored in DS_x and any $\mathrm{doc}(\ell_i)$ can be decoded in $O(t_{\mathsf{SA}})$ time. Therefore, instead of explicitly storing an entry d within $\mathsf{Top}(u'', 2^x)$ in $\log D$ bits, we can refer to the position of d in $\mathsf{Top}(u', 2^x)$ if $d \in \mathsf{Top}(u', 2^x)$, else refer to the relative position of a leaf node ℓ_i in $\mathsf{Leaf}(u'') \backslash \mathsf{Leaf}(u')$ with $\mathrm{doc}(\ell_i) = d$. Therefore, maintaining the following two bitmaps is sufficient.

- $\mathsf{F}[1, 2^x]$, where $\mathsf{F}[i] = 1$ iff ith entry in $\mathsf{Top}(u', 2^x)$ is present in $\mathsf{Top}(u'', 2^x)$
- $\mathsf{F}'[1, |\mathsf{Leaf}(u'')\backslash\mathsf{Leaf}(u')|]$, $\mathsf{F}'[i] = 1$ iff $\mathsf{doc}(\cdot)$ of ith leaf node in $\mathsf{Leaf}(u'')\backslash\mathsf{Leaf}(u')$ is present in $\mathsf{Top}(u'', 2^x)$, but not in $\mathsf{Top}(u', 2^x)$.

As the total length and the number of 1's over F and F' is $O(g + 2^x)$ and $O(2^x)$, respectively, we can encode them in $O(2^x \log(g/2^x)) = O(2^x \log\log n)$ bits. Therefore, $|\mathsf{DS}^*_x| = O((n/(x2^x \log^\epsilon n))2^x \log\log n)$ bits and $\sum_{x=1}^{\log D} |\mathsf{DS}_x| = o(n)$ bits.

Lemma 3 *By maintaining a $|\mathsf{CSA}| + o(n) + D\log(n/D) + O(D)$ bits space structure (which includes the space of CSA and B), a candidate set of size $O(k \log k \log^\epsilon n)$ can be generated for any query (P, k) in time $O(t_{\mathsf{SA}} \cdot k \log k \log^\epsilon n)$.*

We now turn our attention to Step 2 of the query algorithm. Let $[sp, ep]$ be the suffix range of P in CSA and $[sp_d, ep_d]$ be the suffix range of P in CSA_d, the compressed suffix array of T_d. Hon et al. [3] showed that by additionally maintaining all CSA_d's (in space roughly $\approx |\mathsf{CSA}|$ bits), any $[sp_d, ep_d]$ can be computed in time $O(t_{\mathsf{SA}} \log n)$ (and thus $\mathsf{tf}(P, d) = ep_d - sp_d + 1$). Belazzougui et al. [1] improved this time to $O(t_{\mathsf{SA}} \log\log n)$ using $o(n)$ extra bits. Combined with Lemma 3, this gives the following.

Theorem 1 ([4]) *Using a $2|\mathsf{CSA}| + o(n) + D\log(n/D) + O(D)$ bits space index, top-k frequent document retrieval queries can be answered in $O(\mathsf{search}(p) + k \cdot t_{\mathsf{SA}} \log k \log^\epsilon n)$ time.*

Space-Optimal Index

Space-optimal indexes essentially circumvent the need of CSA_d's. We first present a simplified version of Tsur's index (with slightly worse query time). To handle type-x queries, first identify the set of nodes Mark_g based on a grouping factor g (to be fixed later). Tsur proved that each node $u' \in \mathsf{Mark}_g$ can be associated with a set $\mathsf{Set}(u')$ of $O(2^x + \sqrt{2^x g})$ document identifiers, such that $\mathsf{Set}(u'_P)$ represents a candidate set, where u'_P is the highest node in the subtree of u_P that belongs to Mark_g. Therefore, we can store $S(u')$ for every $u' \in \mathsf{Mark}_g$ along with $\mathsf{tf}(u', d)$ of every $d \in S(u')$ in $O((n/g)(2^x + \sqrt{2^x g}) \log n)$ bits. Now a type-x query can be processed as follows:

1. Find u'_P, the highest node in the subtree of u_P that belongs to Mark_g.
2. Extract $\mathsf{Set}(u'_P)$ and $\mathsf{tf}(u'_P, d)$ of all $d \in \mathsf{Set}(u'_P)$.
3. Scan the leaves in $\mathsf{Leaf}(u_P)\backslash\mathsf{Leaf}(u'_P)$, decode the corresponding $\mathsf{doc}(\cdot)$ values and compute $\mathsf{tf}(u_P, d) - \mathsf{tf}(u'_P, d)$ for all $d \in \mathsf{Set}(u'_P)$.
4. Then obtain $\mathsf{tf}(u_P, d) = \mathsf{tf}(u'_P, d) + (\mathsf{tf}(u_P, d) - \mathsf{tf}(u'_P, d))$ for all $d \in \mathsf{Set}(u'_P)$.
5. Report k documents within $\mathsf{Set}(u'_P)$ with highest $\mathsf{tf}(u_P, \cdot)$ values as output.

In summary, an $O((n/g)(2^x + \sqrt{2^x g}) \log n)$-bit structure (along with GST and B) can answer any type-x query in $O(\mathsf{search}(p) + (|\mathsf{Set}(u'_P)| + |\mathsf{Leaf}(u_P)\backslash\mathsf{Leaf}(u'_P)|) \cdot t_{\mathsf{SA}}) = O(\mathsf{search}(p) + (2^x + \sqrt{2^x g} + g) \cdot t_{\mathsf{SA}}) = O(\mathsf{search}(p) + (g + 2^x) \cdot t_{\mathsf{SA}})$ time. By fixing $g = x^2 2^x \log^{2+\epsilon} n$, the query time can be bounded by $O(\mathsf{search}(p) + k \cdot t_{\mathsf{SA}} \log^2 k \log^{2+\epsilon} n)$ and the overall space corresponding to $x = 1, 2, 3, \ldots, \log D$ is $o(n)$ bits. We remark that the index originally proposed by Tsur is even faster.

Navarro and Thankachan [8] observed that each document identifier and the associated $\mathsf{tf}(\cdot,\cdot)$ value can be compressed into $O(\log\log n)$ bits. For compressing document identifiers, ideas from Hon et al. [4] were borrowed. For compressing $\mathsf{tf}(\cdot,\cdot)$ values, they introduced an $o(n)$-bit structure, called *sampled document array* that can compute an approximate value of any $\mathsf{tf}(\cdot,\cdot)$ (denoted by $\mathsf{tf}^*(\cdot,\cdot)$) in time $O(\log\log n)$ within an additive error of at most $\log^2 n$. This means that, instead of storing $\mathsf{tf}(\cdot,\cdot)$, storing $\mathsf{tf}(\cdot,\cdot) - \mathsf{tf}^*(\cdot,\cdot)$ (in just $O(\log\log n)$ bits) is sufficient. In summary, by maintaining an $O((n/g)(2^x + \sqrt{2^x g}) \log\log n)$-bit structure (along with GST, B and the sampled document array), any type-x query can be answered in $O((g + 2^x) \cdot t_{\mathsf{SA}})$ time.

A similar analysis with $g = x^2 2^x \log^\epsilon n$ gives the following result.

Theorem 2 ([8]) *Top-k frequent document retrieval queries can be answered in* $O(\mathsf{search}(p) + k \cdot t_{\mathsf{SA}} \log^2 k \log^\epsilon n)$ *time using a* $|\mathsf{CSA}| + o(n) + D \log(n/D) + O(D)$*-bit index.*

Cross-References

- Compressed Suffix Array
- Compressed Suffix Trees
- Document Retrieval on String Collections
- Rank and Select Operations on Bit Strings
- Suffix Trees and Arrays

Recommended Reading

1. Belazzougui D, Navarro G, Valenzuela D (2013) Improved compressed indexes for full-text document retrieval. J Discret Algorithms 18:3–13
2. Gagie T, Kärkkäinen J, Navarro G, Puglisi SJ (2013) Colored range queries and document retrieval. Theor Comput Sci 483:36–50
3. Hon WK, Shah R, Vitter JS (2009) Space-efficient framework for top-k string retrieval problems. In: FOCS, Atlanta, pp 713–722
4. Hon WK, Shah R, Thankachan SV, Vitter JS (2014) Space-efficient frameworks for top-k string retrieval. J ACM 61(2):9
5. Navarro G (2014) Spaces, trees, and colors: the algorithmic landscape of document retrieval on sequences. ACM Comput Surv 46(4):52
6. Navarro G, Mäkinen V (2007) Compressed full-text indexes. ACM Comput Surv 39(1):2
7. Navarro G, Nekrich Y (2012) Top-k document retrieval in optimal time and linear space. In: SODA, Kyoto, pp 1066–1077
8. Navarro G, Thankachan SV (2014) New space/time tradeoffs for top-k document retrieval on sequences. Theor Comput Sci 542:83–97
9. Raman R, Raman V, Satti SR (2007) Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Trans Algorithms 3(4):43
10. Russo L, Navarro G, Oliveira AL (2011) Fully compressed suffix trees. ACM Trans Algorithms 7(4):53
11. Shah R, Sheng C, Thankachan SV, Vitter JS (2013) Top-k document retrieval in external memory. In: ESA, Sophia Antipolis, pp 803–814
12. Tsur D (2013) Top-k document retrieval in optimal space. Inf Process Lett 113(12): 440–443

Compressed Range Minimum Queries

Johannes Fischer
Technical University Dortmund, Dortmund, Germany

C

Keywords

Compressed data structures; Succinct data structures

Years and Authors of Summarized Original Work

2011; Fischer, Heun

Problem Definition

Given a static array A of n totally ordered objects, the range minimum query problem (RMQ problem) is to build a data structure $\mathcal{D}$ on A that allows us to answer efficiently subsequent online queries of the form "what is the position of a minimum element in the subarray ranging from i to j?" (We consider the *minimum*; all results hold for *maximum* as well.) Such queries are denoted by $\mathrm{RMQ}_A(i, j)$ and are formally defined by $\mathrm{RMQ}_A(i, j) = \mathrm{argmin}_{i \le k \le j}\{A[k]\}$ for an array $A[1, n]$ and indices $1 \le i \le j \le n$. In the succinct or compressed setting, the goal is to use as few *bits* as possible for $\mathcal{D}$, hopefully sublinear in the space needed for storing A itself. The space for A is denoted by $|A|$ and is $|A| = \Theta(n \log n)$ bits if A stores numbers from a universe of size $n^{\Theta(1)}$.

Indexing Versus Encoding Model

There are two variations of the problem, depending on whether the input array A is available at query time (*indexing model*) or not (*encoding model*). In the indexing model, some space for the data structure $\mathcal{D}$ can in principle be saved, as the query algorithm can substitute the "missing information" by consulting A when answering the

queries, and this is indeed what all indexing data structures make heavy use of. However, due to the need to access A at query time, the *total* space (which is $|A| + |\mathcal{D}|$ bits) will never be sublinear in the space needed for storing the array A itself.

This is different in the *encoding model*, where the data structure $\mathcal{D}$ must be built in a way such that the query algorithm can derive its answers *without* consulting A. Such encoding data structures are important when only the *positions* of the minima matter (and not the actual *values*) or when the access to A itself is slow.

Any encoding data structure $\mathcal{D}_\mathrm{E}$ is automatically also an indexing data structure; conversely, an indexing data structure $\mathcal{D}_\mathrm{I}$ can always be "converted" to an encoding data structure by storing $\mathcal{D}_\mathrm{I}$ plus (a copy of) A. Hence, differentiating between the two concepts only makes sense if there are indexing data structures that use less space than the best encoding data structures and if there exist encoding data structures which use space sublinear in $|A|$. Interestingly, for range minimum queries, exactly this is the case.

Model of Computation

All results assume the usual word RAM model of computation with word size $\Omega(\log n)$ bits.

Key Results

Table 1 summarizes the key results from [12] by showing the sizes of data structures for range minimum queries (left column). The first data structure is in the indexing model, and the last two are encoding data structures. The leading terms $(2n/c(n)$ bits with $O(c(n))$ query time in the indexing model and $2n$ bits for arbitrary query time in the encoding model) are optimal: in the encoding model, this is rather easy to see by establishing a bijection between the class of binary trees and the class of arrays with different answers for at least one RMQ [12], and in the indexing model, Brodal et al. [4] prove the lower bound. Particular emphasis is placed on the additional space needed for constructing the data structure (middle column), where it is important to use asymptotically less space than the final data structure.

Compressed Range Minimum Queries, Table 1 Data structures [12] for range minimum queries, where $|A|$ denotes the space of the (read-only) input array A. Construction space is *in addition* to the final space of the data structure. All data structures can be constructed in $O(n)$ time, and query time is $O(1)$ unless noted otherwise

Final space (bits)	Construction space	Comment
$\lvert A\rvert + \frac{2n}{c(n)} - \Theta\left(\frac{n \log\log n}{c(n) \log n}\right)$	$O(\log^3 n)$	Query time $O(c(n))$ for $c(n) = O(n^\varepsilon)$, $0 < \varepsilon < 1$
$2n + O(n \log\log n / \log n)$	$\lvert A\rvert + n + O\left(\frac{n \log\log n}{\log n}\right)$	Construction space improved to $\lvert A\rvert + o(n)$ [8]
$2n + O(n/\mathrm{polylog}\, n)$	$\lvert A\rvert + O(n)$	Using succinct data structures from [23] and [21]

Extensions

Surpassing the Lower Bound

Attempts have been made to break the lower bound for special cases. If A is compressible under the order-k empirical entropy measure $H_k(A)$, then also the leading term $2n/c(n)$ of the indexing data structure can be compressed to $nH_k(A)$ [12]. Other results (in both the indexing and the encoding model) exist for compressibility measures based on the number of *runs* in A [2]. Davoodi et al. [8] show that *random* input arrays can be encoded in expected $1.919n + o(n)$ bits for RMQs. All of the above results retain constant query times.

Top-k Queries

A natural generalization of RMQs is listing the k smallest (or largest) values in a query range (k needs only be specified at query time). In the indexing model, any RMQ structure with constant query time can be used to answer top-k queries in $O(k)$ time [16].

Recently, an increased interest in encoding data structures for top-k queries can be observed. For general k, Grossi et al. [15] show that $\Omega(n \log k)$ bits are needed for answering top-k queries. Therefore, interesting encodings can only exist if an upper limit κ on k is given at construction time. This lower bound is matched

asymptotically by an encoding data structure using $O(n \log \kappa)$ bits and $O(k)$ query time by Navarro et al. [22]. For the specific case $\kappa = 2$, Davoodi et al. [8] provide a lower bound of $2.656n - O(\log n)$ bits (using computer-assisted search) and also give an encoding data structure using at most $3.272n + o(n)$ bits supporting top-2 queries in $O(1)$ time.

Range Selection

Another generalization are queries asking for the k-th smallest (or largest) value in the query range (k is again part of the query). This problem is harder for nonconstant k, as Jørgensen and Larsen prove a lower bound of $\Omega(\log k / \log \log n)$ on the query time when using $O(n\text{polylog}n)$ words of space [17]. Also, the abovementioned space lower bounds on encoding top-k queries also apply to range selection. Again, Navarro et al. [22] give a matching upper bound: $O(n \log \kappa)$ bits suffice to answer queries asking for the k-largest element in a query range ($k \leq \kappa$) in $O(\log k / \log \log n)$ time. Note that this includes queries asking for the *median* in a query range.

Higher Dimensions

Indexing and encoding data structures for range minima also exist for higher-dimensional arrays (matrices) of total size N. Atallah and Yuan [25] show an (uncompressed) indexing data structure of size $O(2^d d! N)$ words with $O(3^d)$ query time, where d is the dimension of the underlying matrix A.

Tighter results exist in the two-dimensional case [1], where A is an $(m \times n)$ matrix consisting of $N = m \cdot n$ elements (w.l.o.g. assume $m \leq n$). In the indexing model, the currently best solution is a data structure of size $|A| + O(N/c)$ bits ($1 \leq c \leq n$) that answers queries in $O(c \log c \log^2 \log c)$ time [3], still leaving a gap between the highest lower bounds of $\Omega(c)$ query time for $O(N/c)$ bits of space [4].

In the encoding model, a lower bound of $\Omega(N \log m)$ bits exists [4], but the best data structure with constant query time achieves only $O(N \min\{m, \log n\})$ bits of space [4], which still leaves a gap unless $m = n^{\Omega(1)}$. However, Brodal et al. [5] *do* show an encoding using only $O(N \log m)$ bits, but nothing better than the trivial $O(N)$ can be said about its query time. Special cases for small (constant) values of m (e.g., optimal $5n + o(n)$ bits for $m = 2$) and also for random input arrays are considered by Golin et al. [14].

Further Extensions

The indexing technique [12] has been generalized such that a specific minimum (e.g., the position of the median of the minima) can be returned if the minimum in the query range is not unique [11]. Further generalizations include functions other than the "minimum" on the query range, e.g., median [17], mode [6], etc. RMQs have also been generalized to edge-weighted trees [9], where now a query specifies two nodes v and w, and a minimum-weight edge on the path from v to w is sought.

Applications

Data structures for RMQs have many applications. Most notably, the problem of preprocessing a tree for *lowest common ancestor* (LCA) queries is equivalent to the RMQ problem. In succinctly encoded trees (using balanced parentheses), RMQs can also be used to answer LCA queries in constant time; in this case, the RMQ structure is built on the virtual excess sequence of the parentheses and uses only $o(n)$ bits in addition to the parenthesis sequence [24]. Other applications of RMQs include document retrieval [20], succinct trees [21], compressed suffix trees [13], text-index construction [10], Lempel-Ziv text compression [e.g., 18], orthogonal range searching [19], and other kinds of range queries [7].

Cross-References

- Compressed Document Retrieval on String Collections
- Compressed Suffix Trees
- Compressed Tree Representations
- Document Retrieval on String Collections
- Lempel-Ziv Compression

- Lowest Common Ancestors in Trees
- Orthogonal Range Searching on Discrete Grids
- Rank and Select Operations on Bit Strings
- Suffix Trees and Arrays

Recommended Reading

1. Amir A, Fischer J, Lewenstein M (2007) Two-dimensional range minimum queries. In: Proceedings of the CPM, London, LNCS, vol 4580. Springer, pp 286–294
2. Barbay J, Fischer J, Navarro G (2012) LRM-trees: compressed indices, adaptive sorting, and compressed permutation. Theor Comput Sci 459:26–41
3. Brodal GS, Davoodi P, Lewenstein M, Raman R, Rao SS (2012) Two dimensional range minimum queries and Fibonacci lattices. In: Proceedings of the ESA, Ljubljana, LNCS, vol 7501. Springer, pp 217–228
4. Brodal GS, Davoodi P, Rao SS (2012) On space efficient two dimensional range minimum data structures. Algorithmica 63(4):815–830
5. Brodal GS, Brodnik A, Davoodi P (2013) The encoding complexity of two dimensional range minimum data structures. In: Proceedings of the ESA, Sophia Antipolis, LNCS, vol 8125. Springer, pp 229–240
6. Chan TM, Durocher S, Larsen KG, Morrison J, Wilkinson BT (2014) Linear-space data structures for range mode query in arrays. Theory Comput Syst 55(4):719–741
7. Chen KY, Chao KM (2004) On the range maximum-sum segment query problem. In: Proceedings of the ISAAC, Hong Kong, LNCS, vol 3341. Springer, pp 294–305
8. Davoodi P, Navarro G, Raman R, Rao SS (2014) Encoding range minima and range top-2 queries. Philos Trans R Soc A 372:20130,131
9. Demaine ED, Landau GM, Weimann O (2014) On Cartesian trees and range minimum queries. Algorithmica 68(3):610–625
10. Fischer J (2011) Inducing the LCP-array. In: Proceedings of the WADS, New York, LNCS, vol 6844. Springer, pp 374–385
11. Fischer J, Heun V (2010) Finding range minima in the middle: approximations and applications. Math Comput Sci 3(1):17–30
12. Fischer J, Heun V (2011) Space efficient preprocessing schemes for range minimum queries on static arrays. SIAM J Comput 40(2):465–492
13. Fischer J, Mäkinen V, Navarro G (2009) Faster entropy-bounded compressed suffix trees. Theor Comput Sci 410(51):5354–5364
14. Golin M, Iacono J, Krizanc D, Raman R, Rao SS (2011) Encoding 2d range maximum queries. In: Proceedings of the ISAAC, Yokohama, LNCS, vol 7074. Springer, pp 180–189
15. Grossi R, Iacono J, Navarro G, Raman R, Rao SS (2013) Encodings for range selection and top-k queries. In: Proceedings of the ESA, Sophia Antipolis, LNCS, vol 8125. Springer, pp 553–564
16. Hon WK, Shah R, Thankachan SV, Vitter JS (2014) Space-efficient frameworks for top-k string retrieval. J ACM 61(2):Article No. 9
17. Jørgensen AG, Larsen KG (2011) Range selection and median: tight cell probe lower bounds and adaptive data structures. In: Proceedings of the SODA, San Francisco. ACM/SIAM, pp 805–813
18. Kärkkäinen J, Kempa D, Puglisi SJ (2013) Lightweight Lempel-Ziv parsing. In: Proceedings of the SEA, Rome, LNCS, vol 7933. Springer, pp 139–150
19. Lewenstein M (2013) Orthogonal range searching for text indexing. In: Space-efficient data structures, streams, and algorithms, LNCS, vol 8066. Springer, Heidelberg, pp 267–302
20. Navarro G (2014) Spaces, trees, and colors: the algorithmic landscape of document retrieval on sequences. ACM Comput Surv 46(4):Article No. 52
21. Navarro G, Sadakane K (2014) Fully functional static and dynamic succinct trees. ACM Trans Algorithms 10(3):Article No. 16
22. Navarro G, Raman R, Satti SR (2014) Asymptotically optimal encodings for range selection. In: Proceedings of the FSTTCS, New Delhi. IBFI Schloss Dagstuhl, paper to be published
23. Pătraşcu M (2008) Succincter. In: Proceedings of the FOCS, Washington, DC. IEEE Computer Society, pp 305–313
24. Sadakane K (2007) Compressed suffix trees with full functionality. Theory Comput Syst 41(4):589–607
25. Yuan H, Atallah MJ (2010) Data structures for range minimum queries in multidimensional arrays. In: Proceedings of the SODA, Austin. ACM/SIAM, pp 150–160

Compressed Representations of Graphs

J. Ian Munro[1] and Patrick K. Nicholson[2]
[1]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada
[2]Department D1: Algorithms and Complexity, Max Planck Institut für Informatik, Saarbrücken, Germany

Keywords

Digraphs; Graph representations; k-page graphs; Partial orders; Planar graphs; Succinct data structures

Years and Authors of Summarized Original Work

1989; Jacobson

Problem Definition

The problem is to represent a graph of a given *size* and *type* (e.g., an n node planar graph) in a compressed form while still supporting efficient navigation operations. More formally, if $G = (V, E)$ is a graph of a given type χ, with n nodes and m edges, then represent G using $\lg|\chi| + o(\lg|\chi|)$ bits of space, and support a set of appropriate operations on the graph in constant time (assuming we can access $\Theta(\lg n)$ consecutive bits in one operation). This may not be possible; if not, then explore what trade-offs are possible. Data structures that achieve this space bound are called *succinct* [13]. To simplify the statement of results, we assume the graph G in question contains no self-loops, and we also restrict ourselves to the static case.

Key Results

Outerplanar, Planar, and k-Page Graphs

The area of succinct data structures was initiated by Jacobson [13], who presented a succinct representation of planar graphs. His approach was to decompose the planar graph into at most *four* one-page (or outerplanar) graphs by applying a theorem of Yannikakis [20]. Each one-page graph is then represented as a sequence of *balanced parentheses*: this representation extends naturally to k-pages for $k \geq 1$. Using this representation, it is straightforward to support the following operations efficiently:

- `Adjacent`(x, y): report whether there is an edge $(x, y) \in E$.
- `Neighbors`(x): report all vertices that are adjacent to vertex x.
- `Degree`(x): report the degree of vertex x.

Munro and Raman [16] improved Jacobson's balanced parenthesis representation, thereby improving the constant factor in the space bound for representing both k-page and planar graphs. We present Table 1 which compares a representative selection of the various succinct data structures for representing planar graphs. Subsequent simplifications to the representation of balanced parentheses have been presented; cf., [11, 17]. Barbay et al. [1] present results for larger values of k, as well as the case where the edges or vertices of the graph have labels.

The decomposition into four one-page graphs is not the only approach to representing static planar graphs. Chuang et al. [7] presented another encoding based on *canonical orderings* of a planar graph and represented the graph using a *multiple parentheses sequence* (a sequence of balanced parentheses of more than one type). Later, Chiang et al. [6] generalized the notion of canonical orderings to *orderly spanning trees*, yielding improved constant factors in terms of n and m. Gavoille and Hanusse [10] presented an alternate encoding scheme for k-page graphs that yields a trade-off based on the number of isolated nodes (connected components with one vertex). Further improvements have been presented by Chuang et al. [7], as well as Castelli Aleardi et al. [5], for the special case of planar triangulations.

Blandford et al. [2] considered unlabelled *separable graphs*. A separable graph is one that admits an $O(n^c)$ separator for $c < 1$. Their structure occupies $O(n)$ bits and performs all three query types optimally. Subsequently, Blelloch and Farzan [3] made the construction of Blandford et al. [2] succinct in the sense that, given a graph G from a separable class of graphs χ (e.g., the class of arbitrary planar graphs), their data structure represents G using $\lg|\chi| + o(n)$ bits. Interestingly, we need not even *know* the value of $\lg|\chi|$ in order to use this representation.

Arbitrary Directed Graphs, DAGs, Undirected Graphs, and Posets

We consider the problem of designing succinct data structures for *arbitrary* digraphs. In a directed graph, we refer to the set of vertices $\{y : (x, y) \in E\}$ as the *successors* of x and

Compressed Representations of Graphs, Table 1 Comparison of various succinct planar graph representations. The second column indicates whether the representation supports multigraphs. For the entries marked with a †, the query cost is measured in *bit* accesses. Notation: n is the number of vertices in G, m is the number of edges in G, i is the number of isolated vertices in G, ε is an arbitrary positive constant, $\tau = \min\{\lg k/\lg\lg m, \lg\lg k\}$, and H is the information theoretic lower space bound for storing a graph G drawn from a class of separable graphs

Type	Multi	Ref.	Space in bits	`Adjacent`(x, y)	`Neighbors`(x)	`Degree`(x)
k-page	N	[13]	$O(kn)$	$O(\lg n + k)$†	$O(\deg(x)\lg n + k)$†	$O(\lg n)$†
	N	[10]	$(2(m+i) + o(m+i))\lg k$	$O(\tau \lg k)$	$O(\deg(x)\tau)$	$O(1)$
	N	[1]	$2m\lg k + n + o(m\lg k)$	$O(\lg k \lg\lg k)$	$O(\deg(x)\lg\lg k)$	$O(1)$
	N	[1]	$(2 + \varepsilon)m\lg k + n + o(m\lg k)$	$O(\lg k)$	$O(\deg(x))$	$O(1)$
	Y	[16]	$2m + 2kn + o(kn)$	$O(k)$	$O(\deg(x) + k)$	$O(1)$
Planar	N	[13]	$O(n)$	$O(\lg n)$†	$O(\deg(x)\lg n)$†	$O(\lg n)$†
	N	[10]	$12n + 4i + o(n)$	$O(1)$	$O(\deg(x))$	$O(1)$
	N	[7]	$\frac{5}{3}m + (5+\varepsilon)n + o(n)$	$O(1)$	$O(\deg(x))$	$O(1)$
	N	[6]	$2m + 2n + o(m+n)$	$O(1)$	$O(\deg(x))$	$O(1)$
	Y	[16]	$2m + 8n + o(n)$	$O(1)$	$O(\deg(x))$	$O(1)$
	Y	[7]	$2m + (5+\varepsilon)n + o(n)$	$O(1)$	$O(\deg(x))$	$O(1)$
	Y	[6]	$2m + 3n + o(m+n)$	$O(1)$	$O(\deg(x))$	$O(1)$
Triangulation	N	[7]	$2m + n + o(n)$	$O(1)$	$O(\deg(x))$	$O(1)$
	Y	[7]	$2m + 2n + o(n)$	$O(1)$	$O(\deg(x))$	$O(1)$
Separable	N	[2]	$O(n)$	$O(1)$	$O(\deg(x))$	$O(1)$
	N	[3]	$H + o(n)$	$O(1)$	$O(\deg(x))$	$O(1)$

the set of vertices $\{y : (y, x) \in E\}$ as the *predecessors* of x.

It is well known that an arbitrary directed graph can be represented using $n \times n$ bits by storing its adjacency matrix. This representation supports the `Adjacency`(x, y) operation by probing a single bit in the table in constant time. On the other hand, we can represent the graph using $\Theta(m \lg n)$ bits using an adjacency list representation, such that the following operations can be supporting in constant time:

- `Successor`(x, i): list the ith successor of vertex x
- `Predecessor`(x, i): list the ith predecessor of vertex x

The information theoretic lower bound dictates that essentially $\lg \binom{n^2}{m}$ bits are necessary for representing an arbitrary digraph. By representing each row (resp. column) of the adjacency matrix using an *indexable dictionary* [19], we get a data structure that supports `Adjacency` and `Successor` (resp. `Predecessor`) queries in constant time and occupies $\lg \binom{n^2}{m} + o\left(\lg \binom{n^2}{m}\right)$ bits of space. Note that we can only support two of the three operations with this approach. Farzan and Munro [9] showed that if $\Theta(n^\varepsilon) \le m \le \Theta\left(n^{2-\varepsilon}\right)$ for some constant $\varepsilon > 0$, then $(1+\varepsilon')\binom{n^2}{m}$ bits are sufficient and required to support all three operations. In a more general setting, Golynski [12] had proven that the difficulty in supporting both `Successor` and `Predecessor` queries simultaneously using succinct space relates to the fact that they have the so-called *reciprocal property*. In other words, if the graph is not extremely sparse or dense, then it is impossible to support all three operations succinctly. On the other hand, if $m = o(n^\varepsilon)$ or $m = \Omega(n^2/\lg^{1-\varepsilon} n)$, for some constant $\varepsilon > 0$, then Farzan and Munro [9] showed that succinctness can be achieved while supporting the three operations.

Suppose G is a directed acyclic graph instead of an arbitrary digraph. In this case, one can exploit the fact that the graph is acyclic by ordering the vertices topologically. This ordering induces an adjacency matrix which is upper triangular. Exploiting this fact, when $\Theta(n^\varepsilon) \le$

$m \leq \Theta(n^{2-\varepsilon})$, the data structure of Farzan and Munro can support all three operations using $(1+\varepsilon)\lg\binom{\binom{n}{2}}{m}$ bits, for an arbitrary positive constant ε, which is optimal. If $m = o(n^{\epsilon})$ or $m = \Omega(n^2/\lg^{1-\varepsilon} n)$, for some constant $\varepsilon > 0$, then they achieve $\lg\binom{\binom{n}{2}}{m}(1+o(1))$ bits. By orienting the edges in an *undirected* graph so that they are directed toward the vertex with the with the larger label, this representation can also be used to support the operations `Adjacency` and `Neighbors` on an arbitrary undirected graph.

A partial order or *poset* is a directed acyclic graph with the additional transitivity constraint on the set of edges E: if (x, y) and (y, z) are present in E, then it is implied that $(x, z) \in E$. Farzan and Fischer [8] showed that a poset can be stored using $2nw(1+o(1)) + (1+\varepsilon)n\lg n$ bits of space, where w is the *width* of the poset – i.e., the length of the maximum antichain – and ε is an arbitrary positive constant. Their data structure supports `Adjacency` queries in constant time, as well as many other operations in time proportional to w. This matches a lower bound of Brightwell and Goodall [4] up to the additive $\varepsilon n \lg n$ term when n is sufficiently large relative to w. For an arbitrary poset, Kleitman and Rothschild showed that $n^2/4 + O(n)$ bits are sufficient and necessary by a constructive counting argument [14]. Munro and Nicholson [15,18] showed that there is a data structure that occupies $n^2/4 + o(n^2)$ bits, such that `Adjacency`, `Predecessor`, and `Successor` queries can be supported in constant time.

Cross-References

- Compressed Tree Representations
- Rank and Select Operations on Bit Strings
- Recursive Separator Decompositions for Planar Graphs

Recommended Reading

1. Barbay J, Castelli Aleardi L, He M, Munro JI (2012) Succinct representation of labeled graphs. Algorithmica 62(1–2):224–257
2. Blandford DK, Blelloch GE, Kash IA (2003) Compact representations of separable graphs. In: SODA, ACM/SIAM, pp 679–688
3. Blelloch GE, Farzan A (2010) Succinct representations of separable graphs. In: Amir A, Parida L (eds) CPM. Lecture notes in computer science, vol 6129. Springer, Berlin/New York, pp 138–150
4. Brightwell G, Goodall S (1996) The number of partial orders of fixed width. Order 13(4):315–337. doi:10.1007/BF00405592, http://dx.doi.org/10.1007/BF00405592
5. Castelli Aleardi L, Devillers O, Schaeffer G (2005) Succinct representation of triangulations with a boundary. In: Dehne FKHA, López-Ortiz A, Sack JR (eds) WADS. Lecture notes in computer science, vol 3608. Springer, Berlin/New York, pp 134–145
6. Chiang YT, Lin CC, Lu HI (2005) Orderly spanning trees with applications. SIAM J Comput 34(4):924–945
7. Chuang RCN, Garg A, He X, Kao MY, Lu HI (1998) Compact encodings of planar graphs via canonical orderings and multiple parentheses. In: Larsen KG, Skyum S, Winskel G (eds) ICALP. Lecture notes in computer science, vol 1443. Springer, Berlin/New York, pp 118–129
8. Farzan A, Fischer J (2011) Compact representation of posets. In: Asano T, Nakano SI, Okamoto Y, Watanabe O (eds) ISAAC. Lecture notes in computer science, vol 7074. Springer, Berlin/New York, pp 302–311
9. Farzan A, Munro JI (2013) Succinct encoding of arbitrary graphs. Theor Comput Sci 513:38–52
10. Gavoille C, Hanusse N (2008) On compact encoding of pagenumber. Discret Math Theor Comput Sci 10(3)
11. Geary RF, Rahman N, Raman R, Raman V (2006) A simple optimal representation for balanced parentheses. Theor Comput Sci 368(3):231–246
12. Golynski A (2009) Cell probe lower bounds for succinct data structures. In: Proceedings of the 20th annual ACM-SIAM symposium on discrete algorithms (SODA). Society for Industrial and Applied Mathematics, pp 625–634
13. Jacobson G (1989) Space-efficient static trees and graphs. 30th annual IEEE symposium on foundations of computer science, pp 549–554
14. Kleitman DJ, Rothschild BL (1975) Asymptotic enumeration of partial orders on a finite set. Trans Am Math Soc 205:205–220
15. Munro JI, Nicholson PK (2012) Succinct posets. In: Epstein L, Ferragina P (eds) ESA. Lecture notes in computer science, vol 7501. Springer, Berlin/New York, pp 743–754
16. Munro JI, Raman V (2001) Succinct representation of balanced parentheses and static trees. SIAM J Comput 31(3):762–776
17. Navarro G, Sadakane K (2014) Fully-functional static and dynamic succinct trees. ACM Trans Algorithms 10(3):article 16

18. Nicholson PK (2013) Space efficient data structures in the word-RAM and bitprobe models. PhD thesis, University of Waterloo
19. Raman R, Raman V, Rao SS (2007) Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Trans Algorithms 3(4)
20. Yannakakis M (1989) Embedding planar graphs in four pages. J Comput Syst Sci 38(1):36–67

Compressed Suffix Array

Djamal Belazzougui, Veli Mäkinen, and Daniel Valenzuela
Department of Computer Science, Helsinki Institute for Information Technology (HIIT), University of Helsinki, Helsinki, Finland

Keywords

Burrows-wheeler transform; FM-index; Self-indexing

Years and Authors of Summarized Original Work

2000, 2003; Sadakane
2000, 2005; Grossi, Vitter
2000, 2005; Ferragina, Manzini

Problem Definition

Given a *text string* $T = t_1 t_2 \dots t_n$ over an alphabet Σ of size σ, the suffix array $A[1,n]$ is a permutation of the interval $[1,n]$ that sorts the suffixes of T. More precisely, it satisfies $T[A[i],n] < T[A[i+1],n]$ for all $1 \leq i < n$, where "<" between strings is the lexicographical order. The *suffix array* is the canonical full-text index that allows to efficiently compute basic string matching queries on T.

The *compressed suffix array (CSA)* problem asks to replace A with a space-efficient data structure that is capable of efficiently computing $A[i]$.

If a CSA does not require T to operate, and is capable of efficiently answering substring queries on T, it is called a *self-index*, as it can be seen as a replacement of T itself. Typical queries required from such an index are the following:

- count(P): count how many times a given *pattern string* $P = p_1 p_2 \dots p_m$ occurs in T.
- locate(P): return the locations where P occurs in T.
- display(i, j): return $T[i, j]$.

Key Results

Ψ-Based CSAs

The first solution to the problem is by Grossi and Vitter [8], who exploit the regularities of the suffix array via the Ψ-function:

Definition 1 Given suffix array $A[1,n]$, *function* $\Psi : [1,n] \to [1,n]$ is defined so that, for all $1 \leq i \leq n$, $A[\Psi(i)] = A[i] + 1$. The exception is $A[1] = n$, in which case the requirement is that $A[\Psi(1)] = 1$ so that Ψ is a permutation.

The following lemma shows that Ψ is appealing to compression:

Lemma 1 *Given a text $T[1,n]$, its suffix array $A[1,n]$, and the corresponding function Ψ, it holds $\Psi(i) < \Psi(i+1)$ whenever $T_{A[i]} = T_{A[i+1]}$.*

Grossi and Vitter used a hierarchical decomposition of Ψ into $h = \lceil \log \log n \rceil$ levels. The piecewise increasing property of Ψ can be used to represent each level of Ψ in $\frac{1}{2} n \log \sigma$ bits [8]. By storing some sampled values of A in the bottom level, any $A[i]$ can be computed by traversing the hierarchical structure. Other tradeoffs are possible using different amount of levels. The following one involves the use of a constant number of levels:

Theorem 1 (inspired from [8]) *The* Compressed Suffix Array *of Grossi and Vitter supports retrieving $A[i]$ in $O(\log^{\epsilon} n)$ time using $(\frac{1}{\epsilon} n) \log \sigma + O(n \log \log \sigma)$ bits of space, for any $0 < \epsilon < 1$.*

As a consequence, simulating the classical binary searches [13] to find the range of suffix array containing all the occurrences of a pattern $P[1,m]$ in $T[1,n]$, can then be done in $O(m \log^{1+\epsilon} n)$ time.

Sadakane [16] shows how the above compressed suffix array can be converted into a self-index, and at the same time optimized it in several ways.

Sadakane represents both A and T using the full function Ψ, and a few extra structures. Imagine one wishes to compare P against $T[A[i],n]$. For the binary search, one needs to extract enough characters from $T[A[i],n]$ so that its lexicographical relation to P is clear. Retrieving character $T[A[i]]$, given i, is easy. Use a bit vector $F[1,n]$ marking the suffixes of $A[i]$ where the first character changes from that of $A[i-1]$. After preprocessing F for $rank$-queries, computing $j = \text{rank}_1(F,i)$ tells us that $T[A[i]] = c_j$, where c_j is the j-th smallest alphabet character. Once $T[A[i]] = c_j$ is determined this way, one needs to obtain the next character, $T[A[i]+1]$. But $T[A[i]+1] = T[A[\Psi(i)]]$, so one can simply move to $i' = \Psi(i)$ and keep extracting characters with the same method, as long as necessary. Note that at most $|P| = m$ characters suffice to decide a comparison with P. Thus the binary search is simulated in $O(m \log n)$ time.

Up to now, the space used is $n + o(n) + \sigma \log \sigma$ bits for F and Σ. Sadakane [16] gives an improved representation for Ψ using $O(nH_0 + n \log\log \sigma)$ bits, where H_0 is the *zeroth order entropy* of T.

Sadakane also shows how $A[i]$ can be retrieved, by plugging in the hierarchical scheme of Grossi and Vitter. He adds to the scheme the retrieval of the inverse $A^{-1}[j]$. This is used in order to retrieve arbitrary text substrings $T[p,r]$, by first applying $i = A^{-1}[p]$ and then continuing as before to retrieve $r-p+1$ first characters of suffix $T[A[i],n]$. This capability turns the compressed suffix array into self-index. The following bound is a modified version of Sadakane's CSA taken from [15]:

Theorem 2 *The* Compressed Suffix Array *of Sadakane is a self-index occupying* $\frac{1}{\epsilon} nH_0 + O(n \log\log \sigma)$ *bits, and supporting retrieval of values* $A[i]$ *and* $A^{-1}[j]$ *in* $O(\log^{\epsilon} n)$ *time, counting of pattern occurrences in* $O(m \log n)$ *time, and displaying any substring of* T *of length* ℓ *in* $O(\ell + \log^{\epsilon} n)$ *time. Here* $0 < \epsilon \leq 1$ *is an arbitrary constant.*

Grossi, Gupta, Vitter, and Foschini [6, 9] have improved the space requirement of compressed suffix arrays to depend on the *k-th order entropy* H_k of T. The idea behind this improvement is a more careful analysis of regularities captured by the Ψ-function when combined with the indexing capabilities of their new elegant data structure, *wavelet tree*. They obtain, among other results, the following tradeoff:

Theorem 3 (Grossi, Gupta, and Vitter [9]) *The* Compressed Suffix Array *of Grossi, Gupta, and Vitter is a self-index of size* $\frac{1}{\epsilon} nH_k + o(n \log \sigma)$ *bits, that supports* $A[i]$ *and* $A^{-1}[j]$ *in* $O(\log^{1+\epsilon} n/\epsilon)$ *time,* count(P) *in* $O(m \log \sigma + \log^{2+\epsilon} n/\epsilon)$ *time, and* display(i,j) *in* $O((j-i)/\log_\sigma n + \log^{1+\epsilon} n/\epsilon)$ *time. Here* $0 < \epsilon \leq 1$ *is an arbitrary constant,* $k \leq \alpha \log_\sigma n$ *for some constant* $0 < \alpha < 1$.

They also obtain an interesting special case:

Theorem 4 (Grossi, Gupta, and Vitter [9]) *The space optimized* Compressed Suffix Array *of Grossi, Gupta, and Vitter is a self-index of size* $nH_k + o(n \log \sigma)$ *bits, that supports* $A[i]$ *and* $A^{-1}[j]$ *in* $O(\log^2 n/\log\log n)$ *time,* count(P) *in* $O(m \log n \log \sigma + \log^3 n/\log\log n)$ *time, and* display(i,j) *in* $O((j-i)/\log \sigma + \log^2 n/\log\log n)$ *time. Here* $k \leq \alpha \log_\sigma n$ *for some constant* $0 < \alpha < 1$.

In the above results, value k must be fixed before building the indexes. Later, they notice that a simple coding of Ψ-values yields an nH_k-dependent bound without the need of fixing k beforehand [6].

FM-Index

A different solution to the problem (at least on the surface) is obtained by exploiting the connection of *Burrows-Wheeler Transform (BWT)* [2] and *Suffix Array* data structure

[13]. The BWT is formed by a permutation T^{bwt} of T defined as $T^{\text{bwt}}[i] = T[A[i] - 1]$ for $A[i] > 1$ and $T^{\text{bwt}}[i] = T[n]$ for $A[i] = 1$. Without lack of generality, one can assume that T ends with $T[n] = \$$ with \$ being distinct symbol smaller than other symbols in T. Then $T^{\text{bwt}}[1] = T[n-1]$.

A property of the BWT is that symbols having the same context (i.e., string following them in T) are consecutive in T^{bwt}. This makes it easy to compress T^{bwt} achieving space close to high-order empirical entropies [14].

Ferragina and Manzini [3] discovered a way to combine the compressibility of the BWT and the indexing properties of the suffix array. The structure is essentially a compressed representation of the BWT plus some small additional structures to make it searchable.

To retrieve the whole text from the structure (that is, to support display$(1, n)$), it is enough to invert the BWT. For this purpose, let us consider a table $LF[1, n]$ defined such that if $T[i]$ is permuted to $T^{\text{bwt}}[j]$ and $T[i-1]$ to $T^{\text{bwt}}[j']$ then $LF[j] = j'$. It is then immediate that T can be retrieved *backwards* by printing $\$ \cdot T^{\text{bwt}}[1] \cdot T^{\text{bwt}}[LF[1]] \cdot T^{\text{bwt}}[LF[LF[1]]] \ldots$.

To represent array LF space-efficiently, Ferragina and Manzini noticed that each $LF[i]$ can be expressed as follows:

Lemma 2 (Ferragina and Manzini [3]) *$LF[i] = C(c) + \text{rank}_c(i)$, where $c = T^{\text{bwt}}[i]$, $C(c)$ tells how many times symbols smaller than c appear in T^{bwt} and $\text{rank}_c(i)$ tells how many times symbol c appears in $T^{\text{bwt}}[1, i]$.*

It was later observed that LF is in fact the inverse of Ψ.

It also happens that the very same two-part expression of $LF[i]$ enables efficient count(P) queries. The idea is that if one knows the range of the suffix array, say $A[sp_i, ep_i]$, such that the suffixes $T[A[sp_i], n], T[A[sp_i + 1], n], \ldots, T[A[ep_i], n]$ are the only ones containing $P[i, m]$ as a prefix, then one can compute the new range $A[sp_{i-1}, ep_{i-1}]$ where the suffixes contain $P[i-1, m]$ as a prefix, as follows: $sp_{i-1} = C(P[i-1]) + \text{rank}_{P[i-1]}(sp_i - 1) + 1$ and $ep_{i-1} = C(P[i-1]) + \text{rank}_{P[i-1]}(ep_i)$. It is then enough to scan the pattern *backwards* and compute values $C()$ and $\text{rank}_c()$ $2m$ times to find out the (possibly empty) range of the suffix array where all the suffixes start with the complete P. Returning $ep_1 - sp_1 + 1$ solves the count(P) query without the need of having the suffix array available at all.

For locating each such occurrence $A[i]$, $sp_1 \leq i \leq ep_1$, one can compute the sequence i, $LF[i]$, $LF[LF[i]]$, ..., until $LF^k[i]$ is a *sampled suffix array position*; sampled positions can be marked in a bit vector B such that $B[LF^k[i]] = 1$ indicates that $\texttt{samples}[\text{rank}_1(B, LF^k[i])] = A[LF^k[i]]$, where `samples` is a compact array storing the sampled suffix array values. Then $A[i] = A[LF^k[i]] + k = \texttt{samples}[\text{rank}_1(B, LF^k[i])] + k$. A similar structure can be used to support display(i, j).

Values $C()$ can be stored trivially in a table of $\sigma \log_2 n$ bits. $T^{\text{bwt}}[i]$ can be computed in $O(\sigma)$ time by checking for which c is $\text{rank}_c(i) \neq \text{rank}_c(i-1)$. The suffix array sampling rate can be chosen as $s = \Theta(\log^{1+\epsilon} n)$ so that the samples require $o(n)$ bits. The real challenge is to preprocess the text for $\text{rank}_c()$ queries. The original proposal builds several small partial sum data structures on top of the compressed BWT, and achieves the following result:

Theorem 5 (Ferragina and Manzini [3]) *The* FM-Index (FMI) *is a* self-index *of size $5nH_k + o(n \log \sigma)$ bits that supports* count(P) *in $O(m)$ time,* locate(P) *in $O(\sigma \log^{1+\epsilon} n)$ time per occurrence, and* display(i, j) *in $O(\sigma(j - i + \log^{1+\epsilon} n))$ time. Here $\sigma = o(\log n / \log \log n)$, $k \leq \log_\sigma(n / \log n) - \omega(1)$, and $\epsilon > 0$ is an arbitrary constant.*

The original FM-Index has a severe restriction on the alphabet size. This has been removed in follow-up works. Conceptually, the easiest way to achieve a more alphabet-friendly instance of the FM-index is to build a wavelet tree [9] on T^{bwt}. It allows one to simulate a single $\text{rank}_c()$ query or to obtain $T^{\text{bwt}}[i]$ in $O(\log \sigma)$ time. Some later enhancements have improved the time

requirement, so as to obtain, for example, the following result:

Theorem 6 (Mäkinen and Navarro [11]) *The CSA problem can be solved using a so-called* Succinct Suffix Array (SSA)*, of size* $nH_0 + o(n \log \sigma)$ *bits that supports* count(P) *in* $O(m(1 + \log \sigma / \log\log n))$ *time,* locate(P) *in* $O(\log^{1+\epsilon} n\ (1 + \log \sigma / \log\log n))$ *time per occurrence, and* display(i, j) *in* $O((j - i + \log^{1+\epsilon} n)(1 + \log \sigma / \log\log n))$ *time. Here* $\sigma = o(n)$ *and* $\epsilon > 0$ *is an arbitrary constant.*

Ferragina et al. [4] developed a technique called *compression boosting* that finds an optimal partitioning of T^{bwt} such that, when one compresses each piece separately using its zero-order model, the result is proportional to the k-th order entropy. It was observed in [10] that a fixed block partitioning achieves the same result.

Compression boosting can be combined with the idea of SSA by building a wavelet tree separately for each piece and some additional structures in order to solve global $\text{rank}_c()$ queries from the individual wavelet trees:

Theorem 7 (Ferragina et al. [5]) *The CSA problem can be solved using a so-called* Alphabet-Friendly FM-Index (AF-FMI)*, of size* $nH_k + o(n \log \sigma)$ *bits, with the same time complexities and restrictions of SSA with* $k \leq \alpha \log_\sigma n$*, for any constant* $0 < \alpha < 1$.

A careful analysis [12] reveals that the space of the plain SSA is bounded by the same $nH_k + o(n \log \sigma)$ bits, making the boosting approach to achieve the same result unnecessary in theory. By plugging a better wavelet tree implementation [7], the space of Theorem 7 can be improved to $nH_k + o(n)$ bits.

The wavelet tree is space-efficient, but it cannot operate in time better than $O(1 + \frac{\log \sigma}{\log\log n})$. To achieve better performance, some other techniques must be used. One example, is the following fastest FM-index with dominant term nH_k in the space.

Theorem 8 (Belazzougui and Navarro [1]) *The CSA problem can be solved using an index of size* $nH_k + o(n \log \sigma)$*, that supports* count(P) *in* $O(m)$ *time,* locate(P) *in* $O(\log_\sigma n \log\log n)$ *time per occurrence, and* display(i, j) *in* $O((j - i) + \log_\sigma n \log\log n)$ *with* $k \leq \alpha \log_\sigma n$*,* $\sigma = O(n)$*, and* α *is any constant such that* $0 < \alpha < 1$.

Cross-References

- Burrows-Wheeler Transform
- Rank and Select Operations on Bit Strings
- Rank and Select Operations on Sequences
- Suffix Trees and Arrays
- Wavelet Trees

Recommended Reading

1. Belazzougui D, Navarro G (2011) Alphabet-independent compressed text indexing. In: ESA, Saarbrücken, pp 748–759
2. Burrows M, Wheeler D (1994) A block sorting lossless data compression algorithm. Technical report 124, Digital Equipment Corporation
3. Ferragina P, Manzini G (2005) Indexing compressed texts. J ACM 52(4):552–581
4. Ferragina P, Giancarlo R, Manzini G, Sciortino M (2005) Boosting textual compression in optimal linear time. J ACM 52(4):688–713
5. Ferragina P, Manzini G, Mäkinen V, Navarro G (2007) Compressed representations of sequences and full-text indexes. ACM Trans Algorithms 3(2):20
6. Foschini L, Grossi R, Gupta A, Vitter JS (2006) When indexing equals compression: experiments with compressing suffix arrays and applications. ACM Trans Algorithms 2(4):611–639
7. Golynski A, Raman R, Srinivasa Rao S (2008) On the redundancy of succinct data structures. In: SWAT, Gothenburg, pp 148–159
8. Grossi R, Vitter J (2006) Compressed suffix arrays and suffix trees with applications to text indexing and string matching. SIAM J Comput 35(2):378–407
9. Grossi R, Gupta A, Vitter J (2003) High-order entropy-compressed text indexes. In: Proceedings of the 14th annual ACM-SIAM symposium on discrete algorithms (SODA), Baltimore, pp 841–850
10. Kärkkäinen J, Puglisi SJ (2011) Fixed block compression boosting in fm-indexes. In: SPIRE, Pisa, pp 174–184
11. Mäkinen V, Navarro G (2005) Succinct suffix arrays based on run-length encoding. Nord J Comput 12(1):40–66
12. Mäkinen V, Navarro G (2008) Dynamic entropy-compressed sequences and full-text indexes. ACM Trans Algorithms 4(3):32

13. Manber U, Myers G (1993) Suffix arrays: a new method for on-line string searches. SIAM J Comput 22(5):935–948
14. Manzini G (2001) An analysis of the Burrows-Wheeler transform. J ACM 48(3):407–430
15. Navarro G, Mäkinen V (2007) Compressed full-text indexes. ACM Comput Surv 39(1): Article 2
16. Sadakane K (2003) New text indexing functionalities of the compressed suffix arrays. J Algorithms 48(2):294–313

Compressed Suffix Trees

Luís M.S. Russo
Departamento de Informática, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal
INESC-ID, Lisboa, Portugal

Keywords

Compressed index; Data compression; Enhanced suffix array; Longest common prefix; Range minimum query; Succinct data structure; Suffix link

Years and Authors of Summarized Original Work

2007; Sadakane
2009; Fischer, Mäkinen, Navarro
2010; Ohlebusch, Fischer, Gog
2011; Russo, Navarro, Oliveira

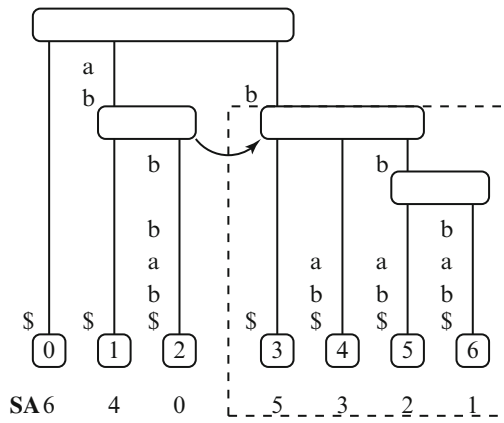
Problem Definition

The problem consists in representing suffix trees in main memory. The representation needs to support operations efficiently, using a reasonable amount of space.

Suffix trees were proposed by Weiner in 1973 [16]. Donald Knuth called them the "Algorithm of the Year." Their ubiquitous nature was quickly perceived and used to solve a myriad of string processing problems. The downside of this flexibility was the notorious amount of space necessary to keep it in main memory. A direct implementation is several times larger than the file it is indexing. Initial research into this matter discovered smaller data structures, sometimes by sacrificing functionality, namely, suffix arrays [6], directed acyclic word graphs [5], or engineered solutions.

A suffix tree is obtained from a sequence of characters T by considering all its suffixes. The suffixes are collated together by their common prefixes into a labeled tree. This means that suffixes that share a common prefix are united by that prefix and split only when the common prefix ends, i.e., in the first letter where they mismatch. A special terminator character \$ is placed at the end to force these mismatches for the small suffixes of T. The string depth of a node is the number of letters between the node and the root. Figure 1 shows the suffix tree of the string *abbbab*.

A viable representation needs to support several operations: tree navigation, such as finding a parent node, a child node, or a sibling node; labeling, such as reading the letters along a branch or using a letter to choose a child node; and indexing operations, such as determining a leaf's index or a node's string depth.



Compressed Suffix Trees, Fig. 1 Suffix tree of string *abbbab*, with the leaves numbered. The *arrow* shows the SLINK between nodes ab and b. Below we show the suffix array. The portion of the tree corresponding to node b and respective leaves interval is within a *dashed box*

Besides the usual tree topology, suffix trees contain suffix links. For a given node, the suffix link points to a second node. The string from the root to the second node can be obtained by removing the first letter of the string from the root to the first node. For example, the suffix link of node ab is node b. See Fig. 1.

An acceptable lower bound to represent a DNA sequence is $2n$ bits, i.e., $n \log \sigma$ bits (we use logarithms in base 2) for a text of size n with an alphabet of size σ. The size of the suffix array for such a sequence, on a 32-bit machine, is 16 times bigger than this bound. A space-engineered suffix tree would be 40 times larger. Succinct data structures are functional representations whose space requirements are close to the space needed to represent the data (i.e., close to $2n$ bits, in our example). If we consider the order-k statistical compressibility of the text, then the information-theory lower bound is even smaller, $nH_k + o(n)$ bits, where H_k is the empirical entropy of the text. Such representations require data compression techniques, which need to be made functional. The prospect of representing suffix trees within this space was significant.

Objective

Obtain a representation of a suffix tree that requires $n \log \sigma + o(n)$ bits, or even $nH_k + o(n)$ bits, or close, and that supports all operations efficiently.

Key Results

Classical Results

Suffix arrays [6] are a common alternative to suffix trees. They do not provide the same set of operations or the same time bounds, and still they require only 5 bytes per text character as opposed to the >10 bytes of suffix trees.

The suffix array stores the lexicographical order of the suffixes of T. Figure 1 shows the suffix array SA of our running example, i.e., the suffixes in the suffix tree are lexicographically ordered. Suffix arrays lack node information. Still they can represent nodes as an interval of suffixes, for example, node b corresponds to the interval $[3, 6]$. This mapping is injective, i.e., no two nodes can map to the same interval. Hence, a given interval corresponds to no more than one node. Some intervals do not correspond to any node, for example, $[4, 6]$ does not correspond to a node on the suffix tree. To determine which intervals are invalid and speed up navigation operations, the suffix array can be augmented with longest common prefix (LCP) information. It is enough to store the length of the LCP between consecutive suffixes. For an arbitrary pair of suffixes, the LCP value can be computed as a range minimum query (RMQ) on the corresponding leaf interval. For example, $\mathrm{LCP}(3, 5)$ can be computed as the minimum of $1, 1, 2$. The suffix array enhanced with LCP information [2] can now be used to emulate several algorithms that required suffix tree navigation.

Another approach is to reduce suffix tree redundancy, by factoring repeated structures. The following lemma is used to build directed acyclic word graphs.

Lemma 1 *If the sub-trees rooted at nodes v and v', of the suffix tree of T, have the same number of leaves and v' is the suffix link of v, then the sub-trees are isomorphic.*

Succinct Results

A fundamental component of compressed suffix trees is the underlying compressed index, which provides the suffix array information. Additionally these indexes provide support for ψ and LF, which are used to compute suffix links and backward search. In fact, ψ is the equivalent to suffix links over the suffix array, i.e., $\mathrm{SA}[\psi(i)] = \mathrm{SA}[i] + 1$. Moreover, LF is the inverse of ψ, i.e., $\mathrm{LF}(\psi(i)) = \psi(\mathrm{LF}(i)) = i$. There is a wide variety of these indexes, depending on the underlying data compression technique, namely, the Burrows-Wheeler transform, δ-coding, or Lempel-Ziv. For a complete survey on these indexes, consult the survey by Navarro and Mäkinen [7]. For our purposes, we consider the following index.

Theorem 1 *For a string T, over an alphabet of size* polylog n, *there exists a suffix array*

representation that requires $nH_k + o(n)$ *bits and computes* ψ, LF *and retrieves letters* $T[\text{SA}[i]]$ *in* $O(1)$ *time, while it obtains values* $\text{SA}[i]$ *in* $O(\log n \log\log n)$ *time.*

Sadakane was the first to combine a compressed suffix array with a succinct tree and a succinct representation of string depth information. The combination of these three ingredients leads to the first succinct representation of suffix trees, which set the basic structure for later developments.

Theorem 2 ([14]) *There is a compressed suffix tree representation that requires* $nH_k + 6n + o(n)$ *bits and supports child in* $O(\log^2 n \log\log n)$, *string depth and edge letters in* $O(\log n \log\log n)$ *time, and the remaining operations in* $O(1)$ *time.*

The $6n$ term is composed of $4n$ for the succinct tree and $2n$ to store LCP values. A tree can be represented succinctly as a sequence of balanced parentheses. A suffix tree contains at most $2n - 1$ nodes; since each node requires exactly 2 parentheses, this accounts for the $4n$ parcel. For example, the parentheses representation of the tree in Fig. 1 is `((0)((1)(2))((3)(4)((5)(6))));` the numbers correspond to the leaf indexes and are not part of the representation; only the parentheses are necessary. These parentheses are encoded with bits, set to 0 or 1, respectively.

We refer to the position of T in the suffix array as t, i.e., $\text{SA}[t] = 0$, in our running example $t = 2$. A technique used to store the SA values is to use the relation $\text{SA}[\psi(i)] = \text{SA}[i] + 1$. This means that, if ψ is supported, we can store the $\text{SA}[\psi^{il}(t)]$ values, with $l = \log n \log\log n$, and obtain the missing values in at most l steps. The resulting SA values require only $n/\log\log n$ bits to store. To encode the LCP of internal nodes, Sadakane used that $\text{LCP}(\psi(i), \psi(i)+1) \geq \text{LCP}(i, i+1) - 1$. Hence $\text{LCP}(\psi^k(i), \psi^k(i)+1) + k$ forms an increasing sequence of numbers, which can be encoded in at most $2n$ bits such that any element can be accessed with a "select" operation on the bits (find the position of the jth 1, which can be solved in constant time with an $o(n)$-bit extra index). Hence computing LCP requires determining k and subtracting it from the number in the sequence; this can be achieved with SA. Subsequent research focused on eliminating the $6n$ term. Fischer, Mäkinen, and Navarro obtained a smaller representation.

Theorem 3 ([4]) *There is a compressed suffix tree representation which, for any constant* $\epsilon > 0$, *requires* $nH_k(2\log(1/H_k) + (1/\epsilon) + O(1)) + o(n\log\sigma)$ *bits and supports all operations in* $O(\log^{\epsilon} n)$ *time, except level ancestor queries (LAQ) which need* $O(\log^{1+\epsilon} n)$ *time.*

This bound is obtained by compressing the differential LCP values in Sadakane's representation and discarding the parentheses representation. Instead it relies exclusively on range minimum queries over the LCP values. The next smaller value (NSV) and previous smaller value (PSV) operations are presented to replace the need to find matching closing or opening parentheses. Later, Fischer [3] further improved the speed at which ϵ vanishes.

Russo, Navarro, and Oliveira [13] obtained the smallest representation by showing that $\text{LCA}(\text{SLINK}(v), \text{SLINK}(v')) = \text{SLINK}(\text{LCA}(v, v'))$ holds for any nodes v and v'. $\text{LCA}(v, v')$ means the lowest common ancestor of nodes v and v', and $\text{SLINK}(v)$ means the suffix link of node v. This relation generalizes Lemma 1.

Theorem 4 ([13]) *There is a compressed suffix tree representation that requires* $nH_k + o(n)$ *bits and supports child in* $O(\log n(\log\log n)^2)$ *and the other operations in time* $O(\log n \log\log n)$.

The reduced space requirements are obtained by storing information about a few sampled nodes. Information for the remaining nodes is computed with the property above. Only the sampled nodes are stored in a parentheses representation and moreover string depth is stored only for these nodes. Although Theorem 4 obtains optimal space, the logarithmic time is significant, in theory and in practice. This limitation was

recently improved by Navarro and Russo [9] with a new approach to compare sampled nodes. They obtain $O(\text{polyloglog } n)$ time for all operations within the same space, except for child, which retains the previous time bound.

Applications

There is a myriad of practical applications for suffix trees. An extensive description can be found in the book by Gusfield [5]. Nowadays, most applications are related to bioinformatics, due to the string-like nature of DNA sequences. Most of these problems can be solved with compressed suffix trees and in fact can only be computed in reasonable time if the ever-increasing DNA database can be kept in main memory with suffix tree functionality.

Experimental Results

The results we have described provided a firm ground for representing suffix trees efficiently in compressed space. Still the goal was not a theoretical endeavor, since these data structures play a center role in the analysis of genomic data, among others. In practice, several aspects of computer architecture come into play, which can significantly impact the resulting performance. Different approaches can sacrifice space optimality to be orders of magnitude faster than the smaller variants.

Abeliuk, Cánovas, and Navarro [1] presented an exhaustive experimental analysis of existing CSTs. They obtained practical CSTs by implementing the PSV and NSV operations of Fischer, Mäkinen, and Navarro [4], with a range min-max tree [10]. Their CSTs covered a wide range in the space and time spectrum. Roughly, they need 8–12 bits per character (bpc) and perform the operations in microseconds. Further, practical variants considered reducing the $6n$ term of Sadakane's representation by using a single data structure that simultaneously provides RMQ/PSV/NSV. Ohlebusch and Gog [11] used only $2n + o(n)$ bits, obtaining around 10–12 bpc and operations in microseconds. Ohlebusch, Fischer, and Gog [12] also used this approach to obtain the same time performance of Theorem 2, within $3n$ extra bits instead of $6n$. This yields around 16 bpc and operations running in micro- to nanoseconds. An implementation of Theorem 4 [13] needed around 4 bpc but queries required milliseconds, although better performance is expected for the new version [9].

The proposal in [1] is also designed to adapt efficiently to highly repetitive sequence collections, obtaining 1–2 bpc and operations in milliseconds. Also for repetitive texts, Navarro and Ordóñez [8] obtained a speedup to microsecond operations, with a slight space increase, 1–3 bpc, by representing the parenthesis topology explicitly but in grammar-compressed form.

URLs to Code and Data Sets

An implementation of Sadakane's compressed suffix tree [14] is available from the SuDS group at http://www.cs.helsinki.fi/group/suds/cst/. Implementation details and engineering decisions are described by Välimäki, Gerlach, Dixit, and Mäkinen [15]. The compressed suffix tree of Abeliuk, Cánovas, and Navarro [1] is available in the libcds library at https://github.com/fclaude/libcds.

An alternative implementation of Sadakane's CST is available in the Succinct Data Structure Library at https://github.com/simongog/sdsl-lite. The SDSL also contains an implementation of the CST++ [12].

The Pizza and Chili site contains a large and varied dataset to test compressed indexes, http://pizzachili.dcc.uchile.cl.

Cross-References

- Burrows-Wheeler Transform
- Compressed Range Minimum Queries
- Compressed Suffix Array
- Compressed Tree Representations

- Grammar Compression
- Rank and Select Operations on Bit Strings
- String Matching
- Suffix Trees and Arrays

Recommended Reading

1. Abeliuk A, Cánovas R, Navarro G (2013) Practical compressed suffix trees. Algorithms 6(2):319–351
2. Abouelhoda MI, Kurtz S, Ohlebusch E (2004) Replacing suffix trees with enhanced suffix arrays. J Discr Algorithms 2(1):53–86
3. Fischer J (2010) Wee LCP. Inf Process Lett 110(8–9):317–320
4. Fischer J, Mäkinen V, Navarro G (2009) Faster entropy-bounded compressed suffix trees. Theor Comput Sci 410(51):5354–5364
5. Gusfield D (1997) Algorithms on strings, trees and sequences computer science and computational biology. Cambridge University Press, Cambridge/New York
6. Manber U, Myers G (1993) Suffix arrays: a new method for on-line string searches. SIAM J Comput 22(5):935–948
7. Navarro G, Mäkinen V (2007) Compressed full-text indexes. ACM Comput Surv 39(1): article 2
8. Navarro G, Ordóñez A (2014) Faster compressed suffix trees for repetitive text collections. In: Proceedings of the 13th international symposium on experimental algorithms (SEA), Copenhagen. LNCS 8504, pp 424–435
9. Navarro G, Russo L (2014) Fast fully-compressed suffix trees. In: Proceedings of the 24th data compression conference (DCC), Snowbird, pp 283–291
10. Navarro G, Sadakane K (2014) Fully-functional static and dynamic succinct trees. ACM Trans Algorithms 10(3):article 16
11. Ohlebusch E, Gog S (2009) A compressed enhanced suffix array supporting fast string matching. In: String processing and information retrieval, Saariselkä. Springer, pp 51–62
12. Ohlebusch E, Fischer J, Gog S (2010) CST++. In: String processing and information retrieval, Los Cabos. Springer, pp 322–333
13. Russo L, Navarro G, Oliveira A (2011) Fully-compressed suffix trees. ACM Trans Algorithms (TALG) 7(4):article 53, 35p
14. Sadakane K (2007) Compressed suffix trees with full functionality. Theory Comput Syst 41(4):589–607
15. Välimäki N, Gerlach W, Dixit K, Mäkinen V (2007) Engineering a compressed suffix tree implementation. In: Experimental algorithms, Rome. Springer, pp 217–228
16. Weiner P (1973) Linear pattern matching algorithms. In: IEEE conference record of 14th annual symposium on switching and automata theory, 1973. SWAT'08. IEEE, pp 1–11. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4569717

Compressed Text Indexing

Veli Mäkinen[1] and Gonzalo Navarro[2]
[1]Department of Computer Science, Helsinki Institute for Information Technology (HIIT), University of Helsinki, Helsinki, Finland
[2]Department of Computer Science, University of Chile, Santiago, Chile

Keywords

Compressed full-text indexing; Self-indexing; Space-efficient text indexing

Years and Authors of Summarized Original Work

2005; Ferragina, Manzini

Problem Definition

Given a *text string* $T = t_1 t_2 \ldots t_n$ over an alphabet Σ of size σ, the *compressed text indexing (CTI)* problem asks to *replace* T with a space-efficient data structure capable of efficiently answering basic string matching and substring queries on T. Typical queries required from such an index are the following:

- *count*(P): count how many times a given *pattern string* $P = p_1 p_2 \ldots p_m$ occurs in T.
- *locate*(P): return the locations where P occurs in T.
- *display*(i, j): return $T[i, j]$.

Key Results

An elegant solution to the problem is obtained by exploiting the connection of *Burrows-Wheeler Transform (BWT)* [1] and *Suffix Array* data structure [9]. The suffix array $SA[1, n]$ of T is the permutation of text positions $(1 \ldots n)$ listing the *suffixes* $T[i, n]$ in lexicographic order. That is, $T[SA[i], n]$ is the ith smallest suffix. The BWT

is formed by (1) a permutation T^{bwt} of T defined as $T^{\text{bwt}}[i] = T[SA[i]-1]$, where $T[0] = T[n]$, and (2) the number $i^* = SA^{-1}[1]$.

A property of the BWT is that symbols having the same context (i.e., string following them in T) are consecutive in T^{bwt}. This makes it easy to compress T^{bwt} achieving space close to high-order empirical entropies [10]. On the other hand, the suffix array is a versatile text index, allowing for example $O(m \log n)$ time counting queries (using two binary searches on SA) after which one can locate the occurrences in optimal time.

Ferragina and Manzini [3] discovered a way to combine the compressibility of the BWT and the indexing properties of the suffix array. The structure is essentially a compressed representation of the BWT plus some small additional structures to make it searchable.

We first focus on retrieving arbitrary substrings from this compressed text representation, and later consider searching capabilities. To retrieve the whole text from the structure (that is, to support *display*$(1, n)$), it is enough to invert the BWT. For this purpose, let us consider a table $LF[1, n]$ defined such that if $T[i]$ is permuted to $T^{\text{bwt}}[j]$ and $T[i-1]$ to $T^{\text{bwt}}[j']$ then $LF[j] = j'$. It is then immediate that T can be retrieved *backwards* by printing $T^{\text{bwt}}[i^*] \cdot T^{\text{bwt}}[LF[i^*]] \cdot T^{\text{bwt}}[LF[LF[i^*]]] \ldots$ (by definition $T^{\text{bwt}}[i^*]$ corresponds to $T[n]$).

To represent array LF space-efficiently, Ferragina and Manzini noticed that each $LF[i]$ can be expressed as follows:

Lemma 1 (Ferragina and Manzini [3]) *$LF[i] = C(c) + rank_c(i)$, where $c = T^{\text{bwt}}[i]$, $C(c)$ tells how many times symbols smaller than c appear in T^{bwt} and $rank_c(i)$ tells how many times symbol c appears in $T^{\text{bwt}}[1, i]$.*

General *display*(i, j) queries rely on a regular sampling of the text. Every text position of the form $j' \cdot s$, being s the sampling rate, is stored together with $SA^{-1}[j' \cdot s]$, the suffix array position pointing to it. To solve *display*(i, j) we start from the smallest sampled text position $j' \cdot s > j$ and apply the BWT inversion procedure starting with $SA^{-1}[j' \cdot s]$ instead of i^*. This gives the characters in reverse order from $j' \cdot s - 1$ to i, requiring at most $j - i + s$ steps.

It also happens that the very same two-part expression of $LF[i]$ enables efficient $count(P)$ queries. The idea is that if one knows the range of the suffix array, say $SA[sp_i, ep_i]$, such that the suffixes $T[SA[sp_i], n], T[SA[sp_i + 1], n], \ldots, T[SA[ep_i], n]$ are the only ones containing $P[i, m]$ as a prefix, then one can compute the new range $SA[sp_{i-1}, ep_{i-1}]$ where the suffixes contain $P[i-1, m]$ as a prefix, as follows: $sp_{i-1} = C(P[i-1]) + rank_{P[i-1]}(sp_i - 1) + 1$ and $ep_{i-1} = C(P[i-1]) + rank_{P[i-1]}(ep_i)$. It is then enough to scan the pattern *backwards* and compute values $C()$ and $rank_c()$ $2m$ times to find out the (possibly empty) range of the suffix array where all the suffixes start with the complete P. Returning $ep_1 - sp_1 + 1$ solves the $count(P)$ query without the need of having the suffix array available at all.

For locating each such occurrence $SA[i]$, $sp_1 \leq i \leq ep_1$, one can compute the sequence i, $LF[i]$, $LF[LF[i]]$, ..., until $LF^k[i]$ is a sampled suffix array position and thus it is explicitly stored in the sampling structure designed for *display*(i, j) queries. Then $SA[i] = SA[LF^k[i]] + k$. As we are virtually moving sequentially on the text, we cannot do more than s steps in this process.

Now consider the space requirement. Values $C()$ can be stored trivially in a table of $\sigma \log_2 n$ bits. $T^{\text{bwt}}[i]$ can be computed in $O(\sigma)$ time by checking for which c is $rank_c(i) \neq rank_c(i-1)$. The sampling rate can be chosen as $s = \Theta(\log^{1+\epsilon} n)$ so that the samples require $o(n)$ bits. The only real challenge is to preprocess the text for $rank_c()$ queries. This has been a subject of intensive research in recent years and many solutions have been proposed. The original proposal builds several small partial sum data structures on top of the compressed BWT, and achieves the following result:

Theorem 2 (Ferragina and Manzini [3]) *The CTI problem can be solved using a so-called FM-Index (FMI), of size $5nH_k + o(n \log \sigma)$ bits, that supports $count(P)$ in $O(m)$ time, $locate(P)$ in $O(\sigma \log^{1+\epsilon} n)$ time per occurrence, and*

display(i, j) in $O(\sigma(j-i+\log^{1+\epsilon} n))$ *time. Here* H_k *is the k th order empirical entropy of T,* $\sigma = o(\log n/\log\log n)$, $k \leq \log_\sigma(n/\log n)$, $-\omega(1)$, *and* $\epsilon > 0$ *is an arbitrary constant.*

The original FM-Index has a severe restriction on the alphabet size. This has been removed in follow-up works. Conceptually, the easiest way to achieve a more alphabet-friendly instance of the FM-index is to build a *wavelet tree* [5] on T^{bwt}. This is a binary tree on Σ such that each node v handles a subset $S(v)$ of the alphabet, which is split among its children. The root handles Σ and each leaf handles a single symbol. Each node v encodes those positions i so that $T^{\text{bwt}}[i] \in S(v)$. For those positions, node v only stores a bit vector telling which go to the left, which to the right. The node bit vectors are preprocessed for constant time $rank_1()$ queries using $o(n)$-bit data structures [6, 12]. Grossi et al. [4] show that the wavelet tree built using the encoding of [12] occupies $nH_0 + o(n\log\sigma)$ bits. It is then easy to simulate a single $rank_c()$ query by $\log_2\sigma$ $rank_1()$ queries. With the same cost one can obtain $T^{\text{bwt}}[i]$. Some later enhancements have improved the time requirement, so as to obtain, for example, the following result:

Theorem 3 (Mäkinen and Navarro [7]) *The CTI problem can be solved using a so-called Succinct Suffix Array (SSA), of size* $nH_0 + o(n\log\sigma)$ *bits, that supports* $count(P)$ *in* $O(m(1+\log\sigma/\log\log n))$ *time,* $locate(P)$ *in* $O(\log^{1+\epsilon} n\log\sigma/\log\log n)$ *time per occurrence, and display(i, j) in* $O((j-i+\log^{1+\epsilon} n)\log\sigma/\log\log n)$ *time. Here* H_0 *is the zero-order entropy of T,* $\sigma = o(n)$, *and* $\epsilon > 0$ *is an arbitrary constant.*

Ferragina et al. [2] developed a technique called *compression boosting* that finds an optimal partitioning of T^{bwt} such that, when one compresses each piece separately using its zero-order model, the result is proportional to the kth order entropy. This can be combined with the idea of SSA by building a wavelet tree separately for each piece and some additional structures in order to solve global $rank_c()$ queries from the individual wavelet trees:

Theorem 4 (Ferragina et al. [4]) *The CTI problem can be solved using a so-called Alphabet-Friendly FM-Index (AF-FMI), of size* $nH_k + o(n\log\sigma)$ *bits, with the same time complexities and restrictions of SSA with* $k \leq \alpha\log_\sigma n$, *for any constant* $0 < \alpha < 1$.

A very recent analysis [8] reveals that the space of the plain SSA is bounded by the same $nH_k + o(n\log\sigma)$ bits, making the boosting approach to achieve the same result unnecessary in theory. In practice, implementations of [4, 7] are superior by far to those building directly on this simplifying idea.

Applications

Sequence analysis in Bioinformatics, search and retrieval on oriental and agglutinating languages, multimedia streams, and even structured and traditional database scenarios.

URL to Code and Data Sets

Site Pizza-Chili http://pizzachili.dcc.uchile.cl or http://pizzachili.di.unipi.it contains a collection of standardized library implementations as well as data sets and experimental comparisons.

Cross-References

- Burrows-Wheeler Transform
- Compressed Suffix Array
- Text Indexing

Recommended Reading

1. Burrows M, Wheeler D (1994) A block sorting lossless data compression algorithm. Technical report 124, Digital Equipment Corporation
2. Ferragina P, Giancarlo R, Manzini G, Sciortino M (2005) Boosting textual compression in optimal linear time. J ACM 52(4):688–713
3. Ferragina P, Manzini G (2005) Indexing compressed texts. J ACM 52(4):552–581
4. Ferragina P, Manzini G, Mäkinen V, Navarro G (2007) Compressed representation of sequences and

full-text indexes. ACM Trans Algorithms 3(2):Article 20
5. Grossi R, Gupta A, Vitter J (2003) High-order entropy-compressed text indexes. In: Proceedings of the 14th annual ACM-SIAM symposium on discrete algorithms (SODA), pp 841–850
6. Jacobson G (1989) Space-efficient static trees and graphs. In: Proceedings of the 30th IEEE symposium on foundations of computer science (FOCS), pp 549–554
7. Mäkinen V, Navarro G (2005) Succinct suffix arrays based on run-length encoding. Nord J Comput 12(1):40–66
8. Mäkinen V, Navarro G (2006) Dynamic entropy-compressed sequences and full-text indexes. In: Proceedings of the 17th annual symposium on combinatorial pattern matching (CPM). LNCS, vol 4009. Extended version as TR/DCC-2006-10, Department of Computer Science, University of Chile, July 2006, pp 307–318
9. Manber U, Myers G (1993) Suffix arrays: a new method for on-line string searches. SIAM J Comput 22(5):935–948
10. Manzini G (2001) An analysis of the Burrows-Wheeler transform. J ACM 48(3):407–430
11. Navarro G, Mäkinen V (2007) Compressed full-text indexes. ACM Comput Surv 39(1):Article 2
12. Raman R, Raman V, Rao S (2002) Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In: Proceedings of the 13th annual ACM-SIAM symposium on discrete algorithms (SODA), pp 233–242

Compressed Tree Representations

Gonzalo Navarro[1] and Kunihiko Sadakane[2]
[1]Department of Computer Science, University of Chile, Santiago, Chile
[2]Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan

Keywords

Balanced parentheses; Ordered tree; Succinct data structure

Years and Authors of Summarized Original Work

1989; Jacobson
2001; Munro, Raman
2005; Benoit, Demaine, Munro, Raman, S. Rao
2014; Navarro, Sadakane

Problem Definition

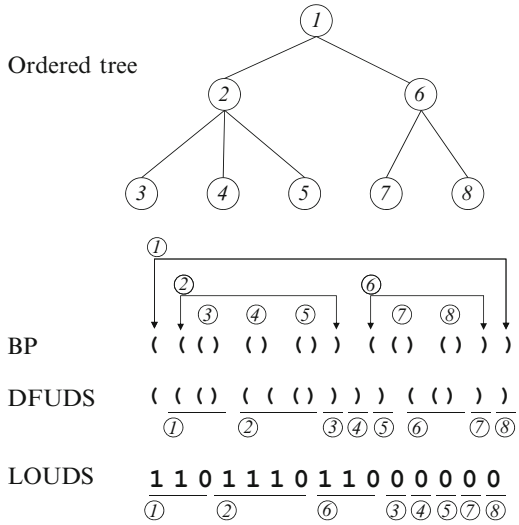
The problem is, given a tree, to encode it compactly so that basic operations on the tree are done quickly, preferably in constant time for static trees. Here, we consider the most basic class of trees: rooted ordered unlabeled trees. The information-theoretic lower bound for representing an n-node ordered tree is $2n - o(n)$ bits because there are $\binom{2n-2}{n-1}/n$ different trees. Therefore, the aim is to encode an ordered tree in $2n + o(n)$ bits including auxiliary data structures so that basic operations are done quickly. We assume that the computation model is the $\Theta(\log n)$-bit word RAM, that is, memory access for consecutive $\Theta(\log n)$ bits and arithmetic and logical operations on two $\Theta(\log n)$-bit integers are done in constant time.

Preliminaries

Let X be a string on alphabet $\mathcal{A}$. The number of occurrences of $c \in \mathcal{A}$ in $X[1 \ldots i]$ is denoted by $rank_c(X, i)$, and the position of j-th c from the left is denoted by $select_c(j)$ (with $select_c(0) = 0$). For binary strings ($|\mathcal{A}| = 2$) of length n, *rank* and *select* are computed in constant time using an $n + o(n)$-bit data structure [4]. Let us define for simplicity $prev_c(i) = select_c(rank_c(i-1))$ and $next_c(i) = select_c(rank_c(i)+1)$ the position of the c preceding and following, respectively, position i in X.

Key Results

Basically, there are three representations of ordered trees: LOUDS (Level-Order Unary Degree Sequence) [11], DFUDS (Depth-First Unary Degree Sequence) [2], and BP (Balanced Parenthesis sequence) [16]. An example is shown in Fig. 1. All these representations are succinct, that is, of $2n + o(n)$ bits. However, their functionality is slightly different.



Compressed Tree Representations, Fig. 1 Succinct representations of trees

LOUDS

In LOUDS representation, the degree of each node is encoded by a unary representation, that is, a node with d children is encoded in d ones, followed by a zero. Codes for the nodes are stored in level order: the root node is encoded first, then its children are encoded from left to right, all the nodes at depth 2 are encoded next, and so on. Let L be the LOUDS representation of a tree. Then L is a 0,1-string of length $2n-1$. Tree navigation operations are expressed by *rank* and *select* operations. The i-th node in level order is represented by $select_0(i-1)+1$. The operations $isleaf(i)$, $parent(i)$, $first\_child(i)$, $last\_child(i)$, $next\_sibling(i)$, $prev\_sibling(i)$, $degree(i)$, $child(i,q)$, and $child\_rank(i)$ (see Table 1) are computed in constant time using *rank* and *select* operations, for example, $degree(i) = next_0(i-1)-i$, $child(i,q) = select_0(rank_1(i-1+q))+1$, and $parent(i) = prev_0(select_1(rank_0(i-1)))+1$. However, because nodes are stored in level order, other operations cannot be done efficiently, such as $depth(i)$, $subtree\_size(i)$, $lca(i,j)$, etc. A merit of the LOUDS representation is that in practice it is fast and simple to implement because all the operations are done by only *rank* and *select* operations.

BP Representation

In BP representation, the tree is traversed in depth-first order, appending an open parenthesis "(" to the sequence when we reach a node and a closing parenthesis ")" when we leave it. These parentheses are represented with the bits 1 and 0, respectively. The result is a sequence of $2n$ parentheses that is balanced: for any open (resp. close) parenthesis, there is a matching close (resp. open) parenthesis to the right (resp. left), so that the areas between two pairs of matching parentheses either nest or are disjoint. Each node is identified with the position of its open parenthesis.

Munro and Raman [16] showed how to implement operations $findclose(i)$, $findopen(i)$, and $enclose(i)$ in constant time and $2n+o(n)$ bits in total. Later, Geary et al. [8] considerably simplified the solutions. With those operations and *rank* and *select* support, many operations in Table 1 are possible. For example, $pre\_rank(i) = rank_1(i)$, $pre\_select(j) = select_1(j)$, $isleaf(i)$ iff there is a 0 at position $i+1$, $isancestor(i,j)$ if $i \le j \le findclose(i)$, $depth(i) = rank_1(i) - rank_0(i)$, $parent(i) = enclose(i)$, $first\_child(i) = i+1$, $next\_sibling(i) = findclose(i)+1$, $subtree\_size(i) = (findclose(i)-i+1)/2$, etc. Operations lca, $height(i)$, and $deepest\_node(i)$ could be added with additional structures for range minimum queries, $rmqi(i,j)$, in $o(n)$ further bits [20]. Some other operations can be also supported in constant time by adding different additional structures. Lu and Yeh [14] gave $o(n)$-bit data structures for $degree(i)$, $child(i,q)$, and $child\_rank(i)$. Geary et al. [9] gave $o(n)$-bit data structures for $LA(i,d)$. However, these extra structures are complicated and add considerably extra space in practice.

DFUDS

In DFUDS representation, nodes are also encoded by a unary representation of their degrees, but stored in depth-first order. The bits 1 and 0 are interpreted as open parenthesis "(" and close parenthesis ")", respectively. By adding a dummy open parenthesis at the beginning, the DFUDS sequence becomes balanced. Each node is identified with the first position of its unary description.

Compressed Tree Representations, Table 1 Operations supported by the data structure of [19]. The time complexities are for the dynamic case; in the static case, all operations take $\mathcal{O}(1)$ time. The first group, of basic operations, is used to implement the others, but could have other uses

Operation	Description	Time complexity	
		Variant 1	Variant 2
inspect(i)	$P[i]$	$\mathcal{O}(\frac{\log n}{\log\log n})$	
findclose(i) / *findopen*(i)	Position of parenthesis matching $P[i]$		
enclose(i)	Position of tightest open parent. enclosing i		
rmqi(i,j) / *RMQi*(i,j)	Position of min/max excess value in range $[i,j]$		
pre_rank(i) / *post_rank*(i)	Preorder/postorder rank of node i	$\mathcal{O}(\frac{\log n}{\log\log n})$	
pre_select(i) / *post_select*(i)	The node with preorder/postorder i		
isleaf(i)	Whether $P[i]$ is a leaf		
isancestor(i,j)	Whether i is an ancestor of j		
depth(i)	Depth of node i		
parent(i)	Parent of node i		
first_child(i) / *last_child*(i)	First/last child of node i		
next_sibling(i) / *prev_sibling*(i)	Next/previous sibling of node i		
subtree_size(i)	Number of nodes in the subtree of node i		
lca(i,j)	The lowest common ancestor of two nodes i,j		
deepest_node(i)	The (first) deepest node in the subtree of i		
height(i)	The height of i (distance to its deepest node)		
in_rank(i)	Inorder of node i		
in_select(i)	Node with inorder i		
leaf_rank(i)	Number of leaves to the left of leaf i		
leaf_select(i)	i-th leaf		
lmost_leaf(i) / *rmost_leaf*(i)	Leftmost/rightmost leaf of node i		
LA(i,d)	Ancestor j of i s.t. *depth*$(j) =$ *depth*$(i) - d$	$\mathcal{O}(\log n)$	
level_next(i) / *level_prev*(i)	Next/previous node of i in BFS order		
level_lmost(d) / *level_rmost*(d)	Leftmost/rightmost node with depth d		
degree(i)	$q =$ number of children of node i	$\mathcal{O}(\frac{q\log n}{\log\log n})$	$\mathcal{O}(\log n)$
child(i,q)	q-th child of node i		
child_rank(i)	$q =$ number of siblings to the left of node i		
insert(i,j)	Insert node given by matching parent at i and j	$\mathcal{O}(\frac{\log n}{\log\log n})$	$\mathcal{O}(\log n)$
delete(i)	Delete node i		

A merit of using DFUDS is that it retains many operations supported in BP, but *degree*(i), *child*(i,q), and *child_rank*(i) are done in constant time by using only the basic operations supported by Munro and Raman [16]. For example, *degree*$(i) =$ *next*$_0(i-1) - i$, *child*$(i,q) =$ *findclose*$(i +$ *degree*$(i) - q) + 1$, and *parent*$(i) =$ *prev*$_0($*findopen*$(i-1)) + 1$. Operation *lca*(i,j) can also be computed using *rmqi*(i,j) [12].

Another important feature of DFUDS is that some trees can be represented in less than $2n$ bits. The number of ordered trees having n_i nodes with degree i ($i = 0,1,\ldots,n-1$) is $\frac{1}{n}\binom{n}{n_0 n_1 \cdots n_{n-1}}$ if $\sum_{i\geq 0} n_i(i-1) = -1$ and 0 otherwise (i.e., there are no trees satisfying the condition). Jansson et al. [12] proposed a compression algorithm for DFUDS sequences that achieves the lower bound $\lg\left(\frac{1}{n}\binom{n}{n_0 n_1 \cdots n_{n-1}}\right) +$

$o(n)$ bits. For example, a full binary tree, that is, a tree in which all internal nodes have exactly two children, is encoded in n bits.

A demerit of DUFDS is that computing $depth(i)$ and $LA(i,d)$ is complicated, though it is possible in constant time [12].

Fully Functional BP Representation

Navarro and Sadakane [19] proposed a data structure for ordered trees using the BP representation [16]. The data structure is called *range min-max tree*. Let P be the BP sequence. For each entry $P[i]$ of the sequence, we define its *excess value* as the number of open parentheses minus the number of close parentheses in $P[1,i]$. If $P[i]$ is an open parenthesis, its excess value is equal to $depth(i)$. Let $E[i]$ denote the excess value for $P[i]$. With the range min-max tree and small additional data structures, they support in constant time the operations $fwd\_search(i,d) = \min\{\{j > i | E[j] = E[i] + d\} \cup \{n+1\}\}$, $bwd\_search(i,d) = \max\{\{j < i | E[j] = E[i] + d\} \cup \{0\}\}$, and $rmqi(i,j) = \mathrm{argmin}_{i \le k \le j} E[k]$. Then, the basic operations can be expressed as $findclose(i) = fwd\_search(i,-1)$, $findopen(i) = bwd\_search(i,0) + 1$, and $enclose(i) = bwd\_search(i,-2) + 1$. In addition to those operations and *rank*/*select*, other operations can be computed, such as $LA(i,d) = bwd\_search(i,-d-1)+1$. Operation $child(i,q)$ and related ones are also done in constant time using small additional structures. The results are summarized as follows:

Theorem 1 ([19]) *For any ordinal tree with n nodes, all operations in Table 1 except insert and delete are carried out in constant time $\mathcal{O}(c)$ with a data structure using $2n + \mathcal{O}(n/\log^c n)$ bits of space on a $\Theta(\log n)$-bit word RAM, for any constant $c > 0$. The data structure can be constructed from the balanced parenthesis sequence of the tree, in $\mathcal{O}(n)$ time using $\mathcal{O}(n)$ bits of space.*

Another merit of using the range min-max tree is that it is easy to dynamize. By using a balanced tree, we obtain the following:

Theorem 2 ([19]) *On a $\Theta(\log n)$-bit word RAM, all operations on a dynamic ordinal tree with n nodes can be carried out within the worst-case complexities given in Table 1, using a data structure that requires $2n + \mathcal{O}(n \log\log n / \log n)$ bits. Alternatively, all the operations of the table can be carried out in $\mathcal{O}(\log n)$ time using $2n + \mathcal{O}(n/\log n)$ bits of space.*

The time complexity $\mathcal{O}(\log n / \log\log n)$ is optimal [3].

Other Representations

There are other representations of ordered trees and other types of trees [6, 7, 10]. The idea is that the entire tree is partitioned into mini-trees using the tree cover technique [10], and mini-trees are again partitioned into micro-trees.

Applications

There are many applications of succinct ordered trees because trees are fundamental data structures. A typical application is compressed suffix trees [20]. Balanced parentheses have also been used to represent planar and k-page graphs [16]. We also have other applications to encoding permutations and functions [17], grammar compression [15], compressing BDDs/ZDDs (binary decision diagrams/zero-suppressed BDDs) [5], etc.

Open Problems

An open problem is to give a dynamic data structure supporting all operations in the optimal $\mathcal{O}(\log n / \log\log n)$ time.

Experimental Results

Arroyuelo et al. [1] implemented LOUDS and the static version of the fully functional BP representation. LOUDS uses little space (as little as $2.1n$ bits) and solves its operations in half a microsecond or less, but its functionality is limited. The range min-max tree [19] requires about $2.4n$ bits and can be used to represent both BP and DFUDS. It solves all the operations within 1–2 microseconds. Previous implementations [8, 18] require more space and are generally slower.

Joannou and Raman [13] proposed an efficient implementation of the dynamic version using splay trees to represent the range min-max tree.

URLs to Code and Data Sets

An implementation of the BP representation using the range min-max tree by the original authors is available at https://github.com/fclaude/libcds. Another implementation by Simon Gog is at https://github.com/simongog/sdsl-lite.

Cross-References

- Compressed Range Minimum Queries
- Compressed Representations of Graphs
- Compressed Suffix Trees
- Grammar Compression
- Lowest Common Ancestors in Trees
- Rank and Select Operations on Bit Strings
- Succinct and Compressed Data Structures for Permutations and Integer Functions

Recommended Reading

1. Arroyuelo D, Cánovas R, Navarro G, Sadakane K (2010) Succinct trees in practice. In: Proceedings of 11th workshop on algorithm engineering and experiments (ALENEX), Austin. SIAM Press, pp 84–97
2. Benoit D, Demaine ED, Munro JI, Raman R, Raman V, Rao SS (2005) Representing trees of higher degree. Algorithmica 43(4):275–292
3. Chan HL, Hon WK, Lam TW, Sadakane K (2007) Compressed indexes for dynamic text collections. ACM Trans Algorithms 3(2):article 21
4. Clark DR (1996) Compact pat trees. PhD thesis, University of Waterloo
5. Denzumi S, Kawahara J, Tsuda K, Arimura H, Minato Si, Sadakane K (2014) DenseZDD: a compact and fast index for families of sets. In: Experimental algorithms (SEA), Copenhagen. LNCS 8503, pp 187–198
6. Farzan A, Munro JI (2014) A uniform paradigm to succinctly encode various families of trees. Algorithmica 68(1):16–40
7. Farzan A, Raman R, Rao SS (2009) Universal succinct representations of trees? In: Automata, languages and programming, 36th international colloquium (ICALP), Rhodes, pp 451–462
8. Geary RF, Rahman N, Raman R, Raman V (2006a) A simple optimal representation for balanced parentheses. Theor Comput Sci 368:231–246
9. Geary RF, Raman R, Raman V (2006b) Succinct ordinal trees with level-ancestor queries. ACM Trans Algorithms 2:510–534
10. He M, Munro JI, Satti SR (2012) Succinct ordinal trees based on tree covering. ACM Trans Algorithms 8(4):42
11. Jacobson G (1989) Space-efficient static trees and graphs. In: Proceedings of IEEE FOCS, Research Triangle Park, pp 549–554
12. Jansson J, Sadakane K, Sung WK (2012) Ultra-succinct representation of ordered trees with applications. J Comput Syst Sci 78(2):619–631
13. Joannou S, Raman R (2012) Dynamizing succinct tree representations. In: 11th international symposium experimental algorithms (SEA) 2012, Bordeaux, pp 224–235
14. Lu HI, Yeh CC (2008) Balanced parentheses strike back. ACM Trans Algorithms (TALG) 4(3):article 28
15. Maruyama S, Tabei Y, Sakamoto H, Sadakane K (2013) Fully-online grammar compression. In: Proceedings of string processing and information retrieval (SPIRE), Jerusalem. LNCS 8214, pp 218–229
16. Munro JI, Raman V (2001) Succinct representation of balanced parentheses and static trees. SIAM J Comput 31(3):762–776
17. Munro JI, Raman R, Raman V, Rao SS (2012) Succinct representations of permutations and functions. Theor Comput Sci 438:74–88
18. Navarro G (2009) Implementing the LZ-index: theory versus practice. J Exp Algorithmics 13:article 2
19. Navarro G, Sadakane K (2014) Fully-functional static and dynamic succinct trees. ACM Trans Algorithms 10(3):article 16
20. Sadakane K (2007) Compressed suffix trees with full functionality. Theory Comput Syst 41(4):589–607

Compressing and Indexing Structured Text

Paolo Ferragina[1] and Srinivasa Rao Satti[2]
[1]Department of Computer Science, University of Pisa, Pisa, Italy
[2]Department of Computer Science and Engineering, Seoul National University, Seoul, South Korea

Keywords

Tree compression; Tree indexing; (labeled) Tree succinct representations; XML compression and indexing

Years and Authors of Summarized Original Work

2005; Ferragina, Luccio, Manzini, Muthukrishnan

Problem Definition

Trees are a fundamental structure in computing. They are used in almost every aspect of modeling and representation for computations like searching for keys, maintaining directories, and representations of parsing or execution traces, to name just a few. One of the latest uses of trees is *XML*, the de facto format for data storage, integration, and exchange over the Internet (see http://www.w3.org/XML/). Explicit storage of trees, with one pointer per child as well as other auxiliary information (e.g., label), is often taken as given but can account for the dominant storage cost. Just to have an idea, a simple tree encoding needs at least 16 bytes per tree node: one pointer to the auxiliary information (e.g., node label) plus three node pointers to the parent, the first child, and the next sibling. This large space occupancy may even prevent the processing of medium-sized trees, e.g., *XML* documents. This entry surveys the best-known storage solutions for unlabeled and labeled trees that are space efficient and support fast navigational and search operations over the tree structure. In the literature, they are referred to as *succinct/compressed tree-indexing* solutions.

Notation and Basic Facts

The information-theoretic storage cost for any item of a universe U can be derived via a simple *counting argument*: at least $\log |U|$ bits are needed to distinguish any two items of U. (Throughout the entry, all logarithms are taken to the base 2, and it is assumed $0 \log 0 = 0$.) Now, let $\mathcal{T}$ be a rooted tree of arbitrary degree and shape, and consider the following three main classes of trees:

Ordinal trees $\mathcal{T}$ is unlabeled and its children are left-to-right *ordered*. The number of ordinal trees on t nodes is $C_t = \binom{2t}{t} / (t+1)$ which induces a lower bound of $2t - \Theta(\log t)$ bits

Cardinal k-ary Trees $\mathcal{T}$ is labeled on its *edges* with symbols drawn from the alphabet $\Sigma = \{1, \ldots, k\}$. Any node has degree at most k because the edges outgoing from each node have *distinct* labels. Typical examples of cardinal trees are the binary tree ($k = 2$), the (uncompacted) trie, and the *Patricia tree*. The number of k-ary cardinal trees on t nodes is $C_t^k = \binom{kt+1}{t} / (kt+1)$ which induces a lower bound of $t(\log k + \log e) - \Theta(\log kt)$ bits, when k is a slowly growing function of t

(Multi-)labeled trees $\mathcal{T}$ is an ordinal tree, labeled on its *nodes* with symbols drawn from the alphabet Σ. In the case of multi-labeled trees, every node has *at least* one symbol as its label. The *same* symbols may repeat among sibling nodes, so that the degree of each node is *unbounded*, and the same labeled subpath may occur many times in $\mathcal{T}$, anchored anywhere. The information-theoretic lower bound on the storage complexity of this class of trees on t nodes comes easily from the decoupling of the tree structure and the storage of tree labels. For labeled trees, it is $\log C_t + t \log |\Sigma| = t(\log |\Sigma| + 2) - \Theta(\log t)$ bits

The following query operations should be supported over $\mathcal{T}$:

Basic navigational queries They ask for the parent of a given node u, the ith child of u, and the degree of u. These operations may be restricted to some label $c \in \Sigma$, if $\mathcal{T}$ is labeled

Sophisticated navigational queries They ask for the jth level-ancestor of u, the depth of u, the subtree size of u, the lowest common ancestor of a pair of nodes, and the ith node according to some node ordering over $\mathcal{T}$, possibly restricted to some label $c \in \Sigma$ (if $\mathcal{T}$ is labeled). For even more operations, see [2, 14]

Subpath query Given a labeled subpath Π, it asks for the (number *occ* of) nodes of $\mathcal{T}$ that immediately descend from every occurrence of

Π in $\mathcal{T}$. Each subpath occurrence may be anchored anywhere in the tree (i.e., not necessarily in its root)

The elementary solution to the tree-indexing problem consists of encoding the tree $\mathcal{T}$ via a mixture of pointers and arrays, thus taking a total of $\Theta(t \log t)$ bits. This supports basic navigational queries in constant time, but it is not space efficient and requires visiting the whole tree to implement the subpath query or the more sophisticated navigational queries. Here, the goal is to design tree storage schemes that are either *succinct*, namely, "close to the information-theoretic lower bound" mentioned before or *compressed* in that they achieve "entropy-bounded storage." Furthermore, these storage schemes do *not* require the whole visit of the tree for most navigational operations. Thus, succinct/compressed tree-indexing solutions are distinct from simply compressing the input and then uncompressing it at query time.

In this entry, it is assumed that $t \geq |\Sigma|$. The model of computation used is the random access machine (RAM) with word size $\Theta(\log t)$, where one can perform various arithmetic and bit-wise Boolean operations on single words in constant time.

Key Results

The notion of *succinct* data structures was introduced by Jacobson [13] in a seminal work over 25 years ago. He presented a storage scheme for ordinal trees using $2t + o(t)$ bits that supports basic navigational queries in $O(\log \log t)$ time (i.e., parent, first child, and next sibling of a node). Later, Munro and Raman [16] closed the issue for ordinal trees on basic navigational queries and the subtree-size query by achieving constant query time and $2t + o(t)$ bits of storage. Their storage scheme is called *balanced parenthesis* (*BP*) representation (some papers [Chiang et al., ACM-SIAM SODA '01; Sadakane, ISAAC '01; Munro et al., J.ALG '01; Munro and Rao, ICALP '04] have extended *BP* to support in constant time other sophisticated navigational queries like *LCA*, node degree, rank/select on leaves and number of leaves in a subtree, level-ancestor and level-successor) representation. Subsequently, Benoit et al. [3] proposed a storage scheme called *depth-first unary degree sequence* (shortly, *DFUDS*) that still uses $2t + o(t)$ bits but performs more navigational queries like ith child, child rank, and node degree in constant time. Geary et al. [10] gave another representation still taking optimal space that extends *DFUDS*'s operations with the level-ancestor query.

Although these three representations achieve the optimal space occupancy, none of them supports every existing operation in constant time: e.g., *BP* does not support ith child and child rank and *DFUDS* and Geary et al.'s representation do not support *LCA*. Later, Jansson et al. [14] extended the *DFUDS* storage scheme in two directions: (1) they showed how to implement in constant time all navigational queries above and (the *BP* representation and the one of Geary et al. [10] have been recently extended to support further operations-like depth/height of a node, next node in the same level, rank/select over various node orders-still in constant time and $2t + o(t)$ bits see [11] and references therein) (2) they showed how to compress the new tree storage scheme up to $H^*(\mathcal{T})$, which denotes the entropy of the distribution of node degrees in $\mathcal{T}$.

Theorem 1 (Jansson et al. [14]) *For any rooted (unlabeled) tree $\mathcal{T}$ with t nodes, there exists a tree-indexing scheme that uses $tH^*(\mathcal{T}) + O(t(\log \log t)^2/\log t)$ bits and supports all navigational queries in constant time.*

This improves the standard tree pointer-based representation, since it needs no more than $H^*(\mathcal{T})$ bits per node and does not compromise the performance of sophisticated navigational queries. Since it is $H^*(\mathcal{T}) \leq 2$, this solution is also never worse than BP or DFUDS, but its improvement may be significant! This result can be extended to achieve the kth-order entropy of the *DFUDS* sequence, by adopting any compressed-storage scheme for strings (see, e.g., [7] and references therein).

Further work in the area of succinct ordinal tree representations came in the form of (i) a

uniform approach to succinct tree representations [5] that simplified and extended the representation of Geary et al., (ii) a *universal representation* [6] that emulates all three representations mentioned above, and (iii) a *fully functional representation* [18] that obtains a simplified ordinal tree encoding with reduced space occupancy.

Benoit et al. [3] extended the use of *DFUDS* to cardinal trees and proposed a tree-indexing scheme whose space occupancy is close to the information-theoretic lower bound and supports various navigational queries in constant time. Raman et al. [19] improved the space by using a different approach (based on storing the tree as a set of edges), thus proving the following:

Theorem 2 (Raman et al. [19]) *For any k-ary cardinal tree $\mathcal{T}$ with t nodes, there exists a tree-indexing scheme that uses $\log C_t^k + o(t) + O(\log\log k)$ bits and supports in constant time the following operations: finding the parent, the degree, the ordinal position among its siblings, the child with label c, and the ith child of a node.*

The subtree-size operation cannot be supported efficiently using this representation, so [3] should be resorted to in case this operation is a primary concern.

Despite this flurry of activity, the fundamental problem of indexing *labeled* trees succinctly has remained mostly unsolved. In fact, the succinct encoding for ordered trees mentioned above might be replicated $|\Sigma|$ times (once for each symbol of Σ), and then the divide-and-conquer approach of [10] might be applied to reduce the final space occupancy. However, the final space bound would be $2t + t\log|\Sigma| + O(t|\Sigma|(\log\log\log t)/(\log\log t))$ bits, which is nonetheless far from the information-theoretic storage bound even for moderately large Σ. On the other hand, if subpath queries are of primary concern (e.g., *XML*), one can use the approach of [15] which consists of a variant of the suffix-tree data structure properly designed to index all $\mathcal{T}$'s labeled paths. Subpath queries can be supported in $O(|\Pi|\log|\Sigma| + occ)$ time, but the required space would still be $\Theta(t\log t)$ bits (with large hidden constants, due to the use of suffix trees). Subsequently, some papers [1, 2, 8, 12] addressed this problem in its whole generality by either dealing simultaneously with subpath and basic navigational queries [8] or by considering *multi*-labeled trees and a larger set of navigational operations [1, 2, 12].

In particular, [8] introduced a *transform* of the labeled tree $\mathcal{T}$, denoted $xbw[\mathcal{T}]$, which linearizes it into *two* coordinated arrays $\langle \mathcal{S}_{\text{last}}, \mathcal{S}_\alpha \rangle$: the former capturing the tree structure and the latter keeping a permutation of the labels of $\mathcal{T}$. $xbw[\mathcal{T}]$ has the optimal (up to lower-order terms) size of $2t + t\log|\Sigma|$ bits and can be built and inverted in optimal linear time. In designing the *XBW*-transform, the authors were inspired by the elegant Burrows-Wheeler transform for strings [4]. The power of $xbw[\mathcal{T}]$ relies on the fact that it allows one to transform compression and indexing problems on labeled trees into easier problems over strings. Namely, the following two string-search primitives are key tools for indexing $xbw[\mathcal{T}]$:$\text{rank}_c(S, i)$ returns the number of occurrences of the symbol c in the string prefix $S[1,i]$, and $\text{select}_c(S, j)$ returns the position of the jth occurrence of the symbol c in string S. The literature offers many time-/space-efficient solutions for these primitives that could be used as a *black box* for the compressed indexing of $xbw[\mathcal{T}]$ (see, e.g., [2, 17] and references therein).

Theorem 3 (Ferragina et al. [8]) *Consider a tree $\mathcal{T}$ consisting of t nodes labeled with symbols drawn from alphabet Σ. There exists a compressed tree-indexing scheme that achieves the following performance:*

- *If $|\Sigma| = O(\text{polylog}(t))$, the index takes at most $tH_0(\mathcal{S}_\alpha) + 2t + o(t)$ bits and supports basic navigational queries in constant time and (counting) subpath queries in $O(|\Pi|)$ time.*
- *For any alphabet Σ, the index takes less than $tH_k(\mathcal{S}_\alpha) + 2t + o(t\log|\sum|))$ bits, but label-based navigational queries and (counting) subpath queries are slowed down by a factor $o((\log\log|\Sigma|)3)$.*

Here, $H_k(s)$ is the kth-order empirical entropy of string s, with $H_k(s) \le H_{k-1}(s)$ for any $k > 0$.

Since $H_k(\mathcal{S}_\alpha) \leq H_0(\mathcal{S}_\alpha) \log |\sum|$, the indexing of $xbw[\mathcal{T}]$ takes at most as much space as its plain representation, up to lower-order terms, but with the additional feature of being able to navigate and search $\mathcal{T}$ efficiently. This is indeed a sort of *pointerless representation* of the labeled tree $\mathcal{T}$ with additional search functionalities (see [8] for details).

If sophisticated navigational queries over labeled trees are a primary concern, and subpath queries are not necessary, then the approach of Barbay et al. [1, 2] should be followed. They proposed the novel concept of *succinct index*, which is different from the concept of *succinct/compressed encoding* implemented by all the above solutions. A succinct index does not *touch* the data to be indexed; it just accesses the data via basic operations offered by the underlying abstract data type (ADT) and requires asymptotically less space than the information-theoretic lower bound on the storage of the data itself. The authors reduce the problem of indexing labeled trees to the one of indexing ordinal trees and strings and the problem of indexing multi-labeled trees to the one of indexing ordinal trees and binary relations. Then, they provide succinct indexes for strings and binary relations. In order to present their result, the following definitions are needed. Let m be the total number of symbols in $\mathcal{T}$, let t_c be the number of nodes labeled c in $\mathcal{T}$, and let ρ_c be the maximum number of labels c in any rooted path of $\mathcal{T}$ (called the *recursivity* of c). Define ρ as the average recursivity of $\mathcal{T}$, namely, $\rho = (1/m) \sum_{c \in \sum} (t_c \rho_c)$.

Theorem 4 (Barbay et al. [1]) *Consider a tree $\mathcal{T}$ consisting of t nodes (multi-)labeled with possibly many symbols drawn from alphabet Σ. Let m be the total number of symbols in $\mathcal{T}$, and assume that the underlying ADT for $\mathcal{T}$ offers basic navigational queries in constant time and retrieves the ith label of a node in time f. There is a succinct index for $\mathcal{T}$ using $m(\log \rho + o(\log(|\Sigma|\rho)))$ bits that supports for a given node u the following operations (where $L = \log\log|\sum| \log\log\log|\sum|$):*

- *Every c-descendant or c-child of u can be retrieved in $O(L(f + \log\log|\sum|))$ time.*
- *The set A of c-ancestors of u can be retrieved in $O(L(f + \log\log|\sum|)) + |A|(\log\log\rho_c + \log\log\log|\sum|(f + \log\log|\sum|)))$ time.*

More recently, He et al. [12] obtained new representations that support a much broader collection of operations than the ones mentioned above.

Applications

As trees are ubiquitous in many applications, this section concentrates just on two examples that, in their simplicity, highlight the flexibility and power of succinct/compressed tree indexes.

The first example regards suffix trees, which are a crucial algorithmic block of many string processing applications – ranging from bioinformatics to data mining, from data compression to search engines. Standard implementations of suffix trees take at least 80 bits per node. The compressed suffix tree of a string $S[1,s]$ consists of three components: the tree topology, the string depths stored into the internal suffix-tree nodes, and the suffix pointers stored in the suffix-tree leaves (also called *suffix array* of S). The succinct tree representation of [14] can be used to encode the suffix-tree topology and the string depths taking $4s + o(s)$ bits (assuming w.l.o.g. that $|\Sigma| = 2$). The suffix array can be compressed up to the kth-order entropy of S via any solution surveyed in [17]. The overall result is never worse than 80 bits per node, but can be significantly better for highly compressible strings.

The second example refers to the *XML* format which is often modeled as a labeled tree. The succinct/compressed indexes in [1, 2, 8] are theoretical in flavor but turn out to be relevant for practical *XML* processing systems. As an example, [9] has published some encouraging experimental results that highlight the impact of the *XBW*-transform on real *XML* datasets. The authors show that a proper adaptation of the *XBW*-transform allows one to compress *XML* data up to state-of-the-art XML-conscious compressors

and to provide access to its content, navigate up and down the *XML* tree structure, and search for simple path expressions and substrings in a few milliseconds over MBs of *XML* data, by uncompressing only a tiny fraction of them at each operation. Previous solutions took several seconds per operation!

Open Problems

For recent results, open problems, and further directions of research in the general area of succinct tree representation, the interested reader is referred to [2, 11, 14, 18] and references therein. Here, we describe two main problems, which naturally derive from the discussion above.

Motivated by *XML* applications, one may like to extend the subpath search operation to the *efficient* search for all leaves of $\mathcal{T}$ whose labels *contain* a substring β and that *descend from* a given subpath Π. The term "efficient" here means in time proportional to $|\Pi|$ and to the number of retrieved occurrences, but independent as much as possible of $\mathcal{T}$'s size in the worst case. Currently, this search operation is possible only for the leaves which are immediate descendants of Π, and even for this setting, the solution proposed in [9] is not optimal.

There are two main encodings for trees which lead to the results above: ordinal tree representation (*BP*, *DFUDS* or the representation of Geary et al. [10]) and *XBW*. The former is at the base of solutions for sophisticated navigational operations, and the latter is at the base of solutions for sophisticated subpath searches. Is it possible to devise *one unique* transform for the labeled tree $\mathcal{T}$ which combines the best of the two worlds and is still compressible?

Experimental Results

See http://mattmahoney.net/dc/text.html and at the paper [9] for numerous experiments on *XML* datasets.

Data Sets

See http://mattmahoney.net/dc/text.html and the references in [9].

URL to Code

Paper [9] contains a list of software tools for compression and indexing of *XML* data.

Cross-References

- Burrows-Wheeler Transform
- Compressed Suffix Array
- Compressed Tree Representations
- Rank and Select Operations on Bit Strings
- Rank and Select Operations on Sequences
- Suffix Trees and Arrays
- Table Compression

Recommended Reading

1. Barbay J, Golynski A, Munro JI, Rao SS (2007) Adaptive searching in succinctly encoded binary relations and tree-structured documents. Theor Comput Sci 387:284–297
2. Barbay J, He M, Munro JI, Rao SS (2011) Succinct indexes for strings, binary relations and multi-labeled trees. ACM Trans Algorithms 7(4):article 52
3. Benoit D, Demaine E, Munro JI, Raman R, Raman V, Rao SS (2005) Representing trees of higher degree. Algorithmica 43:275–292
4. Burrows M, Wheeler D (1994) A block sorting lossless data compression algorithm. Technical report 124, Digital Equipment Corporation
5. Farzan A, Munro JI (2014) A uniform paradigm to succinctly encode various families of trees. Algorithmica 68(1):16–40
6. Farzan A, Raman R, Rao SS (2009) Universal succinct representations of trees? In: Proceedings of the 36th international colloquium on automata, languages and programming (ICALP, Part I), Rhodes. Lecture notes in computer science, vol 5555. Springer, pp 451–462
7. Ferragina P, Venturini R (2007) A simple storage scheme for strings achieving entropy bounds. Theor Comput Sci 372(1):115–121
8. Ferragina P, Luccio F, Manzini G, Muthukrishnan S (2005) Structuring labeled trees for optimal succinctness, and beyond. In: Proceedings of the 46th

IEEE symposium on foundations of computer science (FOCS), Cambridge, pp 184–193. The journal version of this paper appear in J ACM 57(1) (2009)
9. Ferragina P, Luccio F, Manzini G, Muthukrishnan S (2006) Compressing and searching XML data via two zips. In: Proceedings of the 15th World Wide Web conference (WWW), Edingburg, pp 751–760
10. Geary R, Raman R, Raman V (2006) Succinct ordinal trees with level-ancestor queries. ACM Trans Algorithms 2:510–534
11. He M, Munro JI, Rao SS (2012) Succinct ordinal trees based on tree covering. ACM Trans Algorithms 8(4):article 42
12. He M, Munro JI, Zhou G (2012) A framework for succinct labeled ordinal trees over large alphabets. In: Proceedings of the 23rd international symposium on algorithms and computation (ISAAC), Taipei. Lecture notes in computer science, vol 7676. Springer, pp 537–547
13. Jacobson G (1989) Space-efficient static trees and graphs. In: Proceedings of the 30th IEEE symposium on foundations of computer science (FOCS), Triangle Park, pp 549–554
14. Jansson J, Sadakane K, Sung W Ultra-succinct representation of ordered trees. J Comput Syst Sci 78: 619–631 (2012)
15. Kosaraju SR (1989) Efficient tree pattern matching. In: Proceedings of the 20th IEEE foundations of computer science (FOCS), Triangle Park, pp 178–183
16. Munro JI, Raman V (2001) Succinct representation of balanced parentheses and static trees. SIAM J Comput 31(3):762–776
17. Navarro G, Mäkinen V (2007) Compressed full text indexes. ACM Comput Surv 39(1):article 2
18. Navarro G, Sadakane K (2014) Fully functional static and dynamic succinct trees. ACM Trans Algorithms 10(3):article 16
19. Raman R, Raman V, Rao SS (2007) Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. ACM Trans Algorithms 3(4):article 43

Compressing Integer Sequences

Alistair Moffat
Department of Computing and Information Systems, The University of Melbourne, Melbourne, VIC, Australia

Keywords

Binary code; Byte-based code; Compressed set representations; Elias code; Elias-Fano code; Golomb code; Integer code; Interpolative code

Years and Authors of Summarized Original Work

1966; Golomb
1974; Elias
2000; Moffat, Stuiver

Problem Definition

An n-symbol message $M = \langle s_0, s_1, \ldots, s_{n-1} \rangle$ is given, where each symbol s_i is an integer in the range $0 \le s_i < U$. If the s_is are strictly increasing, then M identifies an n-subset of $\{0, 1, \ldots, U-1\}$.

Objective To economically encode M as a binary string over $\{0, 1\}$.

Constraints

1. **Short messages.** The message length n may be small relative to U.
2. **Monotonic equivalence.** Message M is converted to a strictly increasing message M' over the alphabet $U' \le Un$ by taking prefix sums, $s'_i = i + \sum_{j=0}^{i} s_j$ and $U' = s'_{n-1} + 1$. The inverse is to "take gaps," $g_i = s_i - s_{i-1} - 1$, with $g_0 = s_0$.
3. **Combinatorial Limit.** If M is monotonic then $\left\lceil \log_2 \binom{U}{n} \right\rceil \le U$ bits are required in the worst case. When $n \ll U$, $\log_2 \binom{U}{n} \approx n(\log_2(U/n) + \log_2 e)$.

Key Results

Monotonic sequences can be coded in $\min\{U, n(\log_2(U/n) + 2)\}$ bits. Non-monotonic sequences can be coded in $\min\{\sum_{i=0}^{n-1}(\log_2(1 + s_i) + o(\log s_i)), n(\log_2(U'/n) + 2)\}$ bits.

Unary and Binary Codes

The *unary code* represents symbol x as x `1`-bits followed by a single `0`-bit. The unary code for x is $1 + x$ bits long; hence, the corresponding ideal symbol probability distribution (for which this

pattern of codeword lengths yields the minimal message length) is given by $p_x = 2^{-(1+x)}$. Unary is an *infinite code*, for which knowledge of U is not required. But unless M is dominated by small integers, unary is expensive – the representation of a message $M = \langle s_0 \dots s_{n-1} \rangle$ requires $n + \sum_i s_i = U' + 1$ bits.

If $U \leq 2^k$ for integer k, then s_i can be represented in k bits using the *binary code*. Binary is *finite*, with an ideal probability distribution given by $p_x = 2^{-k}$. When $U = 2^k$, the ideal probability $p_x = 2^{-\log_2 U} = 1/U$. When $2^{k-1} < U < 2^k$, then $2^k - U$ of the codewords can be shortened to $k-1$ bits, in a *minimal binary code*. It is usual (but not necessary) to assign the short codewords to $0 \dots 2^k - U - 1$, leaving the codewords for $2^k - U \dots U - 1$ as k bits.

Elias Codes

Peter Elias described a suite of hybrids between unary and binary in work published in 1975 [7]. This family of codes are defined recursively, with unary being the base method. To code a value x, the "predecessor" Elias code is used to specify $x' = \lfloor \log_2(1 + x) \rfloor$, followed by an x'-bit binary code for $x - (2^{x'} - 1)$. The second member of the Elias family is C_γ and is a unary-binary code: unary for the prefix part and then binary to indicate the value of x within the range it specifies. The first few C_γ codewords are `0`, `10 0`, `10 1`, `110 00`, and so on, where spaces are used to illustrate the split between the components. The C_γ codeword for a value x requires $1 + \lfloor \log_2(1 + x) \rfloor$ bits for the unary part and a further $\lfloor \log_2(1+x) \rfloor$ bits for the binary part. The ideal probability distribution is thus given by $p_x \approx 1/(2(1 + x)^2)$. After C_γ, the next member of the Elias family is C_δ, in which C_γ is used to store x'. The first few codewords are `0`, `100 0`, `100 1`, and `101 00`; like unary and C_γ, C_δ is infinite, but now the codeword for x requires $1+2\lfloor \log_2(1+x') \rfloor + x'$ bits, where $x' = \lfloor \log_2(1 + x) \rfloor$. Further members of the family can be generated, but for most practical purposes, C_δ is the last useful one. To see why, note that $|C_\gamma(x')| \leq |C_\delta(x')|$ whenever $x' \leq 30$, meaning that the next Elias code is shorter than C_δ only for values $x' \geq 31$, that is, for $x \geq 2^{31} - 1$.

Fibonacci-Based Codes

Another infinite code arises from the Fibonacci sequence described (for this purpose) as $F_0 = 1$, $F_1 = 2$, $F_2 = 3$, $F_3 = 5$, $F_4 = 8$, and in general $F_i = F_{i-1} + F_{i-2}$. The *Zeckendorf* representation of a natural number is a list of Fibonacci values that add up to it, such that no two adjacent Fibonacci numbers are used. For example, 9 is the sum of $1 + 8 = F_0 + F_4$. The *Fibonacci code* for $x \geq 0$ is derived from the Zeckendorf representation of $x + 1$ and consists of a `1` bit in the ith position (counting from the left) if F_i appears in the sum and a `0` bit if not. Because it is not possible for both F_i and F_{i+1} to be part of the sum, the last 2 bits must be `01`; hence, appending a further `1` bit provides a unique sentinel for the codeword. The code for $x = 0$ is `11`, and the next few codewords are `011`, `0011`, `1011`, `00011`, and `10011`. Because (for large i) $F_i \approx \phi^{i+2}/\sqrt{5}$, where $\phi = (1 + \sqrt{5})/2 \approx 1.62$, the codeword for x requires approximately $\lfloor \log_\phi \sqrt{5} + \log_\phi(1 + x)\sqrt{5} \rfloor \approx \lfloor 1.67 + 1.44 \log_2(1 + x) \rfloor$ bits and is longer than C_γ only for $x = 0$ and $x = 2$. The Fibonacci code is also as good as, or better than, C_δ between $x = 1$ and $F_{18} - 2 = 6{,}763$. Higher-order variants are also possible, with increased minimum codeword lengths and decreased coefficients on the logarithmic term. Fenwick [8] provides coverage of Fibonacci codes.

Byte-Aligned Codes

Extracting bits from bitstrings can slow down decoding rates, especially if each bit is then tested in a loop guard. Operations on larger units tend to be faster. The simplest *byte-aligned code* is an interleaved 8-bit analog of the Elias C_γ mechanism. The top bit in each byte is reserved for a flag that indicates (when `0`) that this is the last byte of the codeword and (when `1`) that the codeword continues; the other 7 bits in each byte are for data. A total of $8\lceil (\log_2 x)/7 \rceil$ bits are used, which makes it more effective asymptotically than the Elias C_γ code or the Fibonacci code. However, the minimum 8 bits means that byte-aligned codes are expensive on messages dominated by small values. A further advantage of byte codes is that

compressed sequences can be searched, using a search pattern converted using the same code [5]. The zero top bit in all final bytes means that false matches can be easily eliminated.

An improvement to the simple byte-aligned coding mechanism arises from the observation that different values for the separating value between the *stopper* and *continuer* bytes lead to different trade-offs in overall codeword lengths [3]. In the (S, C)-*byte-aligned code*, values for $S + C = 256$ are chosen, and each codeword consists of a sequence of zero or more continuer bytes with values greater than or equal to S, followed by a stopper byte with a value less than S. Other variants include methods that use bytes as the coding units to form Huffman codes, either using 8-bit coding symbols or tagged 7-bit units [5] and methods that partially permute the alphabet, but avoid the need for a complete mapping [4]. Culpepper and Moffat [4] also describe a byte-aligned coding method that creates a set of byte-based codewords with the property that the first byte uniquely identifies the length of the codeword. Similarly, *nibble* codes can be designed as a 4-bit analog of the byte-aligned approach, with 1 bit reserved for a stopper-continuer flag, and 3 bits used for data.

Golomb Codes

In 1966 Solomon Golomb observed that the intervals between consecutive members of a random n-subset of the items $0 \ldots U-1$ could be modeled by a geometric probability distribution $p_x = p(1-p)^{x-1}$, where $p = n/U$ [10]. This probability distribution implies that in a *Golomb code*, the representation for $x+b$ should be 1 bit longer than the representation for x when $(1-p)^b \approx 0.5$, that is, when $b = \log 0.5 / \log(1-p) \approx (\ln 2)/p \approx 0.69U/n$. Assuming a monotonic message, each s_i is converted to a gap; then g_i div b is coded in unary; and finally g_i mod b is coded in minimal binary with respect to b. Like unary, the Golomb code is infinite. If the sequence is non-monotonic, then the values s_i are coded directly using parameter $b = 0.69U'/n$. Each `1`-bit that is part of a unary part spans b elements of U, meaning that there are at most $\lfloor U/b \rfloor$ of them in total; and there are exactly n `0`-bits in the unary parts. The minimal binary parts, one per symbol, take fewer than $n\lceil \log_2 b \rceil$ bits in total. Summing these components, and maximizing the cost over different values of n and U by assuming an adversary that forces the use of the first long minimal binary codeword whenever possible, yields a total Golomb code length for a monotonic sequence of at most $n(\log_2(U/n) + 2)$ bits. The variant in which $b = 2^k$ is used, $k = \lfloor \log_2(U/n) \rfloor$, is called a *Rice code*. Note that Golomb and Rice codes are infinite, but require that a parameter be set and that one way of estimating the parameter is based on knowing a value for U.

Other Static Codes

Elias codes and Golomb codes are examples of methods specified by a set of *buckets*, with symbol x coded in two parts: a bucket identifier, followed by an offset within the bucket, the latter usually using minimal binary. For example, the Elias C_γ code employs a vector of bucket sizes $\langle 2^0, 2^1, 2^2, 2^3, 2^4, \ldots \rangle$. Teuhola (see Moffat and Turpin [14]) proposed a hybrid in which a parameter k is chosen, and the vector of bucket sizes is given by $\langle 2^k, 2^{k+1}, 2^{k+2}, 2^{k+3}, \ldots \rangle$. One way of setting the parameter k is the length in bits of the median sequence value, so that the first bit of each codeword approximately halves the range of observed symbol values. Another variant is described by Boldi and Vigna [2], using vector $\langle 2^k-1, (2^k-1)2^k, (2^k-1)2^{2k}, (2^k-1)2^{3k}, \ldots \rangle$ to obtain a family of codes that are analytically and empirically well suited to power-law probability distributions, especially (taking k in the range 2–4) those associated with web-graph compression. Fraenkel and Klein [9] observed that the sequence of symbol *magnitudes* (i.e., the sequence of values $\lfloor \log_2(1 + s_i) \rfloor$) provides a denser range than the message itself and that it can be effective to use a principled code for them. For example, rather than using unary for the prefix part, a Huffman code can be used. Moffat and Anh [12] consider other ways in which the prefix part of each codeword can be reduced; and Fenwick [8] provides general coverage of other static coding methods.

Elias-Fano Codes

In 1974 Elias [6] presented another coding method, noting that it was described independently in 1971 by Robert Fano. The approach is now known as *Elias-Fano coding*. For monotonic sequence M, parameter $k = \lfloor \log_2(U/n) \rfloor$ is used to again break each codeword into quotient and remainder, *without* first taking gaps, with codewords formed relative to a sequence of buckets each of width $b = 2^k$. The number of symbols in the buckets is stored in a bitstring of $\lceil U/b \rceil < 2n$ unary codes. The n remainder parts $r_i = s_i \bmod b$ are stored as a sequence of k-bit binary codes. Each symbol contributes k bits as a binary part and adds 1 bit to one of the unary parts; plus there are at most $\lceil U/b \rceil$ 0-bits terminating the unary parts. Based on these relationships, the total length of an Elias-Fano code can be shown to be at most $n(\log_2(U/n) + 2)$ bits. Vigna [15] has deployed Elias-Fano codes to good effect.

Packed Codes

If each n-subset of $0 \ldots U-1$ is equally likely, the Golomb code is effective in the average case; and the Elias-Fano code is effective in the worst case. But if the elements in the subset are *clustered*, then it is possible to obtain smaller representations, provided that groups of elements themselves can be employed as part of the process of determining the code. The *word-aligned* codes of Anh and Moffat [1] fit as many binary values into each output word as possible. For example, in their Simple-9 method, 32-bit output words are used, and the first 4 bits of each word contain a *selector* which specifies how to decode the next 28 bits: one 28-bit binary number, or two 14-bit binary numbers, or three 9-bit numbers, and so on. Other variants use 64-bit words [1]. In these codes, clusters of low s_i (or g_i) values can be represented more compactly than would occur with the Golomb code and an all-of-message b parameter; and decoding is fast because whole words are expanded without any need for conditionals or branching.

Zukowski et al. [16] describe a different approach, in which blocks of z values from M are coded in binary using k bits each, where k is chosen such that 2^k is larger than most, but not necessarily all, of the z elements in the block. Any s_i's in the block that are larger than $2^k - 1$ are noted as *exceptions* and handled separately; a variety of methods for coding the exceptions have been used in different forms of the *pfordelta code*. This mechanism is again fast when decoding, due to the absence of conditional bit evaluations, and, for typical values such as $z = 128$, also yields effective compression. Lemire and Boytsov have carried out detailed experimentation with packed codes [11].

Context-Sensitive Codes

If the objective is to create the smallest representation, rather than provide a balance between compression effectiveness and decoding speed, the nonsequential *binary interpolative code* of Moffat and Stuiver [13] can be used. As an example, consider message M, shown in Table 1, and suppose that the decoder is aware that $U' = 29$, that is, that $s'_i < 29$. Every item in M is greater than or equal to $lo = 0$ and less than $hi = 29$, and the mid-value in M, in this example $s_4 = 6$ (it doesn't matter which mid-value is chosen), can be transmitted to the decoder using $\lceil \log_2 29 \rceil = 5$ bits. Once that middle number is pinned, the remaining values can be coded recursively within more precise ranges and might require fewer than 5 bits each.

In fact, there are four distinct values in M that precede s'_4 and another five that follow it, so a more restricted range for s'_4 can be inferred: it must be greater than or equal to $lo' = lo + 4 = 4$ and less than $hi' = hi - 5 = 24$. That is, $s'_4 = 6$ can be minimal binary coded as the value $6 - lo' = 2$ within the range $[0, 20)$ using just 4 bits.

It remains to transmit the left part, $\langle 0, 3, 4, 5 \rangle$, against the knowledge that every value is greater than or equal to $lo = 0$ and less than $hi = 6$, and the right part, $\langle 16, 24, 26, 27, 28 \rangle$, against the knowledge that every value is greater than or equal to $lo = 7$ and less than $hi = 29$. These two sublists are processed recursively in the order shown in the remainder of Table 1, with the tighter ranges $[lo', hi')$ also shown at

Compressing Integer Sequences, Table 1 Example encodings of message $M = \langle 0, 3, 4, 5, 6, 16, 24, 26, 27, 28 \rangle$ using the interpolative code. When a minimal binary code is applied to each value in its corresponding range, a total of **20** bits are required. No bits are output if $lo' = hi' - \mathbf{1}$

i	s_i	s'_i	lo	hi	lo'	hi'	$s_i - lo'$	$hi' - lo'$	Binary	MinBin
4	0	6	0	29	4	24	2	20	`00010`	`0010`
1	2	3	0	6	1	4	2	3	`10`	`11`
0	0	0	0	3	0	3	0	3	`00`	`0`
2	0	4	4	6	4	5	0	1	`--`	`--`
3	0	5	5	6	5	6	0	1	`--`	`--`
7	1	26	7	29	9	27	17	18	`10001`	`11111`
5	9	16	7	26	7	25	9	18	`01001`	`1001`
6	7	24	17	26	17	26	7	9	`0111`	`1110`
8	1	27	27	29	27	28	0	1	`--`	`--`
9	0	28	28	29	28	29	0	1	`--`	`--`

each step. One key feature of the interpolative code is that the when the range is just one, codewords of length zero are used. Four of the symbols in M benefit in this way. The interpolative code is particularly effective when the subset contains runs of consecutive items, or localized regions where there is a high density.

The final column of Table 1 uses minimal binary for each value within its bounded range. A refinement is to use a *centered minimal binary code*, so that the short codewords are assigned in the middle of the range rather than at the beginning, recognizing that the midpoint of a set is more likely to be near the middle of the range spanned by the set than it is to be near the ends of the range. Adding this enhancement tends to be beneficial. But improvement is not guaranteed, and on M it adds 1 bit to the compressed representation compared to what is shown in Table 1.

Applications

Messages in this form are often the output of a modeling step in a data compression system. Other examples include recording the hyperlinks in the graph representation of the World Wide Web and storing inverted indexes for large document collections.

Cross-References

- ▸ Arithmetic Coding for Data Compression
- ▸ Huffman Coding

Recommended Reading

1. Anh VN, Moffat A (2010) Index compression using 64-bit words. Softw Pract Exp 40(2):131–147
2. Boldi P, Vigna S (2005) Codes for the world-wide web. Internet Math 2(4):405–427
3. Brisaboa NR, Fariña A, Navarro G, Esteller MF (2003) (S, C)-dense coding: an optimized compression code for natural language text databases. In: Proceedings of the symposium on string processing and information retrieval, Manaus, pp 122–136
4. Culpepper JS, Moffat A (2005) Enhanced byte codes with restricted prefix properties. In: Proceedings of the symposium on string processing and information retrieval, Buenos Aires, pp 1–12
5. de Moura ES, Navarro G, Ziviani N, Baeza-Yates R (2000) Fast and flexible word searching on compressed text. ACM Trans Inf Syst 18(2):113–139
6. Elias P (1974) Efficient storage and retrieval by content and address of static files. J ACM 21(2):246–260
7. Elias P (1975) Universal codeword sets and representations of the integers. IEEE Trans Inf Theory IT-21(2):194–203
8. Fenwick P (2003) Universal codes. In: Sayood K (ed) Lossless compression handbook. Academic, Boston, pp 55–78
9. Fraenkel AS, Klein ST (1985) Novel compression of sparse bit-strings—preliminary report. In: Apostolico A, Galil Z (eds) Combinatorial algorithms on words. NATO ASI series F, vol 12. Springer, Berlin, pp 169–183

10. Golomb SW (1966) Run-length encodings. IEEE Trans Inf Theory IT–12(3):399–401
11. Lemire D, Boytsov L (2014, to appear) Decoding billions of integers per second through vectorization. Softw Pract Exp. http://dx.doi.org/10.1002/spe.2203
12. Moffat A, Anh VN (2006) Binary codes for locally homogeneous sequences. Inf Process Lett 99(5):75–80
13. Moffat A, Stuiver L (2000) Binary interpolative coding for effective index compression. Inf Retr 3(1):25–47
14. Moffat A, Turpin A (2002) Compression and coding algorithms. Kluwer Academic, Boston
15. Vigna S (2013) Quasi-succinct indices. In: Proceedings of the international conference on web search and data mining, Rome, pp 83–92
16. Zukowski M, Héman S, Nes N, Boncz P (2006) Super-scalar RAM-CPU cache compression. In: Proceedings of the international conference on data engineering, Atlanta. IEEE Computer Society, Washington, DC, paper 59

Computing Cutwidth and Pathwidth of Semi-complete Digraphs

Michał Pilipczuk
Institute of Informatics, University of Bergen, Bergen, Norway
Institute of Informatics, University of Warsaw, Warsaw, Poland

Keywords

Cutwidth; Minor; Pathwidth; Semi-complete digraph; Tournament

Years and Authors of Summarized Original Work

2012; Chudnovsky, Fradkin, Seymour
2013; Fradkin, Seymour
2013; Fomin, Pilipczuk
2013; Pilipczuk
2013; Fomin, Pilipczuk

Problem Definition

Recall that a simple digraph T is a tournament if for every two vertices $u, v \in V(T)$, exactly one of the arcs (u, v) and (v, u) exists in T. If we relax this condition by allowing both these arcs to exist at the same time, then we obtain the definition of a *semi-complete* digraph. We say that a digraph T contains a digraph H as a *topological minor* if one can map vertices of H to different vertices of T, and arcs of H to directed paths connecting respective images of the endpoints that are internally vertex disjoint. By relaxing vertex disjointness to arc disjointness, we obtain the definition of an *immersion*. (For simplicity, we neglect here the difference between weak immersions and strong immersions, and we work with weak immersions only.) Finally, we say that T contains H as a *minor* if vertices of H can be mapped to vertex disjoint strongly connected subgraphs of T in such a manner that for every arc (u, v) of H, there exists a corresponding arc of T going from a vertex belonging to the image of u to a vertex belonging to the image of v.

The topological minor, immersion, and minor relations form fundamental containment orderings on the class of digraphs. Mirroring the achievements of the graph minors program of Robertson and Seymour, it is natural to ask what is the complexity of testing these relations when the pattern graph H is assumed to be small. For general digraphs, even very basic problems of this nature are NP-complete [5]; however, the structure of semi-complete digraphs allow us to design efficient algorithms.

On semi-complete digraphs, the considered containment relations are tightly connected to digraph parameters *cutwidth* and *pathwidth*. For a digraph T and an ordering $(v_1, v_2, \ldots, v_n)$ of $V(T)$, by *width* of this ordering, we mean the maximum over $1 \leq t \leq n-1$ of the number of arcs going from $\{v_1, v_2, \ldots, v_t\}$ to $\{v_{t+1}, v_{t+2}, \ldots, v_n\}$ in T. The cutwidth of a digraph T, denoted by $\mathbf{ctw}(T)$, is the minimum width of an ordering of $V(T)$. A *path decomposition* of a digraph T is a sequence $\mathcal{P} = (W_1, W_2, \ldots, W_r)$ of subsets of vertices, called bags, such that (i) $\bigcup_{i=1}^{r} W_i = V(T)$, (ii) $W_j \supseteq W_i \cap W_k$ for all $1 \leq i < j < k \leq r$, and (iii) whenever (u, v) is an edge of T, then u and v appear together in some bag of $\mathcal{P}$ or all

the bags in which u appears are placed after all the bags in which v appears. The *width* of $\mathcal{P}$ is equal to $\max_{1 \leq i \leq r} |W_i| - 1$. The pathwidth of T, denoted by $\mathbf{pw}(T)$, is the minimum width of a path decomposition of T.

It appears that if a semi-complete digraph T excludes some digraph H as an immersion, then its cutwidth is bounded by a constant c_H depending on H only. Similarly, if T excludes H as a minor or as a topological minor, then its pathwidth is bounded by a constant p_H depending on H only. These Erdős-Pósa-style results were proven by Chudnovsky et al. [2] and Fradkin and Seymour [7], respectively. Based on this understanding of the links between containment relations and width parameters, it has been shown that immersion and minor relations are well-quasi-orderings of the class of semi-complete digraphs [1,6].

The aforementioned theorems give also raise to natural algorithms for testing the containment relations. We try to approximate the appropriate width measure: If we obtain a guarantee that it is larger than the respective constant c_H or p_H, then we can conclude that H is contained in T for sure. Otherwise, we can construct a decomposition of T of small width on which a dynamic programming algorithm can be employed. In fact, the proofs of Chudnovsky et al. [2] and Fradkin and Seymour [7] can be turned into (some) approximation algorithms for cutwidth and pathwidth on semi-complete digraphs. Therefore, it is natural to look for more efficient such algorithms, both in terms of the running time and the approximation ratio. The efficiency of an approximation subroutine is, namely, the crucial ingredient of the overall running time for testing containment relations.

Key Results

As a by-product of their proof, Chudnovsky et al. [2] obtained an algorithm that, given an n-vertex semi-complete digraph T and an integer k, either finds an ordering of $V(T)$ of width $\mathcal{O}(k^2)$ or concludes that $\mathbf{ctw}(T) > k$ by finding an appropriate combinatorial obstacle embedded in T. The running time is $\mathcal{O}(n^3)$. Similarly, a by-product of the proof of Fradkin and Seymour [7] is an algorithm that, for the same input, either finds a path decomposition of T of width $\mathcal{O}(k^2)$ or concludes that $\mathbf{pw}(T) > k$, again certifying this by providing an appropriate obstacle. Unfortunately, here the running time is $\mathcal{O}(n^{\mathcal{O}(k)})$; in other words, the exponent of the polynomial grows with k.

The proofs of the Erdős-Pósa statements proceed as follows: One shows that if the found combinatorial obstacle is large enough, i.e., it certifies that $\mathbf{ctw}(T) > k$ or $\mathbf{pw}(T) > k$ for large enough k, then already inside this obstacle one can find an embedding of every digraph H of a fixed size. Of course, the final values of constants c_H and p_H depend on how efficiently we can extract a model of H from an obstacle and, more precisely, how large k must be in terms of $|H|$ in order to guarantee that an embedding of H can be found. (We denote $|H| = |V(H)| + |E(H)|$.) Unfortunately, in the proofs of Chudnovsky et al. [2] and Fradkin and Seymour [7], this dependency is exponential (even multiple exponential in the case of p_H) and so is the overall dependency of constants c_H and p_H on $|H|$. Using the framework presented before, one can obtain an $f(|H|) \cdot n^3$-time algorithm for testing whether H can be immersed into an n-vertex semi-complete digraph T, as well as similar tests for the (topological) minor relations with running time $n^{g(|H|)}$. Here, f and g are some multiple-exponential functions.

The running time of the immersion testing algorithm is fixed-parameter tractable (FPT), while the running time for (topological) minor testing is only XP. It is natural to ask for an FPT algorithm also for the latter problem. Fomin and the current author [3,4,8] approached the issue from a different angle, which resulted in reproving the previous results with better constants, refined running times, and also in giving FPT algorithms for all the containment tests. Notably, the framework seems to be simpler and more uniform compared to the previous work. We now state the results of [3,4,8] formally, since they constitute the best known so far algorithms for topological problems in semi-complete digraphs.

Theorem 1 ([8]) *There exists an algorithm that, given an n-vertex semi-complete digraph T, runs in time $\mathcal{O}(n^2)$ and returns an ordering of $V(T)$ of width at most $\mathcal{O}(\mathbf{ctw}(T)^2)$.*

Theorem 2 ([8, 9]) *There exists an algorithm that, given an n-vertex semi-complete digraph T and an integer k, runs in time $\mathcal{O}(kn^2)$ and either returns a path decomposition of $V(T)$ of width at most $6k$ or correctly concludes that $\mathbf{pw}(T) > k$.*

Theorem 3 ([4, 8, 9]) *There exist algorithms that, given an n-vertex semi-complete digraph T and an integer k, determine whether:*

- $\mathbf{ctw}(T) \leq k$ *in time* $2^{\mathcal{O}(\sqrt{k \log k})} \cdot n^2$.
- $\mathbf{pw}(T) \leq k$ *in time* $2^{\mathcal{O}(k \log k)} \cdot n^2$.

Theorem 4 ([8]) *There exist algorithms that, given a digraph H and an n-vertex semi-complete digraph T, determine whether:*

- *H can be immersed in T in time* $2^{\mathcal{O}(|H|^2 \log |H|)} \cdot n^2$.
- *H is a topological minor of T in time* $2^{\mathcal{O}(|H| \log |H|)} \cdot n^2$.
- *H is a minor of T in time* $2^{\mathcal{O}(|H| \log |H|)} \cdot n^2$.

Thus, Theorems 1 and 2 provide approximation algorithms for cutwidth and pathwidth, Theorem 3 provides FPT algorithms for computing the exact values of these parameters, and Theorem 4 utilizes the approximation algorithms to give efficient algorithms for containment testing. We remark that the exact algorithm for cutwidth (the first bullet of Theorem 3) is a combination of the results of [8] (which gives a $2^{\mathcal{O}(k)} \cdot n^2$-time algorithm) and of [4] (which gives a $2^{\mathcal{O}(\sqrt{k \log k})} \cdot n^{\mathcal{O}(1)}$-time algorithm). A full exposition of this algorithm can be found in the PhD thesis of the current author [9], which contains a compilation of [3, 4, 8]. Moreover, for Theorem 2 work [8] claims only a 7-approximation, which has been consequently improved to a 6-approximation in the aforementioned PhD thesis [9]. Finally, it also follows that in the Erdős-Pósa results, one can take $c_H = \mathcal{O}(|H|^2)$ and $p_H = \mathcal{O}(|H|)$; this claim is not mentioned explicitly in [8], but follows easily from the results proven there.

To conclude, let us shortly deliberate on the approach that led to these results. The key to the understanding is the work [8]. The main observation there is that a large cluster of vertices with very similar outdegrees is already an obstacle for admitting a path decomposition of small width. More precisely, if one finds $4k + 2$ vertices whose outdegrees pairwise differ by at most k (a so-called $(4k + 2, k)$-*degree tangle*), then this certifies that $\mathbf{pw}(T) > k$; see Lemma 46 of [9]. As it always holds that $\mathbf{pw}(T) \leq 2\mathbf{ctw}(T)$, this conclusion also implies that $\mathbf{ctw}(T) > k/2$. Therefore, in semi-complete digraphs of small pathwidth or cutwidth, the outdegrees of vertices must be spread evenly; there is no "knot" with a larger density of vertices around some value of the outdegree. If we then order the vertices of T by their outdegrees, then this ordering should crudely resemble an ordering with the optimal width, as well as the order in which the vertices appear on an optimal path decomposition of T. Indeed, it can be shown that any ordering of $V(T)$ w.r.t. nondecreasing outdegrees has width at most $\mathcal{O}(\mathbf{ctw}(T)^2)$ [8]. Hence, the algorithm of Theorem 1 is, in fact, trivial: We just sort the vertices by their outdegrees. The pathwidth approximation algorithm (Theorem 2) is obtained by performing a left-to-right scan through the outdegree ordering that constructs a path decomposition in a greedy manner. For the exact algorithms (Theorem 3), in the scan we maintain a dynamic programming table of size exponential in k, whose entries correspond to possible endings of partial decompositions for prefixes of the ordering. The key to improving the running time for cutwidth to subexponential in terms of k (shown in [4]) is relating the states of the dynamic programming algorithm to *partition numbers*, a sequence whose subexponential asymptotics is well understood. Finally, the obstacles yielded by the approximation algorithms of Theorems 1 and 2 are more useful for finding embeddings of small digraphs H than the ones used in the previous works. This leads to a better dependence on $|H|$ of constants

c_H, p_H in the Erdős-Pósa results, as well as of the running times of the containment tests (Theorem 4).

Recommended Reading

1. Chudnovsky M, Seymour PD (2011) A well-quasi-order for tournaments. J Comb Theory Ser B 101(1):47–53
2. Chudnovsky M, Ovetsky Fradkin A, Seymour PD (2012) Tournament immersion and cutwidth. J Comb Theory Ser B 102(1):93–101
3. Fomin FV, Pilipczuk M (2013) Jungles, bundles, and fixed parameter tractability. In: Khanna S (ed) SODA, New Orleans. SIAM, pp 396–413
4. Fomin FV, Pilipczuk M (2013) Subexponential parameterized algorithm for computing the cutwidth of a semi-complete digraph. In: Bodlaender HL, Italiano GF (eds) ESA, Sophia Antipolis. Lecture notes in computer science, vol 8125. Springer, pp 505–516
5. Fortune S, Hopcroft JE, Wyllie J (1980) The directed subgraph homeomorphism problem. Theor Comput Sci 10:111–121
6. Kim I, Seymour PD (2012) Tournament minors. CoRR abs/1206.3135
7. Ovetsky Fradkin A, Seymour PD (2013) Tournament pathwidth and topological containment. J Comb Theory Ser B 103(3):374–384
8. Pilipczuk M (2013) Computing cutwidth and pathwidth of semi-complete digraphs via degree orderings. In: Portier N, Wilke T (eds) STACS, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, LIPIcs, vol 20, pp 197–208
9. Pilipczuk M (2013) Tournaments and optimality: new results in parameterized complexity. PhD thesis, University of Bergen, Norway. Available at the webpage of the author

Computing Pure Equilibria in the Game of Parallel Links

Spyros Kontogiannis
Department of Computer Science, University of Ioannina, Ioannina, Greece

Keywords

Convergence of Nash dynamics; Incentive compatible algorithms; Load balancing game; Nashification

Years and Authors of Summarized Original Work

2002; Fotakis, Kontogiannis, Koutsoupias, Mavronicolas, Spirakis
2003; Even-Dar, Kesselman, Mansour
2003; Feldman, Gairing, Lücking, Monien, Rode

Problem Definition

This problem concerns the construction of pure Nash equilibria (PNE) in a special class of atomic congestion games, known as the Parallel Links Game (PLG). The purpose of this note is to gather recent advances in the *existence and tractability of PNE in PLG*.

THE PURE PARALLEL LINKS GAME. Let $N \equiv [n]$ ($\forall k \in \mathbb{N}$, $[k] \equiv \{1, 2, \ldots, k\}$.) be a set of (selfish) players, each of them willing to have her good served by a *unique* shared resource (link) of a system. Let $E = [m]$ be the set of all these links. For each link $e \in E$, and each player $i \in N$,let $D_{i,e}(\cdot) : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$ be the **charging mechanism** according to which link e charges player i for using it. Each player $i \in [n]$ comes with a service requirement (e.g., traffic demand, or processing time) $W[i, e] > 0$, if she is to be served by link $e \in E$. A service requirement $W[i, e]$ is allowed to get the value ∞, to denote the fact that player i would never want to be assigned to link e. The charging mechanisms are functions of each link's cumulative congestion.

Any element $\sigma \in E$ is called a **pure strategy** for a player. Then, this player is assumed to assign her own good to link e. A collection of pure strategies for all the players is called a **pure strategies profile**, or a **configuration** of the players, or a **state** of the game.

The **individual cost** of player i wrt the profile σ is: $\mathtt{IC}_i(\sigma) = D_{i,\sigma_i}(\sum_{j \in [n]:\sigma_j = \sigma_i} W[j, \sigma_j])$. Thus, the **Pure Parallel Links Game** (PLG) is the game in strategic form defined as $\Gamma = \langle N, (\Sigma_i = E)_{i \in N}, (\mathtt{IC}_i)_{i \in N} \rangle$, whose acceptable solutions are only PNE. Clearly, an arbi-

trary instance of PLG can be described by the tuple $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$.

DEALING WITH SELFISH BEHAVIOR. The dominant solution concept for finite games in strategic form, is the Nash Equlibrium [14]. The definition of pure Nash Equilibria for PLG is the following:

Definition 1 (Pure Nash Equilibrium) For any instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ of PLG, a pure strategies profile $\sigma \in E^n$ is a **Pure Nash Equilibrium** (PNE in short), iff: $\forall i \in N, \forall e \in E, \mathtt{IC}_i(\sigma) = D_{i,\sigma_i}\left(\sum_{j\in[n]:\sigma_j=\sigma_i} W[j,\sigma_i]\right) \leq D_{i,e}\left(W[i,e] + \sum_{j\in[n]\setminus\{i\}:\sigma_j=e} W[j,e]\right)$.

A refinement of PNE are the ***k*- robust PNE**, for $n \geq k \geq 1$ [9]. These are pure profiles for which no subset of at most k players may concurrently change their strategies in such a way that the worst possible individual cost among the movers is *strictly decreased*.

QUALITY OF PURE EQUILIBRIA. In order to determine the quality of a PNE, a social cost function that measures it must be specified. The typical assumption in the literature of PLG, is that the social cost function is the worst individual cost paid by the players: $\forall \sigma \in E^n$, $\mathtt{SC}(\sigma) = \max_{i\in N}\{\mathtt{IC}_i(\sigma)\}$ and $\forall \mathbf{p} \in (\Delta_m)^n$, $\mathtt{SC}(\mathbf{p}) = \sum_{\sigma\in E^n}(\prod_{i\in N} p_i(\sigma_i)) \cdot \max_{i\in N}\{\mathtt{IC}_i(\sigma)\}$. Observe that, for mixed profiles, the social cost is the *expectation* of the maximum individual cost among the players.

The measure of the quality of an instance of PLG wrt PNE, is measured by the **Pure Price of Anarchy** (PPoA in short) [12]: $\mathtt{PPoA} = \max\{(\mathtt{SC}(\sigma))/\mathtt{OPT} : \sigma \in E^n \text{ is PNE}\}$ where $\mathtt{OPT} \equiv \min_{\sigma\in E^n}\{\mathtt{SC}(\sigma)\}$.

DISCRETE DYNAMICS. Crucial concepts of strategic games are the best and better responses. Given a configuration $\sigma \in E^n$, an **improvement step**, or **selfish step**, or **better response** of player $i \in N$ is the choice by i of a pure strategy $\alpha \in E \setminus \{\sigma_i\}$, so that player i would have a positive gain by this *unilateral* change (i.?e., provided that the other players maintain the same strategies). That is, $\mathtt{IC}_i(\sigma) > \mathtt{IC}_i(\sigma \oplus_i \alpha)$ where, $\sigma \oplus_i \alpha \equiv (\sigma_1, \ldots, \sigma_{i-1}, \alpha, \sigma_{i+1}, \ldots, \sigma_n)$. A **best response**, or **greedy selfish step** of player i, is any change from the current link σ_i to an element $\alpha^* \in \arg\max_{a\in E}\{\mathtt{IC}_i(\sigma \oplus_i \alpha)\}$. An **improvement path** (aka a **sequence of selfish steps** [6], or an **elementary step system** [3]) is a sequence of configurations $\pi = \langle\sigma(1), \ldots, \sigma(k)\rangle$ such that

$$\forall 2 \leq r \leq k, \exists i_r \in N, \exists \alpha_r \in E:$$
$$[\sigma(r) = \sigma(r-1) \oplus_{i_r} \alpha_r] \wedge [\mathtt{IC}_{i_r}(\sigma(r)) < \mathtt{IC}_{i_r}(\sigma(r-1))] .$$

A game has the **Finite Improvement Property** (FIP) iff any improvement path has *finite* length. A game has the **Finite Best Response Property** (FBRP) iff any improvement path, each step of whose is a best response of some player, has *finite* length.

An alternative trend is to, rather than consider *sequential* improvement paths, let the players conduct selfish improvement steps *concurrently*. Nevertheless, the selfish decisions are no longer deterministic, but rather distributions over the links, in order to have some notion of a priori Nash property that justifies these moves. The selfish players try to minimize their *expected* individual costs this time. Rounds of concurrent moves occur until a posteriori Nash Property is achieved. This is called a **selfish rerouting** policy [4].

Subclasses of PLG

[$\mathbf{PLG}_1$] Monotone PLG: The charging mechanism of each pair of a link and a player, is a *non–decreasing function* of the resource's cumulative congestion.

[$\mathbf{PLG}_2$] Resource Specific Weights PLG (RSPLG): Each player may have a different service demand from every link.

[$\mathbf{PLG}_3$] Player Specific Delays PLG (PSPLG): Each link may have a different charging mechanism for each player. Some special cases of PSPLG are the following:

[PLG$_{3.1}$] Linear Delays PSPLG: Every link has a (player specific) *affine* charging mechanism: $\forall i \in N, \forall e \in E, D_{i,e}(x) = a_{i,e}x + b_{i,e}$ for some $a_{i,e} > 0$ and $b_{i,e} \geq 0$.

[PLG$_{3.1.1}$] Related Delays PSPLG: Every link has a (player specific) *non–uniformly related* charging mechanism: $\forall i \in N, \forall e \in E, W[i,e] = w_i$ and $D_{i,e}(x) = a_{i,e}x$ for some $a_{i,e} > 0$.

[PLG$_4$] Resource Uniform Weights PLG (RUPLG): Each player has a unique service demand from all the resources. Ie, $\forall i \in N, \forall e \in E, W[i,e] = w_e > 0$. A special case of RUPLG is:

[PLG$_{4.1}$] Unweighted PLG: All the players have identical demands from all the links: $\forall i \in N, \forall e \in E, W[i,e] = 1$.

[PLG$_5$] Player Uniform Delays PLG (PUPLG): Each resource adopts a unique charging mechanism, for all the players. That is, $\forall i \in N, \forall e \in E, D_{i,e}(x) = d_e(x)$.

[PLG$_{5.1}$] Unrelated Parallel Machines, or **Load Balancing PLG (LBPLG)**: The links behave as parallel machines. That is, they charge each of the players for the cumulative load assigned to their hosts. One may think (wlog) that all the machines have as charging mechanisms the identity function. That is, $\forall i \in N, \forall e \in E, D_{i,e}(x) = x$.

[PLG$_{5.1.1}$] Uniformly Related Machines LBPLG: Each player has the same demand at every link, and each link serves players at a fixed rate. That is: $\forall i \in N, \forall e \in E, W[i,e] = w_i$ and $D_{i,e}(x) = \frac{x}{s_e}$. Equivalently, service demands proportional to the capacities of the machines are allowed, but the identity function is required as the charging mechanism: $\forall i \in N, \forall e \in E, W[i,e] = \frac{w_i}{s_e}$ and $D_{i,e}(x) = x$.

[PLG$_{5.1.1.1}$] Identical Machines LBPLG: Each player has the same demand at every link, and all the delay mechanisms are the identity function: $\forall i \in N, \forall e \in E, W[i,e] = w_i$ and $D_{i,e}(x) = x$.

[PLG$_{5.1.2}$] Restricted Assignment LBPLG: Each traffic demand is either of unit or infinite size. The machines are identical. Ie, $\forall i \in N, \forall e \in E, W[i,e] \in \{1, \infty\}$ and $D_{i,e}(x) = x$.

Algorithmic Questions Concerning PLG

The following algorithmic questions are considered:

Problem 1 (PNEExistsInPLG(E, N, W, D))
INPUT: An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ of PLG
OUTPUT: Is there a configuration $\sigma \in E^n$ of the players to the links, which is a PNE?

Problem 2 (PNEConstructionInPLG(E,N,W,D))
INPUT: An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ of PLG
OUTPUT: An assignment $\sigma \in E^n$ of the players to the links, which is a PNE.

Problem 3 (BestPNEInPLG(E, N, W, D))
INPUT: An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ of PLG. A **social cost** function $\mathtt{SC} : (\mathbb{R}_{\geq 0})^m \mapsto \mathbb{R}_{\geq 0}$ that characterizes the quality of any configuration $\sigma \in E^N$.
OUTPUT: An assignment $\sigma \in E^n$ of the players to the links, which is a PNE and minimizes the value of the social cost, compared to other PNE of PLG.

Problem 4 (WorstPNEInPLG(E, N, W, D))
INPUT: An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ of PLG. A **social cost** function $\mathtt{SC} : (\mathbb{R}_{\geq 0})^m \mapsto \mathbb{R}_{\geq 0}$ that characterizes the quality of any configuration $\sigma \in E^N$.
OUTPUT: An assignment $\sigma \in E^n$ of the players to the links, which is a PNE and *maximizes* the value of the social cost, compared to other PNE of PLG.

Problem 5 (DynamicsConvergeInPLG(E, N, W, D))
INPUT: An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ of PLG
OUTPUT: Does FIP (or FBRP) hold? How long does it take then to reach a PNE?

Problem 6 (ReroutingConvergeInPLG(E, N, W, D))
INPUT: An instance $\langle N, E, (W[i,e])_{i\in N, e\in E}, (D_{i,e}(\cdot))_{i\in N, e\in E}\rangle$ of PLG

OUTPUT: Compute (if any) a selfish rerouting policy that converges to a PNE.

Status of Problem 1

Player uniform, unweighted atomic congestion games always possess a PNE [15], with no assumption on monotonicity of the charging mechanisms. Thus, Problem 1 is already answered for all unweighted PUPLG. Nevertheless, this is not necessarily the case for weighted versions of PLG:

Theorem 1 ([13]) *There is an instance of (monotone) PSPLG with only three players and three strategies per player, possessing no PNE. On the other hand, any unweighted instance of monotone PSPLG possesses at least one PNE.*

Similar (positive) results were given for LBPLG. The key observation that lead to these results, is the fact that the lexicographically minimum vector of machine loads is always a PNE of the game.

Theorem 2 *There is always a PNE for any instance of Uniformly Related LBPLG* [7]*, and actually for any instance of LBPLG* [3]*. Indeed, there is a $k-$robust PNE for any instance of LBPLG, and any $1 \leq k \leq n$* [9].

Status of Problem 2, 5 and 6

Milchtaich [13] gave a constructive proof of existence for PNE in unweighted, monotone PSPLG, and thus implies a path of length at most n that leads to a PNE. Although this is a very efficient construction of PNE, it is not necessarily an improvement path, when all players are considered to coexist all the time, and therefore there is no justification for the adoption of such a path by the players. Milchtaich [13] proved that from an arbitrary initial configuration and allowing only best reply defections, there is a best reply improvement path of length at most $m \cdot \binom{n+1}{2}$. Finally, [11] proved for unweighted, Related PSPLG that it possesses FIP. Nevertheless, the convergence time is poor.

For LBPLG, the implicit connection of PNE construction to classical scheduling problems, has lead to quite interesting results.

Theorem 3 ([7]) *The LPT algorithm of Graham, yields a PNE for the case of Uniformly Related LBPLG, in time $\mathcal{O}(m \log m)$.*

The drawback of the LPT algorithm is that it is centralized and not selfishly motivated. An alternative approach, called **Nashification**, is to start from an arbitrary initial configuration $\sigma \in E^n$ and then try to construct a PNE of at most the same maximum individual cost among the players.

Theorem 4 ([6]) *There is an $\mathcal{O}(nm^2)$ time Nashification algorithm for any instance of Uniformly Related PLG.*

An alternative style of Nashification, is to let the players follow an arbitrary improvement path. Nevertheless, it is not always the case that this leads to a polynomial time construction of a PNE, as the following theorem states:

Theorem 5 *For Identical Machines LBPLG:*

- *There exist best response improvement paths of length $\Omega\left(\max\left\{2^{\sqrt{n}}, \left(\frac{n}{m^2}\right)^m\right\}\right)$* [3,6].
- *Any best response improvement path is of length $\mathcal{O}(2^n)$* [6].
- *Any best response improvement path, which gives priority to players of maximum weight among those willing to defect in each improvement step, is of length at most n* [3].
- *If all the service demands are integers, then any improvement path which gives priority to unilateral improvement steps, and otherwise allows only selfish 2-flips (ie, swapping of hosting machines between two goods) converges to a 2-robust PNE in at most $\frac{1}{2}(\sum_{i \in N} w_i)^2$ steps* [9].

The following result concerns selfish rerouting policies:

Theorem 6 ([4])

- *For unweighted Identical Machines LBPLG, a simple policy (BALANCE) forcing all the*

players of overloaded links to migrate to a new (random) link with probability proportional to the load of the link, converges to a PNE in $\mathcal{O}(\log\log n + \log m)$ rounds of concurrent moves. The same convergence time holds also for a simple Nash Rerouting Policy, in which each mover actually has an incentive to move.

- *For unweighted Uniformly Related LBPLG, BALANCE has the same convergence time, but the Nash Rerouting Policy may converge in $\Omega(\sqrt{n})$ rounds.*

Finally, a generic result of [5] is mentioned, that computes a PNE for arbitrary *unweighted, player uniform* symmetric network congestion games in polynomial time, by a nice exploitation of Rosenthal's potential and the solution of a proper minimum cost flow problem. Therefore, for PLG the following result is implied:

Theorem 7 ([5]) *For unweighted, monotone PUPLG, a PNE can be constructed in polynomial time.*

Of course, this result provides no answer, e.g., for Restricted Assignment LBPLG, for which it is still not known how to efficiently compute PNE.

Status of Problem 3 and 4

The proposed LPT algorithm of [7] for constructing PNE in Uniformly Related LBPLG, actually provides a solution which is at most $1.52 < \texttt{PPoA}(LPT) < 1.67$ times worse than the optimum PNE (which is indeed the allocation of the goods to the links that minimizes the make-span). The construction of the optimum, as well as the worst PNE are hard problems, which nevertheless admits a PTAS (in some cases):

Theorem 8 *For LBPLG with a social cost function as defined in the Quality of Pure Equilibria paragraph:*

- *For Identical Machines, constructing the optimum or the worst PNE is* **NP***–hard* [7].
- *For Uniformly Related Machines, there is a PTAS for the optimum PNE* [6].
- *For Uniformly Related Machines, it holds that* $\texttt{PPoA} = \Theta\left(\min\left\{(\log m)/(\log\log m), \log(s_{\max})/(s_{\min})\right\}\right)$ [2].
- *For the Restricted Assignments,* $\texttt{PPoA} = \Omega((\log m)/(\log\log m))$ [10].
- *For a generalization of the Restricted Assignments, where the players have goods of any positive, otherwise infinite service demands from the links (and not only elements of $\{1, \infty\}$), it holds that $m - 1 \leq \texttt{PPoA} < m$* [10].

It is finally mentioned that a recent result [1] for unweighted, single commodity network congestion games with linear delays, is translated to the following result for PLG:

Theorem 9 ([1]) *For unweighted PUPLG with linear charging mechanisms for the links, the worst case PNE may be a factor of* $\texttt{PPoA} = 5/2$ *away from the optimum solution, wrt the social cost defined in the Quality of Pure Equilibria paragraph.*

Key Results

None

Applications

Congestion games in general have attracted much attention from many disciplines, partly because they capture a large class of routing and resource allocation scenarios.

PLG in particular, is the most elementary (non–trivial) atomic congestion game among a large number of players. Despite its simplicity, it was proved ([8] that it is asymptotically the worst case instance wrt the maximum individual cost measure, for a large class atomic congestion games involving the so called *layered networks*. Therefore, PLG is considered an excellent starting point for studying congestion games in large scale networks.

The importance of seeking for PNE, rather than arbitrary (mixed in general) NE, is quite obvious in sciences like the economics, ecology,

and biology. It is also important for computer scientists, since it enforces deterministic costs to the players, and both the players and the network designer may feel safer in this case about what they will actually have to pay.

The question whether the Nash Dynamics converge to a PNE in a reasonable amount of time, is also quite important, since (in case of a positive answer) it justifies the selfish, decentralized, local dynamics that appear in large scale communications systems. Additionally, the selfish rerouting schemes are of great importance, since this is what should actually be expected from selfish, decentralized computing environments.

Open Problems

Open Question 1 *Determine the (in)existence of PNE for all the instances of PLG that do not belong in LBPLG, or in monotone PSPLG.*

Open Question 2 *Determine the (in)existence of $k-$robust PNE for all the instances of PLG that do not belong in LBPLG.*

Open Question 3 *Is there a polynomial time algorithm for constructing $k-$robust PNE, even for the Identical Machines LBPLG and $k \geq 1$ being a constant?*

Open Question 4 *Do the improvement paths of instances of PLG other than PSPLG and LBPLG converge to a PNE?*

Open Question 5 *Are there selfish rerouting policies of instances of PLG other than Identical Machines LBPLG converge to a PNE? How long much time would they need, in case of a positive answer?*

Cross-References

- Best Response Algorithms for Selfish Routing
- Price of Anarchy
- Selfish Unsplittable Flows: Algorithms for Pure Equilibria

Recommended Reading

1. Christodoulou G, Koutsoupias E (2005) The price of anarchy of finite congestion games. In: Proceedings of the 37th ACM symposium on theory of computing (STOC '05). ACM, Baltimore, pp 67–73
2. Czumaj A, Vöcking B (2002) Tight bounds for worst-case equilibria. In: Proceedings of the 13th ACM-SIAM symposium on discrete algorithms (SODA '02). SIAM, San Francisco, pp 413–420
3. Even-Dar E, Kesselman A, Mansour Y (2003) Convergence time to nash equilibria. In: Proceedings of the 30th international colloquium on automata, languages and programming (ICALP '03). LNCS. Springer, Eindhoven, pp 502–513
4. Even-Dar E, Mansour Y (2005) Fast convergence of selfish rerouting. In: Proceedings of the 16th ACM-SIAM symposium on discrete algorithms (SODA '05). SIAM, Vancouver, pp 772–781
5. Fabrikant A, Papadimitriou C, Talwar K (2004) The complexity of pure nash equilibria. In: Proceedings of the 36th ACM symposium on theory of computing (STOC '04). ACM, Chicago
6. Feldmann R, Gairing M, Lücking T, Monien B, Rode M (2003) Nashification and the coordination ratio for a selfish routing game. In: Proceedings of the 30th international colloquium on automata, languages and programming (ICALP '03). LNCS. Springer, Eindhoven, pp 514–526
7. Fotakis D, Kontogiannis S, Koutsoupias E, Mavronicolas M, Spirakis P (2002) The structure and complexity of nash equilibria for a selfish routing game. In: Proceedings of the 29th international colloquium on automata, languages and programming (ICALP '02). LNCS. Springer, Málaga, pp 123–134
8. Fotakis D, Kontogiannis S, Spirakis P (2005) Selfish unsplittable flows. Theor Comput Sci 348:226–239. Special Issue dedicated to ICALP (2004) (TRACK-A).
9. Fotakis D, Kontogiannis S, Spirakis P (2006) Atomic congestion games among coalitions. In: Proceedings of the 33rd international colloquium on automata, languages and programming (ICALP '06). LNCS, vol 4051. Springer, Venice, pp 572–583
10. Gairing M, Luecking T, Mavronicolas M, Monien B (2006) The price of anarchy for restricted parallel links. Parallel Process Lett 16:117–131, Preliminary version appeared in STOC 2004
11. Gairing M, Monien B, Tiemann K (2006) Routing (un-)splittable flow in games with player specific linear latency functions. In: Proceedings of the 33rd international colloquium on automata, languages and programming (ICALP '06). LNCS. Springer, Venice, pp 501–512
12. Koutsoupias E, Papadimitriou C (1999) Worst-case equilibria. In: Proceedings of the 16th annual symposium on theoretical aspects of computer science (STACS '99). Springer, Trier, pp 404–413

13. Milchtaich I (1996) Congestion games with player-specific payoff functions. Games Econ Behav 13:111–124
14. Nash J (1951) Noncooperative games. Ann Math 54:289–295
15. Rosenthal R (1973) A class of games possessing pure-strategy nash equilibria. Int J Game Theory 2:65–67

Concurrent Programming, Mutual Exclusion

Gadi Taubenfeld
Department of Computer Science,
Interdiciplinary Center Herzlia, Herzliya, Israel

Keywords

Critical section problem

Years and Authors of Summarized Original Work

1965; Dijkstra

Problem Definition

Concurrency, Synchronization and Resource Allocation

A *concurrent* system is a collection of processors that communicate by reading and writing from a shared memory. A distributed system is a collection of processors that communicate by sending messages over a communication network. Such systems are used for various reasons: to allow a large number of processors to solve a problem together much faster than any processor can do alone, to allow the distribution of data in several locations, to allow different processors to share resources such as data items, printers or discs, or simply to enable users to send electronic mail.

A *process* corresponds to a given computation. That is, given some program, its execution is a process. Sometimes, it is convenient to refer to the program code itself as a process. A process runs on a *processor*, which is the physical hardware. Several processes can run on the same processor although in such a case only one of them may be active at any given time. Real concurrency is achieved when several processes are running simultaneously on several processors.

Processes in a concurrent system often need to synchronize their actions. *Synchronization* between processes is classified as either cooperation or contention. A typical example for cooperation is the case in which there are two sets of processes, called the producers and the consumers, where the producers produce data items which the consumers then consume.

Contention arises when several processes compete for exclusive use of shared resources, such as data items, files, discs, printers, etc. For example, the integrity of the data may be destroyed if two processes update a common file at the same time, and as a result, deposits and withdrawals could be lost, confirmed reservations might have disappeared, etc. In such cases it is sometimes essential to allow at most one process to use a given resource at any given time.

Resource allocation is about interactions between processes that involve contention. The problem is, how to resolve conflicts resulting when several processes are trying to use shared resources. Put another way, how to allocate shared resources to competing processes. A special case of a general resource allocation problem is the *mutual exclusion* problem where only a single resource is available.

The Mutual Exclusion Problem

The *mutual exclusion* problem, which was first introduced by Edsger W. Dijkstra in 1965, is the guarantee of mutually exclusive access to a single shared resource when there are several competing processes [6]. The problem arises in operating systems, database systems, parallel supercomputers, and computer networks, where it is necessary to resolve conflicts resulting when several processes are trying to use shared resources. The problem is of great significance, since it lies at the heart of many interprocess synchronization problems.

The problem is formally defined as follows: it is assumed that each process is executing a sequence of instructions in an infinite loop. The instructions are divided into four continuous sections of code: the *remainder, entry, critical section* and *exit*. Thus, the structure of a mutual exclusion solution looks as follows:

loop forever
remainder code;
entry code;
critical section;
exit code
end loop

A process starts by executing the remainder code. At some point the process might need to execute some code in its critical section. In order to access its critical section a process has to go through an entry code which guarantees that while it is executing its critical section, no other process is allowed to execute its critical section. In addition, once a process finishes its critical section, the process executes its exit code in which it notifies other processes that it is no longer in its critical section. After executing the exit code the process returns to the remainder.

The Mutual exclusion problem is to write the code for the *entry code* and the *exit code* in such a way that the following two basic requirements are satisfied.

Mutual exclusion: *No two processes are in their critical sections at the same time.*

Deadlock-freedom: *If a process is trying to enter its critical section, then some process, not necessarily the same one, eventually enters its critical section.*

The deadlock-freedom property guarantees that the system as a whole can always continue to make progress. However deadlock-freedom may still allow "starvation" of individual processes. That is, a process that is trying to enter its critical section, may never get to enter its critical section, and wait forever in its entry code. A stronger requirement, which does not allow starvation, is defined as follows.

Starvation-freedom: *If a process is trying to enter its critical section, then this process must eventually enter its critical section.*

Although starvation-freedom is strictly stronger than deadlock-freedom, it still allows processes to execute their critical sections arbitrarily many times before some trying process can execute its critical section. Such a behavior is prevented by the following fairness requirement.

First-in-first-out (FIFO): *No beginning process can enter its critical section before a process that is already waiting for its turn to enter its critical section.*

The first two properties, mutual exclusion and deadlock freedom, were required in the original statement of the problem by Dijkstra. They are the minimal requirements that one might want to impose. In solving the problem, it is assumed that once a process starts executing its critical section the process always finishes it regardless of the activity of the other processes. Of all the problems in interprocess synchronization, the mutual exclusion problem is the one studied most extensively. This is a deceptive problem, and at first glance it seems very simple to solve.

Key Results

Numerous solutions for the problem have been proposed since it was first introduced by Edsger W. Dijkstra in 1965 [6]. Because of its importance and as a result of new hardware and software developments, new solutions to the problem are still being designed. Before the results are discussed, few models for interprocess communication are mentioned.

Atomic Operations

Most concurrent solutions to the problem assumes an architecture in which n processes communicate asynchronously via a shared objects. All architectures support *atomic registers*, which are shared objects that support atomic reads and writes operations. A weaker notion than an atomic register, called a *safe* register, is also considered in the literature. In a safe register, a read not concurrent with any writes must obtain the correct value, however, a read that is concurrent with some write, may return an arbitrary value. Most modern

architectures support also some form of atomicity which is stronger than simple reads and writes. Common atomic operations have special names. Few examples are,

- *Test-and-set*: takes a shared registers r and a value *val*. The value *val* is assigned to r, and the old value of r is returned.
- *Swap*: takes a shared registers r and a local register ℓ, and atomically exchange their values.
- *Fetch-and-increment*: takes a register r. The value of r is incremented by 1, and the old value of r is returned.
- *Compare-and-swap*: takes a register r, and two values: *new* and *old*. If the current value of the register r is equal to *old*, then the value of r is set to *new* and the value *true* is returned; otherwise r is left unchanged and the value *false* is returned.

Modern operating systems (such as Unix and Windows) implement synchronization mechanisms, such as semaphores, that simplify the implementation of mutual exclusion locks and hence the design of concurrent applications. Also, modern programming languages (such as Modula and Java) implement the monitor concept which is a program module that is used to ensure exclusive access to resources.

Algorithms and Lower Bounds

There are hundreds of beautiful algorithms for solving the problem some of which are also very efficient. Only few are mentioned below. First algorithms that use only atomic registers, or even safe registers, are discussed.

The Bakery Algorithm. The Bakery algorithm is one of the most known and elegant mutual exclusion algorithms using only safe registers [9]. The algorithm satisfies the FIFO requirement, however it uses unbounded size registers. A modified version, called the Black-White Bakery algorithm, satisfies FIFO and uses bounded number of bounded size atomic registers [14].

Lower bounds. A space lower bound for solving mutual exclusion using only atomic registers is that: any deadlock-free mutual exclusion algorithm for n processes must use at least n shared registers [5]. It was also shown in [5] that this bound is tight. A time lower bound for any mutual exclusion algorithm using atomic registers is that: there is no a priori bound on the number of steps taken by a process in its entry code until it enters its critical section (counting steps only when no other process is in its critical section or exit code) [2]. Many other interesting lower bounds exist for solving mutual exclusion.

A Fast Algorithm. A *fast* mutual exclusion algorithm, is an algorithm in which in the absence of contention only a constant number of shared memory accesses to the shared registers are needed in order to enter and exit a critical section. In [10], a fast algorithm using atomic registers is described, however, in the presence of contention, the winning process may have to check the status of all other n processes before it is allowed to enter its critical section. A natural question to ask is whether this algorithm can be improved for the case where there is contention.

Adaptive Algorithms. Since the other contending processes are waiting for the winner, it is particularly important to speed their entry to the critical section, by the design of an *adaptive* mutual exclusion algorithm in which the time complexity is independent of the total number of processes and is governed only by the current degree of contention. Several (rather complex) adaptive algorithms using atomic registers are known [1, 3, 14]. (Notice that, the time lower bound mention earlier implies that no adaptive algorithm using only atomic registers exists when time is measured by counting *all* steps.)

Local-spinning Algorithms. Many algorithms include busy-waiting loops. The idea is that in order to wait, a process *spins* on a flag register, until some other process terminates the spin with a single write operation. Unfortunately, under contention, such spinning may generate lots of traffic on the interconnection network between the process and the memory. An algorithm satisfies local spinning if the only type of spinning required is local spinning. Local Spinning is the situation where a process is spinning on locally-accessible registers. Shared registers may

be locally-accessible as a result of either coherent caching or when using distributed shared memory where shared memory is physically distributed among the processors.

Three local-spinning algorithms are presented in [4, 8, 11]. These algorithms use strong atomic operations (i.e., fetch-and-increment, swap, compare-and-swap), and are also called *scalable* algorithms since they are both local-spinning and adaptive. Performance studies done, have shown that these algorithms scale very well as contention increases. Local spinning algorithms using only atomic registers are presented in [1, 3, 14].

Only few representative results have been mentioned. There are dozens of other very interesting algorithms and lower bounds. All the results discussed above, and many more, are described details in [15]. There are also many results for solving mutual exclusion in distributed message passing systems [13].

Applications

Synchronization is a fundamental challenge in computer science. It is fast becoming a major performance and design issue for concurrent programming on modern architectures, and for the design of distributed and concurrent systems.

Concurrent access to resources shared among several processes must be synchronized in order to avoid interference between conflicting operations. Mutual exclusion *locks* (i.e., algorithms) are the de facto mechanism for concurrency control on concurrent applications: a process accesses the resource only inside a critical section code, within which the process is guaranteed exclusive access. The popularity of this approach is largely due the apparently simple programming model of such locks and the availability of implementations which are efficient and scalable. Essentially all concurrent programs (including operating systems) use various types of mutual exclusion locks for synchronization.

When using locks to protect access to a resource which is a large data structure (or a database), the *granularity* of synchronization is important. Using a single lock to protect the whole data structure, allowing only one process at a time to access it, is an example of *coarse-grained* synchronization. In contrast, *fine-grained* synchronization enables to lock "small pieces" of a data structure, allowing several processes with non-interfering operations to access it concurrently. Coarse-grained synchronization is easier to program but is less efficient and is not fault-tolerant compared to fine-grained synchronization. Using locks may degrade performance as it enforces processes to wait for a lock to be released. In few cases of simple data structures, such as queues, stacks and counters, locking may be avoided by using lock-free data structures.

Cross-References

- Registers
- Self-Stabilization

Recommended Reading

In 1968, Edsger Wybe Dijkstra has published his famous paper "Co-operating sequential processes" [7], that originated the field of concurrent programming. The mutual exclusion problem was first stated and solved by Dijkstra in [6], where the first solution for two processes, due to Dekker, and the first solution for n processes, due to Dijkstra, have appeared. In [12], a collection of some early algorithms for mutual exclusion are described. In [15], dozens of algorithms for solving the mutual exclusion problems and wide variety of other synchronization problems are presented, and their performance is analyzed according to precise complexity measures.

1. Afek Y, Stupp G, Touitou D (2002) Long lived adaptive splitter and applications. Distrib Comput 30:67–86
2. Alur R, Taubenfeld G (1992) Results about fast mutual exclusion. In: Proceedings of the 13th IEEE real-time systems symposium, Dec 1992, pp 12–21
3. Anderson JH, Kim Y-J (2000) Adaptive mutual exclusion with local spinning. In: Proceedings of the 14th international symposium on distributed computing, Lecture notes in computer science, vol 1914, pp 29–43
4. Anderson TE (1990) The performance of spin lock alternatives for shared-memory multiprocessor. IEEE Trans Parallel Distrib Syst 1(1):6–16

5. Burns JN, Lynch NA (1993) Bounds on shared-memory for mutual exclusion. Inf Comput 107(2):171–184
6. Dijkstra EW (1965) Solution of a problem in concurrent programming control. Commun ACM 8(9):569
7. Dijkstra EW (1968) Co-operating sequential processes. In: Genuys F (ed) Programming languages. Academic, New York, pp 43–112. Reprinted from: technical report EWD-123, Technological University, Eindhoven (1965)
8. Graunke G, Thakkar S (1990) Synchronization algorithms for shared-memory multiprocessors. IEEE Comput 28(6):69–69
9. Lamport L (1974) A new solution of Dijkstra's concurrent programming problem. Commun ACM 17(8):453–455
10. Lamport L (1987) A fast mutual exclusion algorithm. ACM Trans Comput Syst 5(1):1–11
11. Mellor-Crummey JM, Scott ML (1991) Algorithms for scalable synchronization on shared-memory multiprocessors. ACM Trans Comput Syst 9(1):21–65
12. Raynal M (1986) Algorithms for mutual exclusion. MIT, Cambridge. Translation of: Algorithmique du parallélisme (1984)
13. Singhal M (1993) A taxonomy of distributed mutual exclusion. J Parallel Distrib Comput 18(1):94–101
14. Taubenfeld G (2004) The black-white bakery algorithm. In: 18th international symposium on distributed computing, Oct 2004. LNCS, vol 3274. Springer, Berlin, pp 56–70
15. Taubenfeld G (2006) Synchronization algorithms and concurrent programming. Pearson Education/Prentice-Hall, Upper Saddle River. ISBN:0131972596

Connected Dominating Set

Feng Wang[1], Ding-Zhu Du[2,4], and Xiuzhen Cheng[3]
[1]Mathematical Science and Applied Computing, Arizona State University at the West Campus, Phoenix, AZ, USA
[2]Computer Science, University of Minnesota, Minneapolis, MN, USA
[3]Department of Computer Science, George Washington University, Washington, DC, USA
[4]Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Keywords

Techniques for partition

Years and Authors of Summarized Original Work

2003; Cheng, Huang, Li, Wu, Du

Problem Definition

Consider a graph $G = (V, E)$. A subset C of V is called a *dominating set* if every vertex is either in C or adjacent to a vertex in C. If, furthermore, the subgraph induced by C is connected, then C is called a *connected dominating set*. A connected dominating set with a minimum cardinality is called a *minimum connected dominating set (MCDS)*. Computing an MCDS is an NP-hard problem and there is no polynomial-time approximation with performance ratio $\rho H(\Delta)$ for $\rho < 1$ unless $NP \subseteq DTIME(n^{O(\ln \ln n)})$ where H is the harmonic function and Δ is the maximum degree of the input graph [11].

A unit disk is a disk with radius one. A *unit disk graph (UDG)* is associated with a set of unit disks in the Euclidean plane. Each node is at the center of a unit disk. An edge exists between two nodes u and v if and only if $|uv| \leq 1$ where $|uv|$ is the Euclidean distance between u and v. This means that two nodes u and v are connected with an edge if and only if u's disk covers v and v's disk covers u.

Computing an MCDS in a unit disk graph is still NP-hard. How hard is it to construct a good approximation for MCDS in unit disk graphs? Cheng et al. [5] answered this question by presenting a polynomial-time approximation scheme.

Historical Background

The connected dominating set problem has been studied in graph theory for many years [23]. However, recently it becomes a hot topic due to its application in wireless networks for virtual backbone construction [4]. Guha and Khuller [11] gave a two-stage greedy approximation for the minimum connected dominating set in general graphs and showed that its performance ratio is $3 + \ln \Delta$ where Δ is the maximum node degree in the graph. To design a one-step greedy approximation to reach a similar performance

ratio, the difficulty is to find a submodular potential function. In [22], Ruan et al. successfully designed a one-step greedy approximation that reaches a better performance ratio $c + \ln \Delta$ for any $c > 2$. Du et al. [7] showed that there exists a polynomial-time approximation with a performance ratio $a(1 + \ln \Delta)$ for any $a > 1$. The importance of those works is that the potential functions used in their greedy algorithm are non-submodular and they managed to complete its theoretical performance evaluation with fresh ideas.

Guha and Khuller [11] also gave a negative result that there is no polynomial-time approximation with a performance ratio $\rho \ln \Delta$ for $\rho < 1$ unless $NP \subseteq DTIME(n^{O(\ln \ln n)})$. As indicated by [9], dominating sets cannot be approximated arbitrarily well, unless P almost equal to NP. These results move ones' attention from general graphs to unit disk graphs because the unit disk graph is the model for wireless sensor networks, and in unit disk graphs, MCDS has a polynomial-time approximation with a constant performance ratio. While this constant ratio is getting improved step-by-step [1, 2, 20, 25], Cheng et al. [5] closed this story by showing the existence of a polynomial-time approximation scheme (PTAS) for the MCDS in unit disk graphs. This means that theoretically, the performance ratio for polynomial-time approximation can be as small as $1+\varepsilon$ for any positive number ε.

Dubhashi et al. [8] showed that once a dominating set is constructed, a connected dominating set can be easily computed in a distributed fashion. Most centralized results for dominating sets are available at [19]. In particular, a simple constant approximation for dominating sets in unit disk graphs was presented in [19]. Constant-factor approximation for minimum-weight (connected) dominating sets in UDGs was studied in [3]. A PTAS for the minimum dominating set problem in UDGs was proposed in [21]. Kuhn et al. [16] proved that a maximal independent set (MIS) (and hence also a dominating set) can be computed in asymptotically optimal time $O(\log n)$ in UDGs and a large class of bounded independence graphs. Luby [18] reported an elegant local $O(\log n)$ algorithm for MIS on general graphs. Jia et al. [12] proposed a fast $O(\log n)$ distributed approximation for dominating set in general graphs. The first constant-time distributed algorithm for dominating sets that achieves a nontrivial approximation ratio for general graphs was reported in [13]. The matching $\Omega(\log n)$ lower bound is considered to be a classic result in distributed computing [17]. For UDGs a PTAS is achievable in a distributed fashion [15]. The fastest deterministic distributed algorithm for dominating sets in UDGs was reported in [14], and the fastest randomized distributed algorithm for dominating sets in UDGs was presented in [10].

Key Results

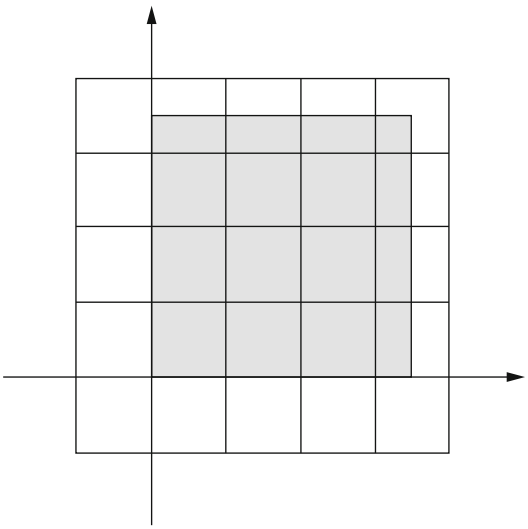
The construction of PTAS for MCDS is based on the fact that there is a polynomial-time approximation with a constant performance ratio. Actually, this fact is quite easy to see. First, note that a unit disk contains at most five independent vertices [2]. This implies that every maximal independent set has a size at most $1 + 4opt$ where opt is the size of an MCDS. Moreover, every maximal independent set is a dominating set and it is easy to construct a maximal independent set with a spanning tree of all edges with length two. All vertices in this spanning tree form a connected dominating set of a size at most $1 + 8opt$. By improving the upper bound for the size of a maximal independent set [26] and the way to interconnecting a maximal independent set [20], the constant ratio has been improved to 6.8 with a distributed implementation.

The basic techniques in this construction are nonadaptive partition and shifting. Its general picture is as follows: First, the square containing all vertices of the input unit disk graph is divided into a grid of small cells. Each small cell is further divided into two areas, the central area and the boundary area. The central area consists of points h distance away from the cell boundary. The boundary area consists of points within distance $h + 1$ from the boundary. Therefore, two areas are overlapping. Then a minimum union of connected dominating sets is computed in each cell

for connected components of the central area of the cell. The key lemma is to prove that the union of all such minimum unions is no more than the minimum connected dominating set for the whole graph. For vertices not in central areas, just use the part of an 8-approximation lying in boundary areas to dominate them. This part together with the above union forms a connected dominating set for the whole input unit disk graph. By shifting the grid around to get partitions at different coordinates, a partition having the boundary part with a very small upper bound can be obtained.

The following details the construction.

Given an input connected unit disk graph $G = (V, E)$ residing in a square $Q = \{(x, y)|0 \le x \le q, 0 \le y \le q\}$ where $q \le |V|$. To construct an approximation with a performance ratio $1 + \varepsilon$ for $\varepsilon > 0$, choose an integer $m = O((1/\varepsilon)\ln(1/\varepsilon))$. Let $p = \lfloor q/m \rfloor + 1$. Consider the square Q''. Partition $\bar{Q}$ into $(p + 1) \times (p + 1)$ grids so that each cell is an $m \times m$ square excluding the top and the right boundaries, and hence, no two cells are overlapping each other. This partition of $\bar{Q}$s is denoted by $P(0)$ (Fig. 1). In general, the partition $P(a)$ is obtained from $P(0)$ by shifting the bottom-left corner of $\bar{Q}$ from $(-m, -m)$ to $(-m + a, -m + a)$. Note that shifting from $P(0)$ to $P(a)$ for $0 \le a \le m$ keeps Q covered by the partition.



Connected Dominating Set, Fig. 1 Squares Q and $\bar{Q}$

For each cell e (an $m \times m$ square), $C_e(d)$ denotes the set of points in e away from the boundary by distance at least d, e.g., $C_e(0)$ is the cell e itself. Denote $B_e(d) = C_e(0) - C_e(d)$. Fix a positive integer $h = 7 + 3\lfloor \log_2(4m^2/\pi) \rfloor$. Call $C_e(h)$ the *central area* of e and $B_e(h + 1)$ the *boundary area* of e. Hence, the boundary area and the central area of each cell are overlapping with width one.

C

Central Area

Let $G_e(d)$ denote the part of input graph G lying in area $C_e(d)$. In particular, $G_e(h)$ is the part of graph G lying in the central area of e. $G_e(h)$ may consist of several connected components. Let K_e be a subset of vertices in $G_e(0)$ with a minimum cardinality such that for each connected component H of $G_e(h)$, K_e contains a connected component dominating H. In other words, K_e is a minimum union of connected dominating sets in $G(0)$ for the connected components of $G_e(h)$.

Now, denote by $K(a)$ the union of K_e for e over all cells in partition $P(a)$. $K(a)$ has two important properties:

Lemma 1 *$K(a)$ can be computed in time $n^{O(m2)}$.*

Lemma 2 *$|K^a| \le opt$ for $0 \le a \le m - 1$.*

Lemma 1 is not hard to see. Note that in a square with edge length $\sqrt{2}/2$, all vertices induce a complete subgraph in which any vertex must dominate all other vertices. It follows that the minimum dominating set for the vertices of $G_e(0)$ has size at most $\left(\left\lfloor \sqrt{2}m \right\rfloor\right)^2$. Hence, the size of K_e is at most $3\left(\left\lfloor \sqrt{2}m \right\rfloor\right)^2$ because any dominating set in a connected graph has a spanning tree with an edge length at most three. Suppose cell $G_e(0)$ has n_e vertices. Then the number of candidates for K_e is at most

$$\sum_{k=0}^{3(\lceil \sqrt{2}m \rceil)^2} \binom{n_e}{k} = n_e^{O(m^2)}.$$

Hence, computing $K(a)$ can be done in time

$$\sum_e n_e^{O(m^2)} \le \left(\sum_e n_e\right)^{O(m^2)} = n^{O(m^2)}.$$

However, the proof of Lemma 2 is quite tedious. The reader who is interested in it may find it in [5].

Boundary Area

Let F be a connected dominating set of G satisfying $|F| \le 8opt + 1$. Denote by $F(a)$ the subset of F lying in the boundary area $B_a(h+1)$. Since F is constructed in polynomial time, only the size of $F(a)$ needs to be studied.

Lemma 3 *Suppose* $h = 7 + 3\lfloor \log_2(4m^2/\pi)\rfloor$ *and* $\lfloor m/(h+1)\rfloor \ge 32/\varepsilon$. *Then there is at least half of* $i = 0, 1, \ldots, \lfloor m/(h+1)\rfloor - 1$ *such that* $|F(i(h+1))| \le \varepsilon \cdot opt$.

Proof Let $F_H(a)$ ($F_V(a)$) denote the subset of vertices in $F(a)$ each with distance $< h+1$ from the horizontal (vertical) boundary of some cell in $P(a)$. Then $F(a) = F_H(a) \cup F_V(a)$. Moreover, all $F_H(i(h+1))$ for $i = 0, 1, \ldots, \lfloor m/(h+1)\rfloor - 1$ are disjoint. Hence,

$$\sum_{i=0}^{\lfloor m/(h+1)\rfloor - 1} |F_H(i(h+1))| \le |F| \le 8opt.$$

Similarly, all $F_V(i(h+1))$ for $i = 0, 1, \ldots, \lfloor m/(h+1)\rfloor - 1$ are disjoint and

$$\sum_{i=0}^{\lfloor m/(h+1)\rfloor - 1} |F_V(i(h+1))| \le |F| \le 8opt.$$

Thus,

$$\sum_{i=0}^{\lfloor m/(h+1)\rfloor - 1} |F(i(h+1))| \le \sum_{i=0}^{\lfloor m/(h+1)\rfloor - 1} (|F_H(i(h+1))| + |F_V(i(h+1))|) \le 16opt.$$

That is,

$$\frac{1}{\lfloor m/(h+1)\rfloor} \sum_{i=0}^{\lfloor m/(h+1)\rfloor - 1} |F(i(h+1))| \le (\varepsilon/2)opt.$$

This means that there are at least half of $F(i(h+1))$ for $i = 0, 1, \lfloor m/(h+1)\rfloor - 1$ satisfying

$$|F(i(h+1))| \le \varepsilon \cdot opt.$$

□

Putting Together

Now put $K(a)$ and $F(a)$. By Lemmas 2 and 3, there exists $a \in \{0, h+1, \ldots, (\lfloor m/(h+1)\rfloor - 1)(h+1)\}$ such that

$$|K(a) \cup F(a)| \le (1+\varepsilon)opt.$$

Lemma 4 *For* $0 \le a \le m-1$, $K(a) \cup F(a)$ *is a connected dominating for input connected graph* G.

Proof $K(a) \cup F(a)$ is clearly a dominating set for input graph G. Its connectivity can be shown as follows. Note that the central area and the boundary area are overlapping with an area of width one. Thus, for any connected component H of the subgraph $G_e(h)$, $F(a)$ has a vertex in H. Hence, $F(a)$ must connect to any connected dominating set for H, especially, the one D_H in $K(a)$. This means that D_H is making up the connections of F lost from cutting a part in H. Therefore, the connectivity of $K(a) \cup F(a)$ follows from the connectivity of F. □

By summarizing the above results, the following result is obtained:

Theorem 1 *There is a* $(1+\varepsilon)$*-approximation for MCDS in connected unit disk graphs, running in time* $n^{O((1/\varepsilon)log(1/\varepsilon)2)}$.

Applications

An important application of connected dominating sets is to construct virtual backbones for wireless networks, especially, wireless sensor net-

works [4]. The topology of a wireless sensor network is often a unit disk graph.

Open Problems

In general, the topology of a wireless network is a disk graph, that is, each vertex is associated with a disk. Different disks may have different sizes. There is an edge from vertex u to vertex v if and only if the disk at u covers v. A virtual backbone in disk graphs is a subset of vertices, which induces a strongly connected subgraph, such that every vertex not in the subset has an in-edge coming from a vertex in the subset and also has an out-edge going into a vertex in the subset. Such a virtual backbone can be considered as a *connected dominating set* in disk graph. Is there a polynomial-time approximation with a constant performance ratio? It is still open right now [6]. Thai et al. [24] has made some effort towards this direction.

Cross-References

- Enumeration of Paths, Cycles, and Spanning Trees
- Exact Algorithms for Dominating Set
- Greedy Set-Cover Algorithms

Recommended Reading

1. Alzoubi KM, Wan P-J, Frieder O (2002) Message-optimal connected dominating sets in mobile ad hoc networks. In: ACM MOBIHOC, Lausanne, 09–11 June 2002
2. Alzoubi KM, Wan P-J, Frieder O (2002) New distributed algorithm for connected dominating set in wireless ad hoc networks. In: HICSS35, Hawaii
3. Ambuhl C, Erlebach T, Mihalak M, Nunkesser M (2006) Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In: Approximation, randomization, and combinatorial optimization. Algorithms and techniques. LNCS, vol 4110. Springer, Berlin, pp 3–14
4. Blum J, Ding M, Thaeler A, Cheng X (2004) Applications of connected dominating sets in wireless networks. In: Du D-Z, Pardalos P (eds) Handbook of combinatorial optimization. Kluwer, pp 329–369
5. Cheng X, Huang X, Li D, Wu W, Du D-Z (2003) A polynomial-time approximation scheme for minimum connected dominating set in ad hoc wireless networks. Networks 42:202–208
6. Du D-Z, Wan P-J (2012) Connected dominating set: theory and applications. Springer, New York
7. Du D-Z, Graham RL, Pardalos PM, Wan P-J, Wu W, Zhao W (2008) Analysis of greedy approximations with nonsubmodular potential functions. In: Proceedings of the 19th annual ACM-SIAM symposium on discrete algorithms (SODA), San Francisco, pp 167–175
8. Dubhashi D, Mei A, Panconesi A, Radhakrishnan J, Srinivasan A (2003) Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. In: SODA, Baltimore, pp 717–724
9. Feige U (1998) A threshold of ln n for approximating set cover. J ACM 45(4):634–652
10. Gfeller B, Vicari E (2007) A randomized distributed algorithm for the maximal independent set problem in growth-bounded graphs. In: PODC, Portland
11. Guha S, Khuller S (1998) Approximation algorithms for connected dominating sets. Algorithmica 20:374–387
12. Jia L, Rajaraman R, Suel R (2001) An efficient distributed algorithm for constructing small dominating sets. In: PODC, Newport
13. Kuhn F, Wattenhofer R (2003) Constant-time distributed dominating set approximation. In: PODC, Boston
14. Kuhn F, Moscibroda T, Nieberg T, Wattenhofer R (2005) Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In: DISC, Cracow
15. Kuhn F, Moscibroda T, Nieberg T, Wattenhofer R (2005) Local approximation schemes for ad hoc and sensor networks. In: DIALM-POMC, Cologne
16. Kuhn F, Moscibroda T, Wattenhofer R (2005) On the locality of bounded growth. In: PODC, Las Vegas
17. Linial N (1992) Locality in distributed graph algorithms. SIAM J Comput 21(1): 193–201
18. Luby M (1986) A simple parallel algorithm for the maximal independent set problem. SIAM J Comput 15:1036–1053
19. Marathe MV, Breu H, Hunt III HB, Ravi SS, Rosenkrantz DJ (1995) Simple heuristics for unit disk graphs. Networks 25:59–68
20. Min M, Du H, Jia X, Huang X, Huang C-H, Wu W (2006) Improving construction for connected dominating set with Steiner tree in wireless sensor networks. J Glob Optim 35: 111–119
21. Nieberg T, Hurink JL (2006) A PTAS for the minimum dominating set problem in unit disk graphs. In: Approximation and online algorithms. LNCS, vol 3879. Springer, Berlin, pp 296–306
22. Ruan L, Du H, Jia X, Wu W, Li Y, Ko K-I (2004) A greedy approximation for minimum connected dominating set. Theor Comput Sci 329:325–330

23. Sampathkumar E, Walikar HB (1979) The connected domination number of a graph. J Math Phys Sci 13:607–613
24. Thai MT, Wang F, Liu D, Zhu S, Du D-Z (2007) Connected dominating sets in wireless networks with different transmission range. IEEE Trans Mob Comput 6(7):721–730
25. Wan P-J, Alzoubi KM, Frieder O (2002) Distributed construction of connected dominating set in wireless ad hoc networks. In: IEEE INFOCOM, New York
26. Wu W, Du H, Jia X, Li Y, Huang C-H (2006) Minimum connected dominating sets and maximal independent sets in unit disk graphs. Theor Comput Sci 352:1–7

Connected Set-Cover and Group Steiner Tree

Lidong Wu[1], Huijuan Wang[2], and Weili Wu[3,4,5]
[1]Department of Computer Science, The University of Texas, Tyler, TX, USA
[2]Shandong University, Jinan, China
[3]College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi Province, China
[4]Department of Computer Science, California State University, Los Angeles, CA, USA
[5]Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Keywords

Approximation algorithms; Connected set-cover; Group Steiner tree; Performance ratio

Years and Authors of Summarized Original Work

2012; Zhang, Wu, Lee, Du
2012; Elbassion, Jelic, Matijevic
2013; Wu, Du, Wu, Li, Lv, Lee

Problem Definition

Given a collection C of subsets of a finite set X, find a minimum subcollection C′ of C such that every element of X appears in some subset in C′. This problem is called the *minimum set-cover* problem. Every feasible solution, i.e., a subcollection C′ satisfying the required condition, is called a *set-cover*. The minimum set-cover problem is NP-hard, and the complexity of approximation for it is well solved. It is well known that (1) the minimum set-cover problem has a polynomial-time $(1 + \ln n)$-approximation where $n = |X|$ [2, 7, 8], and moreover (2) if the minimum set-cover problem has a polynomial-time $(\rho \ln n)$-approximation for any $0 < \rho < 1$, then $NP \subseteq DTIME(n^{O(\log \log n)})$ [4].

The *minimum connected set-cover* problem is closely related to the minimum set-cover problem, which can be described as follows: Given a collection C of subsets of a finite set X and a graph G with vertex set C, find a minimum set-cover C′ $\subseteq$ C such that the subgraph induced by C′ is connected. An issue of whether the minimum connected set-cover problem has a polynomial-time $O(\log n)$-approximation or not [1, 9, 11] was open for several years.

Key Results

Zhang et al. [12] solved this problem by discovering a relationship between the minimum connected set-cover problem and the group Steiner tree problem.

Given a graph $G = (V, E)$ with edge nonnegative weight $c : E \to N$ and k subsets (called groups) of vertices, $V_1, \ldots, V_k$, find the minimum edge-weight tree interconnecting those k vertex subsets, i.e., containing at least one vertex from each subset. This is called the *group Steiner tree* problem. It has another formulation as follows: Given a graph $G = (V, E)$ with edge nonnegative weight $c : E \to R^+$, a special vertex r, and k subsets of vertices, $V_1, \ldots, V_k$, find the minimum edge-weight tree with root r, interconnecting those k vertex subsets.

These two formulations are equivalent in the sense that one has a polynomial-time ρ-approximation and so does the other one. Actually, consider vertex r as a group with only one member. Then it is immediately known that if the first formulation has a polynomial-time ρ-

approximation, so does the second formulation. Next, assume the second formulation has a polynomial-time ρ-approximation. In the first formulation, fix a group V_1, for each vertex $v \in V_1$, and apply the polynomial-time ρ-approximation algorithm for the second formulation to the root $r = v$ and $k - 1$ groups $V_2, \ldots, V_k$. Choose the shortest one from $|V_1|$ obtained trees, which would be a polynomial-time ρ-approximation for the first formulation.

The following are well-known results for the group Steiner tree problem

Theorem 1 (Halperin and Krauthgamer [6]) *The group Steiner tree problem has no polynomial-time $O(\log^{2-\varepsilon} n)$-approximation for any $\varepsilon > 0$ unless NP has quasi-polynomial Las-Vega algorithm.*

Theorem 2 (Garg, Konjevod, Ravi [5]) *The group Steiner tree problem has a polynomial-time random $O(\log^2 n \log k)$-approximation where n is the number of nodes in the input graph and k is the number of groups.*

Zhang et al. [12] showed that if the minimum connected set-cover problem has a polynomial-time ρ-approximation, then for any $\varepsilon > 0$, there is a polynomial-time $(\rho + \varepsilon)$-approximation for the group Steiner tree problem. Therefore, by Theorem 1 they obtained the following result.

Theorem 3 (Zhang et al. [12]) *The connected set-cover problem has no polynomial-time $O(\log^{2-\varepsilon} n)$-approximation for any $\varepsilon > 0$ unless NP has quasi-polynomial Las-Vega algorithm.*

To obtain a good approximation for the minimum connected set-cover problem, Wu et al. [10] showed that if the group Steiner tree problem has a polynomial-time ρ-approximation, so does the minimum connected set-cover problem. Therefore, they obtained the following theorem.

Theorem 4 (Wu et al. [10]) *The connected set-cover problem has a polynomial-time random $O(\log^2 n \log k)$-approximation where $n = |\mathrm{C}|$ and $k = |X|$.*

Combining what have been proved by Zhang et al. [12] and by Wu et al. [10], it is easy to know the following relation.

Theorem 5 *The connected set-cover problem has a polynomial-time $(\rho + \varepsilon)$-approximation for any $\varepsilon > 0$ if and only if the group Steiner tree problem has a polynomial-time $(\rho + \varepsilon)$-approximation.*

This equivalence is also independently discovered by [3]. Actually, this equivalence is similar to the one between the minimum set-cover problem and the *minimum hitting set* problem.

For each element $x \in X$, define a collection of subsets:

$$\mathrm{C}_x = \{S \mid x \in S \in \mathrm{C}\}.$$

Then, the minimum set-cover problem becomes the minimum hitting set problem as follows: Given a finite set C and a collection of subsets of C, $\{\mathrm{C}_x \mid x \in\}$, find the minimum hitting set, i.e., a subset C' of C such that for every $x \in X$, $\mathrm{C}' \cap \mathrm{C}_x \neq \emptyset$.

Similarly, the minimum connected set-cover problem becomes the equivalent *connected hitting set* problem as follows: Given a finite set C, a graph G with vertex set C, and a collection of subsets of C, $\{\mathrm{C}_x \mid x \in\}$, find the minimum connected hitting set where a connected hitting set is a hitting set C' such that C' induces a connected subgraph of G.

To see the equivalence between the minimum connected hitting set problem and the group Steiner tree problem, it is sufficient to note the following two facts.

First, the existence of a connected hitting set C' is equivalent to the existence of a tree with weight $|\mathrm{C}'| - 1$, interconnecting groups C_x for all $x \in X$ when the graph G is given unit weight for each edge. This is because in the subgraph induced by C', we can construct a spanning tree with weight $|\mathrm{C}'| - 1$.

Second, a graph with nonnegative integer edge weight can be turned into an equivalent graph with unit edge weight by adding some new vertices to cut each edge with weight bigger than one into several edges with unit weight.

Open Problems

It is an open problem whether there exists or not a polynomial-time approximation for the minimum connected set-cover problem with performance ratio $O(\log^{\alpha} n)$ for $2 < \alpha < 3$.

Cross-References

- ▸ Minimum Connected Sensor Cover
- ▸ Performance-Driven Clustering

Recommended Reading

1. Cerdeira JO, Pinto LS (2005) Requiring connectivity in the set covering problem. J Comb Optim 9: 35–47
2. Chvátal V (1979) A greedy heuristic for the set-covering problem. Math Oper Res 4(3): 233–235
3. Elbassion K, Jelic S, Matijevic D (2012) The relation of connected set cover and group Steiner tree. Theor Comput Sci 438:96–101
4. Feige U (1998) A threshold of $\ln n$ for approximating set-cover. J ACM 45(4):634–652
5. Garg N, Konjevod G, Ravi R (1998) A polylogarithmic approximation algorithm for the group Steiner tree problem. In: SODA, San Francisco
6. Halperin E, Krauthgamer R (2003) Polylogarithmic inapproximability. In: STOC, San Diego, pp 585–594
7. Johnson DS (1974) Approximation algorithms for combinatorial problems. J Comput Syst Sci 9(3):256–278
8. Lovász L (1975) On the ratio of optimal integral and fractional covers. Discret Math 13(4): 383–390
9. Shuai T-P, Hu X (2006) Connected set cover problem and its applications. In: Proceedings of the AAIM 2006, Hong Kong. LNCS, vol 4041, pp 243–254
10. Wu L, Du H, Wu W, Li D, Lv J, Lee W (2013) Approximations for minimum connected sensor cover. In: INFOCOM, Turin
11. Zhang Z, Gao X, Wu W (2009) Algorithms for connected set cover problem and fault-tolerant connected set cover problem. Theor Comput Sci 410: 812–817
12. Zhang W, Wu W, Lee W, Du D-Z (2012) Complexity and approximation of the connected set-cover problem. J Glob Optim 53(3):563–572

Connectivity and Fault Tolerance in Random Regular Graphs

Sotiris Nikoletseas
Computer Engineering and Informatics Department, University of Patras, Patras, Greece
Computer Technology Institute and Press "Diophantus", Patras, Greece

Keywords

Average case analysis of algorithms; Connectivity; Random graphs; Robustness

Years and Authors of Summarized Original Work

2000; Nikoletseas, Palem, Spirakis, Yung

Problem Definition

A new model of random graphs was introduced in [10], that of random regular graphs with edge faults (denoted hereafter by $G^r_{n,p}$), obtained by selecting the edges of a random member of the set of all regular graphs of degree r independently and with probability p. Such graphs can represent a communication network in which the links fail independently and with probability $f = 1 - p$. A formal definition of the probability space $G^r_{n,p}$ follows.

Definition 1 (The $G^r_{n,p}$ Probability Space) Let G^r_n be the probability space of all random regular graphs with n vertices where the degree of each vertex is r. The probability space $G^r_{n,p}$ of random regular graphs with edge faults is constructed by the following two subsequent random experiments: first, a random regular graph is chosen from the space G^r_n and, second, each edge is randomly and independently deleted from this graph with probability $f = 1 - p$.

Important connectivity properties of $G^r_{n,p}$ are investigated in this entry by estimating the ranges of r, f for which, with high probability, $G^r_{n,p}$

graphs (a) are highly connected (b) become disconnected and (c) admit a giant (i.e., of $\Theta(n)$ size) connected component of small diameter.

Notation. The terms "almost certainly" (a.c.) and "with high probability" (w.h.p.) will be frequently used with their standard meaning for random graph properties. A property defined in a random graph holds almost certainly when its probability tends to 1 as the independent variable (usually the number of vertices in the graph) tends to infinity. "With high probability" means that the probability of a property of the random graph (or the success probability of a randomized algorithm) is at least $1 - n^{-\alpha}$, where $\alpha > 0$ is a constant and n is the number of vertices in the graph.

The interested reader can further study [1] for an excellent exposition of the probabilistic method and its applications, [3] for a classic book on random graphs, as well as [6] for an excellent book on the design and analysis of randomized algorithms.

Key Results

Summary. This entry studies several important connectivity properties of random regular graphs with edge faults. In order to deal with the $G^r_{n,p}$ model, [10] first extends the notion of configurations and the translation lemma between configurations and random regular graphs provided by B. Bollobás [2, 3], by introducing the concept of *random configurations* to account for edge faults and by also providing an *extended translation lemma* between random configurations and random regular graphs with edge faults.

For this new model of random regular graphs with edge faults [10] shows that:

1. For all failure probabilities $f = 1 - p \leq n^{-\epsilon}$ ($\epsilon \geq \frac{3}{2r}$ fixed) and any $r \geq 3$ the biggest part of $G^r_{n,p}$ (i.e., the whole graph except of $O(1)$ vertices) remains connected and this connected part cannot be separated, almost certainly, unless more than r vertices are removed. Note interestingly that the situation for this range of f and r is very similar, despite the faults, to the properties of G^r_n which is r-connected for $r \geq 3$.
2. $G^r_{n,p}$ is *disconnected* a.c. for constant f and any $r = o(\log n)$ but is *highly connected*, almost certainly, when $r \geq \alpha \log n$, where $\alpha > 0$ an appropriate constant.
3. Even when $G^r_{n,p}$ becomes disconnected, it still has *a giant component of small diameter*, even when $r = O(1)$. An $O(n \log n)$-time algorithm to construct a giant component is provided.

Configurations and Translation Lemmata

Note that it is not as easy (from the technical point of view) as in the $G_{n,p}$ case to argue about random regular graphs, because of the stochastic dependencies on the existence of the edges due to regularity. The following notion of *configurations* was introduced by B. Bollobás [2, 3] to translate statements for random regular graphs to statements for the corresponding configurations which avoid the edge dependencies due to regularity and thus are much easier to deal with:

Definition 2 (Bollobás [2]) Let $w = \cup_{j=1}^{n} w_j$ be a fixed set of $2m = \sum_{j=1}^{n} d_j$ labeled vertices where $|w_j| = d_j$. A configuration F is a partition of w into m pairs of vertices, called edges of F.

Given a configuration F, let $\theta(F)$ be the (multi)graph with vertex set V in which (i, j) is an edge if and only if F has a pair (edge) with one element in w_i and the other in w_j. Note that every regular graph $G \in G^r_n$ is of the form $\theta(F)$ for exactly $(r!)^n$ configurations. However not every configuration F with $d_j = r$ for all j corresponds to a $G \in G^r_n$ since F may have an edge entirely in some w_j or parallel edges joining w_i and w_j.

Let ϕ be the set of all configurations F and let G^r_n be the set of all regular graphs. Given a property (set) $Q \subseteq G^r_n$ let $Q^* \subseteq \phi$ such that $Q^* \cap \theta^{-1}(G^r_n) = \theta^{-1}(Q)$. By estimating the probability of possible cycles of length one (self-loops) and two (loops) among pairs w_i, w_j in $\theta(F)$, the following important lemma follows:

Lemma 1 (Bollobás [3]) *If $r \geq 2$ is fixed and property Q^* holds for a.e. configuration, then property Q holds for a.e. $r-$regular graph.*

The main importance of the above lemma is that when studying random regular graphs, instead of considering the set of all random regular graphs, one can study the (much more easier to deal with) set of configurations.

In order to deal with edge failures, [10] introduces here the following extension of the notion of configurations:

Definition 3 (Random Configurations) Let $w = \cup_{j=1}^{n} w_j$ be a fixed set of $2m = \sum_{j=1}^{n} d_j$ labeled "vertices" where $|w_j| = d_j$. Let F be any configuration of the set ϕ. For each edge of F, remove it with probability $1 - p$, independently. Let $\hat{\phi}$ be the new set of objects and $\hat{F}$ the outcome of the experiment. $\hat{F}$ is called a *random configuration*.

By introducing probability p in every edge, an extension of the proof of Lemma 1 leads (since in both $\bar{Q}$ and $\hat{Q}$ each edge has the same probability and independence to be deleted, thus the modified spaces follow the properties of Q and Q^*) to the following extension to random configurations.

Lemma 2 (Extended Translation Lemma) *Let $r \geq 2$ fixed and $\bar{Q}$ be a property for $G_{n,p}^r$ graphs. If $\hat{Q}$ holds for a.e. random configuration, then the corresponding property $\bar{Q}$ holds for a.e. graph in $G_{n,p}^r$.*

Multiconnectivity Properties of $G_{n,p}^r$

The case of constant link failure probability f is studied, which represents a worst case for connectivity preservation. Still, [10] shows that logarithmic degrees suffice to guarantee that $G_{n,p}^r$ remains w.h.p. highly connected, despite these constant edge failures. More specifically:

Theorem 1 *Let G be an instance of $G_{n,p}^r$ where $p = \Theta(1)$ and $r \geq \alpha \log n$, where $\alpha > 0$ an appropriate constant. Then G is almost certainly k-connected, where*

$$k = O\left(\frac{\log n}{\log \log n}\right)$$

The proof of the above theorem uses Chernoff bounds to estimate the vertex degrees in $G_{n,p}^r$ and "similarity" of $G_{n,p}^r$ and $G_{n,p'}$ (whose properties are known) for a suitably chosen p'.

Now the (more practical) case in which $f = 1 - p = o(1)$ is considered and it is proved that the desired connectivity properties of random regular graphs are almost preserved despite the link failures. More specifically:

Theorem 2 *Let $r \geq 3$ and $f = 1 - p = O(n^{-\epsilon})$ for $\epsilon \geq \frac{3}{2r}$. Then the biggest part of $G_{n,p}^r$ (i.e., the whole graph except of $O(1)$ vertices) remains connected and this connected part (excluding the vertices that were originally neighbors of the $O(1)$-sized disconnected set) cannot be separated unless more than r vertices are removed, with probability tending to 1 as n tends to $+\infty$.*

The proof is carefully extending, in the case of faults, a known technique for random regular graphs about not admitting small separators.

$G_{n,p}^r$ Becomes Disconnected

Next remark that a constant link failure probability dramatically alters the connectivity structure of the regular graph in the case of low degrees. In particular, by using the notion of random configurations, [10] proves the following theorem:

Theorem 3 *When $2 \leq r \leq \frac{\sqrt{\log n}}{2}$ and $p = \Theta(1)$ then $G_{n,p}^r$ has at least one isolated node with probability at least $1 - n^{-k}, k \geq 2$.*

The regime for disconnection is in fact larger, since [10] shows that $G_{n,p}^r$ is a.c. disconnected even for any $r = o(\log n)$ and constant f. The proof of this last claim is complicated by the fact that due to the range for r one has to avoid using the extended translation lemma.

Existence of a Giant Component in $G_{n,p}^r$

Since $G_{n,p}^r$ is a.c. disconnected for $r = o(\log n)$ and $1 - p = f = \Theta(1)$, it would be interesting to know whether at least a large part

of the network represented by $G^r_{n,p}$ is still connected, i.e., whether the biggest connected component of $G^r_{n,p}$ is large. In particular, [10] shows that:

Theorem 4 *When $f < 1 - \frac{32}{r}$ then $G^r_{n,p}$ admits a giant (i.e., $\Theta(n)$-sized) connected component for any $r \geq 64$ with probability at least $1 - O\left(\frac{\log^2 n}{n^{\alpha/3}}\right)$, where $\alpha > 0$ a constant that can be selected.*

In fact, the proof of the existence of the component includes first proving the existence (w.h.p.) of a sufficiently long (of logarithmic size) path as a basis for a BFS process starting from the vertices of that path that creates the component. The proof is quite complex: occupancy arguments are used (bins correspond to the vertices of the graphs while balls correspond to its edges); however, the random variables involved are not independent, and in order to use Chernoff-Hoeffding bounds for concentration one must prove that these random variables, although not independent, are negatively associated. Furthermore, the evaluation of the success of the BFS process uses a careful, detailed average case analysis.

The path construction and the BFS process can be viewed as an algorithm that (in case of no failures) actually reveals a giant connected component. This algorithm is very efficient, as shown by the following result:

Theorem 5 *A giant component of $G^r_{n,p}$ can be constructed in $O(n \log n)$ time, with probability at least $1 - O\left(\frac{\log^2 n}{n^{\alpha/3}}\right)$, where $\alpha > 0$ a constant that can be selected.*

Applications

In recent years the development and use of distributed systems and communication networks has increased dramatically. In addition, state-of-the-art multiprocessor architectures compute over structured, regular interconnection networks. In such environments, several applications may share the same network while executing concurrently. This may lead to unavailability of certain network resources (e.g., links) for certain applications. Similarly, faults may cause unavailability of links or nodes. The aspect of *reliable distributed computing* (which means computing with the available resources and resisting faults) adds value to applications developed in such environments.

When computing in the presence of faults, one cannot assume that the actual structure of the computing environment is known. Faults may happen even in execution time. In addition, what is a "faulty" or "unavailable" link for one application may in fact be the de-allocation of that link because it is assigned (e.g., by the network operation system) to another application. The problem of analyzing allocated computation or communication in a network over a *randomly assigned subnetwork* and *in the presence of faults* has a nature different from fault analysis of special, well-structured networks (e.g., hypercube), which does not deal with network aspects. The work presented in this entry addresses this interesting issue, i.e., analyzing the average case taken over a set of possible topologies and focuses on multiconnectivity and existence of giant component properties required for reliable distributed computing in such randomly allocated unreliable environments.

The following important application of this work should be noted: multitasking in distributed memory multiprocessors is usually performed by assigning an arbitrary subnetwork (of the interconnection network) to each task (called the *computation graph*). Each parallel program may then be expressed as communicating processors over the computation graph. Note that a multiconnectivity value k of the computation graph means also that the execution of the application can tolerate up to $k - 1$ *online additional faults*.

Open Problems

The ideas presented in [10] inspired already further interesting research. Andreas Goerdt [4] continued the work presented in a preliminary version [8] of [10] and showed the following results: if the degree r is fixed then $p = \frac{1}{r-1}$ is a

threshold probability for the existence of a linear-sized component in the faulty version of almost all random regular graphs. In fact, he further shows that if each edge of an *arbitrary* graph G with maximum degree bounded above by r is present with probability $p = \frac{\lambda}{r-1}$, when $\lambda < 1$, then the faulty version of G has only components whose size is at most logarithmic in the number of nodes, with high probability. His result implies some kind of optimality of random regular graphs with edge faults. Furthermore, [5, 7] investigates important expansion properties of random regular graphs with edge faults, as well as [9] does in the case of fat trees, a common type of interconnection networks. It would be also interesting to further pursue this line of research, by also investigating other combinatorial properties (and also provide efficient algorithms) for random regular graphs with edge faults.

Recommended Reading

1. Alon N, Spencer J (1992) The probabilistic method. Wiley, New York
2. Bollobás B (1980) A probabilistic proof of an asymptotic formula for the number of labeled regular graphs. Eur J Comb 1:311–316
3. Bollobás B (1985) Random graphs. Academic, London
4. Goerdt A (1997) The giant component threshold for random regular graphs with edge faults. In: Proceedings of mathematical foundations of computer science (MFCS'97), Bratislava, pp 279–288
5. Goerdt A (2001) Random regular graphs with edge faults: expansion through cores. Theor Comput Sci (TCS) J 264(1):91–125
6. Motwani R, Raghavan P (1995) Randomized algorithms. Cambridge University Press, Cambridge
7. Nikoletseas S, Spirakis P (1995) Expander properties in random regular graphs with edge faults. In: 12th annual symposium on theoretical aspects of computer science (STACS), München, pp 421–432
8. Nikoletseas S, Palem K, Spirakis P, Yung M (1994) Short vertex disjoint paths and multiconnectivity in random graphs: reliable network computing. In: 21st international colloquium on automata, languages and programming (ICALP), Jerusalem, pp 508–515
9. Nikoletseas S, Pantziou G, Psycharis P, Spirakis P (1997) On the reliability of fat-trees. In: 3rd international European conference on parallel processing (Euro-Par), Passau, pp 208–217
10. Nikoletseas S, Palem K, Spirakis P, Yung M (2000) Connectivity properties in random regular graphs with edge faults. Spec Issue Random Comput Int J Found Comput Sci (IJFCS) 11(2):247–262. World Scientific Publishing Company

Consensus with Partial Synchrony

Bernadette Charron-Bost[1] and André Schiper[2]
[1]Laboratory for Informatics, The Polytechnic School, Palaiseau, France
[2]EPFL, Lausanne, Switzerland

Keywords

Agreement problem

Years and Authors of Summarized Original Work

1988; Dwork, Lynch, Stockmeyer

Problem Definition

Reaching agreement is one of the central issues in fault tolerant distributed computing. One version of this problem, called *Consensus*, is defined over a fixed set $\Pi = \{p_1, \ldots, p_n\}$ of n processes that communicate by exchanging messages along channels. Messages are correctly transmitted (no duplication, no corruption), but some of them may be lost. Processes may fail by prematurely stopping (crash), may omit to send or receive some messages (omission), or may compute erroneous values (Byzantine faults). Such processes are said to be *faulty*. Every process $p \in \Pi$ has an initial value v_p and non-faulty processes must decide irrevocably on a common value v. Moreover, if the initial values are all equal to the same value v, then the common decision value is v. The properties that define Consensus can be split into safety properties (processes decide on the same value; the decision value must be consistent with initial values) and a liveness property (processes must eventually decide).

Various Consensus algorithms have been described [6, 12] to cope with any type of process

failures if there is a known (Intuitively, "known bound" means that the bound can be "built into" the algorithm. A formal definition is given in the next section.) bound on the transmission delay of messages (*communication is synchronous*) and a known bound on process relative speeds (*processes are synchronous*). In completely asynchronous systems, where there exists no bound on transmission delays and no bound on process relative speeds, Fischer, Lynch, and Paterson [8] have proved that there is no Consensus algorithm resilient to even one crash failure. The paper by Dwork, Lynch, and Stockmeyer [7] introduces the concept of *partial synchrony*, in the sense it lies between the completely synchronous and completely asynchronous cases, and shows that partial synchrony makes it possible to solve Consensus in the presence of process failures, whatever the type of failure is.

For this purpose, the paper examines the quite realistic case of asynchronous systems that behave synchronously during some "good" periods of time. Consensus algorithms designed for synchronous systems do not work in such systems since they may violate the safety properties of Consensus during a bad period, that is when the system behaves asynchronously. This leads to the following question: is it possible to design a Consensus algorithm that never violates safety conditions in an asynchronous system, while ensuring the liveness condition when some additional conditions are met?

Key Results

The paper has been the first to provide a positive and comprehensive answer to the above question. More precisely, the paper (1) defines various types of partial synchrony and introduces a new round based computational model for partially synchronous systems, (2) gives various Consensus algorithms according to the severity of failures (crash, omission, Byzantine faults with or without authentication), and (3) shows how to implement the round based computational model in each type of partial synchrony.

Partial Synchrony

Partial synchrony applies both to communications and to processes. Two definitions for *partially synchronous communications* are given: (1) for each run, there exists an upper bound Δ on communication delays, but Δ is *unknown* in the sense it depends on the run; (2) there exists an upper bound Δ on communication delays that is common for all runs (Δ is *known*), but holds only after some time T, called the *Global Stabilization Time* (*GST*) that may depend on the run (*GST* is *unknown*). Similarly, *partially synchronous processes* are defined by replacing "transmission delay of messages" by "relative process speeds" in (1) and (2) above. That is, the upper bound on relative process speed Φ is unknown, or Φ is known but holds only after some unknown time.

Basic Round Model

The paper considers a round based model: computation is divided into *rounds* of message exchange. Each round consists of a *send step*, a *receive step*, and then a *computation step*. In a send step, each process sends messages to any subset of processes. In a receive step, some subset of the messages sent to the process during the send step at the *same* round is received. In a computation step, each process executes a state transition based on its current state and the set of messages just received.

Some of the messages that are sent may not be received, i.e., some can be lost. However, the *basic round model* assumes that there is some round GSR, such that all messages sent from non faulty processes to non faulty processes at round GSR or afterward are received.

Consensus Algorithm for Benign Faults (Requires $f < n/2$)

In the paper, the algorithm is only described informally (textual form). A formal expression is given by Algorithm 1: the code of each process is given round by round, and each round is specified by the send and the computation steps (the receive step is implicit). The constant f denotes the maximum number of processes that may be faulty (crash or omission). The algorithm requires $f < n/2$.

Algorithm 1 Consensus algorithm in the basic round model for benign faults ($f < n/2$)

```
 1: Initialization:
 2:     Acceptable_p := {v_p}                                   {v_p is the initial value of p}
 3:     Proper_p := {v_p}          {All the lines for maintaining Proper_p are trivial to write, and so are omitted}
 4:     vote_p := ⊥
 5:     Lock_p := ∅

 6: Round r = 4k − 3:
 7:     Send:
 8:         send ⟨Acceptable_p⟩ to coord_k

 9:     Compute:
10:         if p = coord_k and p receives at least ≥ n − f messages containing a common value then
11:             vote_p := select one of these common acceptable values

12: Round r = 4k − 2:
13:     Send:
14:         if p = coord_k and vote_p ≠ ⊥ then
15:             send ⟨vote_p⟩ to all processes

16:     Compute:
17:         if received ⟨v⟩ from coord_k then
18:             Lock_p := Lock_p ∖ {v, −}; Lock_p := Lock_p ∪ {(v, k)};

19: Round r = 4k − 1:
20:     Send:
21:         if ∃v s.t. (v, k) ∈ Lock_p then
22:             send ⟨ack⟩ to coord_k

23:     Compute:
24:         if p = coord_k then
25:             if received at least ≥ f + 1 ack messages then
26:                 DECIDE(vote_p);
27:             vote_p := ⊥

28: Round r = 4k:
29:     Send:
30:         send ⟨Lock_p⟩ to all processes

31:     Compute:
32:         for all (v, θ) ∈ Lock_p do
33:             if received (w, θ̄) s.t. w ≠ v and θ̄ ≥ θ then                {release lock on v}
34:                 Lock_p := Lock_p ∪ {(w, θ̄)} ∖ {(v, θ)};
35:         if |Lock_p| = 1 then
36:             Acceptable_p := v where (v, −) ∈ Lock_p
37:         else
38:             if Lock_p = ∅ then Acceptable_p := Proper_p else Acceptable_p := ∅
```

Rounds are grouped into phases, where each phase consists in four consecutive rounds. The algorithm includes the rotating coordinator strategy: each phase k is led by a unique coordinator – denoted by $coord_k$ – defined as process p_i for phase $k = i\,(mod\ n)$. Each process p maintains a set $Proper_p$ of values that p has heard of (*proper* values), initialized to $\{v_p\}$ where v_p is p's initial value. Process p attaches $Proper_p$ to each message it sends.

Process p may *lock* value v when p thinks that some process might decide v. Thus value v is an *acceptable* value to p if (1) v is a proper value to p, and (2) p does not have a lock on any value except possibly v (lines 35–38).

At the first round of phase k (round $4k-3$), each process sends the list of its acceptable values to $coord_k$. If $coord_k$ receives at least $n-f$ sets of acceptable values that all contain some value v, then $coord_k$ votes for v (line 11), and sends its vote to all at second round $4k-2$. Upon receiving a vote for v, any process locks v in the current phase (line 18), releases any earlier lock on v, and sends an acknowledgment to $coord_k$ at the next round $4k-1$. If the latter process receives acknowledgments from at least $f+1$ processes, then it decides (line 26). Finally locks are released at round $4k$ – for any value v, only the lock from the most recent phase is kept, see line 34 – and the set of values *acceptable* to p is updated (lines 35–38).

Consensus Algorithm for Byzantine Faults (Requires $f < n/3$)

Two algorithms for Byzantine faults are given. The first algorithm assumes *signed messages*, which means that any process can verify the origin of all messages. This fault model is

called *Byzantine faults with authentication*. The algorithm has the same phase structure as Algorithm 1. The difference is that (1) messages are signed, and (2) "proofs" are carried by some messages. A proof carried by message m sent by some process p_i in phase k consists of a set of signed messages $sgn_j(m', k)$, proving that p_i received message (m', k) in phase k from p_j before sending m. A proof is carried by the message send at line 16 and line 30 (Algorithm 1). Any process receiving a message carrying a proof accepts the message and behaves accordingly if – and only if the proof is found valid. The algorithm requires $f < n/3$ (less than a third of the processes are faulty).

The second algorithm does not assume a mechanism for signing messages. Compared to Algorithm 1, the structure of a phase is slightly changed. The problem is related to the vote sent by the coordinator (line 15). Can a Byzantine coordinator fool other processes by not sending the right vote? With signed messages, such a behavior can be detected thanks to the "proofs" carried by messages. A different mechanism is needed in the absence of signature.

The mechanism is a small variation of the *Consistent Broadcast* primitive introduced by Srikanth and Toueg [15]. The broadcast primitive ensures that (1) if a non faulty process broadcasts m, then every non faulty process delivers m, and (2) if some non faulty process delivers m, then all non faulty processes also eventually deliver m. The implementation of this broadcast primitive requires two rounds, which define a *superround*. A phase of the algorithm consists now of three superrounds. The superrounds $3k-2$, $3k-1$, $3k$ mimic rounds $4k-3$, $4k-2$, and $4k-1$ of Algorithm 1, respectively. Lock-release of phase k occurs at the end of superround $3k$, i.e., does not require an additional round, as it does in the two previous algorithms. The algorithm also requires $f < n/3$.

The Special Case of Synchronous Communication

By strengthening the round based computational model, the authors show that synchronous communication allow higher resiliency. More precisely, the paper introduces the model called the *basic round model with signals*, in which upon receiving a signal at round r, every process knows that all the non faulty processes have received the messages that it has sent during round r. At each round after *GSR*, each non faulty process is guaranteed to receive a signal. In this computational model, the authors present three new algorithms tolerating less than n benign faults, $n/2$ Byzantine faults with authentication, and $n/3$ Byzantine faults respectively.

Implementation of the Basic Round Model

The last part of the paper consists of algorithms that simulate the basic round model under various synchrony assumption, for crash faults and Byzantine faults: first with partially synchronous communication and synchronous processes (case 1), second with partially synchronous communication and processes (case 2), and finally with partially synchronous processes and synchronous communication (case 3).

In case 1, the paper first assumes the basic case $\Phi = 1$, i.e., all non faulty process progress exactly at the same speed, which means that they have a common notion of time. Simulating the basic round model is simple in this case. In case 2 processes do not have a common notion of time. The authors handle this case by designing an algorithm for clock synchronization. Then each process uses its private clock to determine its current round. So processes alternate between steps of the clock synchronization algorithm and steps simulating rounds of the basic round model. With synchronous communication (case 3), the authors show that for any type of faults, the so-called basic round model with signals is implementable.

Note that, from the very definition of partial synchrony, the six algorithms share the fundamental property of tolerating message losses, provided they occur during a finite period of time.

Upper Bound for Resiliency

In parallel, the authors exhibit upper bounds for the resiliency degree of Consensus algorithms in each partially synchronous model, according to the type of faults. They show that their Consensus

Consensus with Partial Synchrony, Table 1 Tight resiliency upper bounds (P stands for "process", C for "communication"; 0 means "asynchronous", 1/2 means "partially synchronous", and 1 means "synchronous")

	$P = 0 \quad C = 0$	$P = 1/2 \quad C = 1/2$	$P = 1 \quad C = 1/2$	$P = 1/2 \quad C = 1$	$P = 1 \quad C = 1$
Benign	0	$\lceil (n-1)/2 \rceil$	$\lceil (n-1)/2 \rceil$	$n-1$	$n-1$
Authenticated Byzantine	0	$\lceil (n-1)/3 \rceil$	$\lceil (n-1)/3 \rceil$	$\lceil (n-1)/2 \rceil$	$n-1$
Byzantine	0	$\lceil (n-1)/3 \rceil$	$\lceil (n-1)/3 \rceil$	$\lceil (n-1)/3 \rceil$	$\lceil (n-1)/3 \rceil$

algorithms achieve these upper bounds, and so are optimal with respect to their resiliency degree. These results are summarized in Table 1.

Applications

Availability is one of the key features of critical systems, and is defined as the ratio of the time the system is operational over the total elapsed time. Availability of a system can be increased by replicating its critical components. Two main classes of replication techniques have been considered: *active* replication and *passive* replication. The Consensus problem is at the heart of the implementation of these replication techniques. For example, active replication, also called *state machine replication* [10, 14], can be implemented using the group communication primitive called *Atomic Broadcast*, which can be reduced to Consensus [3].

Agreement needs also to be reached in the context of distributed transactions. Indeed, all participants of a distributed transaction need to agree on the output *commit* or *abort* of the transaction. This agreement problem, called *Atomic Commitment*, differs from Consensus in the validity property that connects decision values (*commit* or *abort*) to the initial values (favorable to commit, or demanding abort) [9]. In the case decisions are required in all executions, the problem can be reduced to Consensus if the abort decision is acceptable although all processes were favorable to commit, in some restricted failure cases.

Open Problems

A slight modification to each of the algorithms given in the paper is to force a process repeatedly to broadcast the message "Decide v" after it decides v. Then the resulting algorithms share the property that all non faulty processes definitely make a decision within $O(f)$ rounds after GSR, and the constant factor varies between 4 (benign faults) and 12 (Byzantine faults). A question raised by the authors at the end of the paper is whether this constant can be reduced. Interestingly, a positive answer has been given later, in the case of benign faults and $f < n/3$, with a constant factor of 2 instead of 4. This can be achieved with deterministic algorithms, see [4], based on the communication schema of the Rabin randomized Consensus algorithm [13].

The second problem left open is the generalization of this algorithmic approach – namely, the design of algorithms that are always safe and that terminate when a sufficiently long good period occurs – to other fault tolerant distributed problems in partially synchronous systems. The latter point has been addressed for the Atomic Commitment and Atomic Broadcast problems (see section "Applications").

Cross-References

- Asynchronous Consensus Impossibility
- Failure Detectors
- Randomization in Distributed Computing

Recommended Reading

1. Bar-Noy A, Dolev D, Dwork C, Strong HR (1987) Shifting gears: changing algorithms on the fly To expedite Byzantine agreement. In: PODC, pp 42–51
2. Chandra TD, Hadzilacos V, Toueg S (1996) The weakest failure detector for solving consensus. J ACM 43(4):685–722
3. Chandra TD, Toueg S (1996) Unreliable failure detectors for reliable distributed systems. J ACM 43(2):225–267
4. Charron-Bost B, Schiper A (2007) The "Heard-Of" model: computing in distributed systems with benign failures. Technical report, EPFL

5. Dolev D, Dwork C, Stockmeyer L (1987) On the minimal synchrony needed for distributed consensus. J ACM 34(1):77–97
6. Dolev D, Strong HR (1983) Authenticated algorithms for Byzantine agreement. SIAM J Comput 12(4):656–666
7. Dwork C, Lynch N, Stockmeyer L (1988) Consensus in the presence of partial synchrony. J ACM 35(2):288–323
8. Fischer M, Lynch N, Paterson M (1985) Impossibility of distributed consensus with one faulty process. J ACM 32:374–382
9. Gray J (1990) A comparison of the Byzantine agreement problem and the transaction commit problem. In: Fault-tolerant distributed computing [Asilomar-Workshop 1986], vol 448, LNCS. Springer, Berlin, pp 10–17
10. Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21(7):558–565
11. Lamport L (1998) The part-time parliament. ACM Trans Comput Syst 16(2):133–169
12. Pease MC, Shostak RE, Lamport L (1980) Reaching agreement in the presence of faults. J ACM 27(2):228–234
13. Rabin M (1983) Randomized Byzantine generals. In: Proceedings of the 24th annual ACM symposium on foundations of computer science, pp 403–409
14. Schneider FB (1993) Replication management using the state-machine approach. In: Mullender S (ed) Distributed systems. ACM, pp 169–197
15. Srikanth TK, Toueg S (1987) Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distrib Comput 2(2):80–94

Convex Graph Drawing

Md. Saidur Rahman
Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh

Keywords

Convex drawing; Linear-time algorithm; Planar graphs

Years and Authors of Summarized Original Work

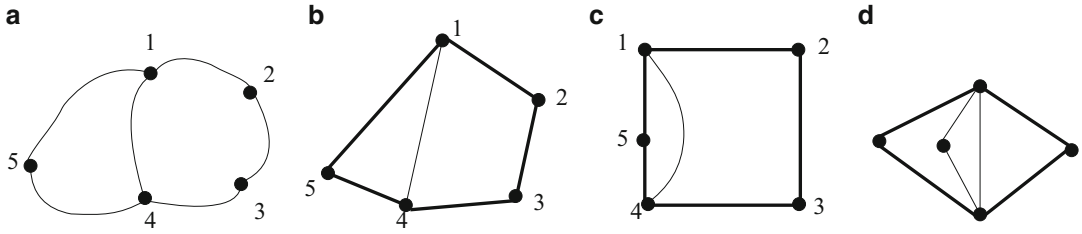
1984; Chiba, Yamanouchi, Nishizeki

Problem Definition

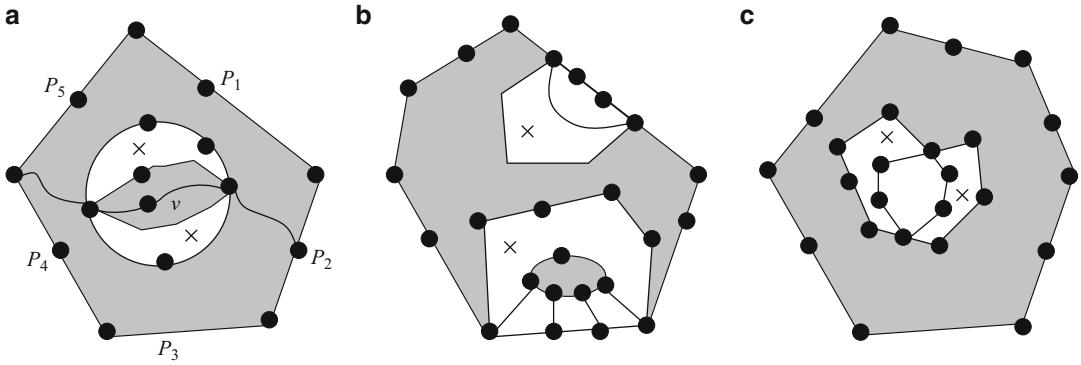
A convex drawing of a planar graph G is a planar drawing of G where every vertex is drawn as a point, every edge is drawn as a straight line segment, and every face is drawn as a convex polygon. Not every planar graph has a convex drawing. The planar graph in Fig. 1a has a convex drawing as shown in Fig. 1b whereas the planar graph in Fig. 1d has no convex drawing. Tutte [11] showed that every 3-connected planar graph has a convex drawing, and obtained a necessary and sufficient condition for a planar graph to have a convex drawing with a prescribed outer polygon. Furthermore, he gave a "barycentric mapping" method for finding a convex drawing, which requires solving a system of $O(n)$ linear equations [12] and leads to an $O(n^{1.5})$ time convex drawing algorithm for a planar graph with a fixed embedding. Development of faster algorithms for determining whether a planar graph (where the embedding is not fixed) has a convex drawing and finding such a drawing if it exits is addressed in the paper of Chiba, Yamanouchi, and Nishizeki [2].

A Characterization for Convex Drawing A plane graph is a planar graph with a fixed embedding. In a convex drawing of a plane graph G, the outer cycle $C_o(G)$ is also drawn as a convex polygon. The polygonal drawing C_o^* of $C_o(G)$, called an *outer convex polygon*, plays a crucial role in finding a convex drawing of G. The plane graph G in Fig. 1a admits a convex drawing if an outer convex polygon C_o^* has all vertices 1, 2, 3, 4, and 5 of $C_o(G)$ as the *apices* (i.e., geometric vertices) of C_o^*, as illustrated in Fig. 1b. However, if C_o^* has only apices 1, 2, 3, and 4, then G does not admit a convex drawing as depicted in Fig. 1c. We say that an outer convex polygon C_o^* is *extendible* if there exists a convex drawing of G in which $C_o(G)$ is drawn as C_o^*. Thus, the outer convex polygon drawn by thick lines in Fig. 1b is extendible, while that in Fig. 1c is not. If the outer facial cycle C_o has an extendible outer convex polygon, we say that the facial cycle C_o is *extendible.*

Tutte established a necessary and sufficient condition for an outer convex polygon to be



Convex Graph Drawing, Fig. 1 Plane graphs and drawings



Convex Graph Drawing, Fig. 2 G and C_o^* violating Conditions (i)–(iii) in Theorem 1

extendible [11]. The following theorem obtained by Thomassen [9] is slightly more general than the result of Tutte.

Theorem 1 *Let G be a 2-connected plane graph, and let C_o^* be an outer convex polygon of G. Let C_o^* be a k-gon, $k \geq 3$, and let $P_1, P_2, \ldots, P_k$ be the paths in $C_o(G)$, each corresponding to a side of the polygon C_o^*, as illustrated in Fig. 2a. Then, C_o^* is extendible if and only if the following Conditions (i)–(iii) hold.*

(i) For each inner vertex v with $d(v) \geq 3$, there exist three paths disjoint except v, each joining v and an outer vertex.
(ii) $G - V(C_o(G))$ has no connected component H such that all the outer vertices adjacent to vertices in H lie on a single path P_i, and no two outer vertices in each path P_i are joined by an inner edge.
(iii) Any cycle containing no outer edge has at least three vertices of degree ≥ 3.

Figure 2a–c violate Conditions (i)–(iii) of Theorem 1, respectively, where each of the faces marked by $\times$ cannot be drawn as a convex polygon.

Key Results

Two linear algorithms for convex drawings are the key contribution of the paper of Chiba, Yamanouchi, and Nishizeki [2]. One algorithm is for finding a convex drawing of a plane graph if it exists, and the other algorithm is for testing whether there is a planar embedding of a given planar graph which has a convex drawing. Thus, the main result of the paper can be stated as in the following theorem.

Theorem 2 *Let G be a 2-connected planar graph. Then, one can determine whether G has a convex drawing in linear time and find such a drawing in linear time if it exists.*

Convex Drawing Algorithm

In this section, we describe the drawing algorithm of Chiba, Yamanouchi, and Nishizeki [2] which is

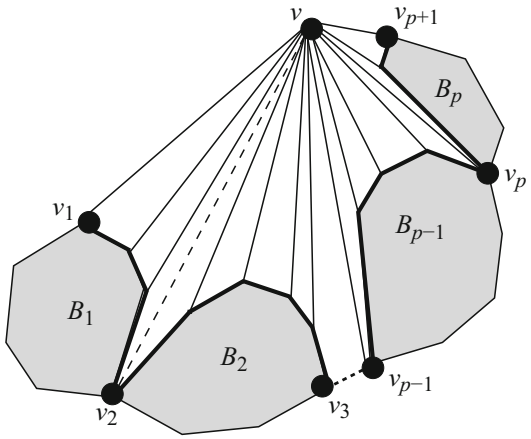
based on Thomassen's short proof of Theorem 1. Suppose that a 2-connected plane graph G and an outer convex polygon C_o^* satisfy conditions in Theorem 1. The convex drawing algorithm extends C_o^* into a convex drawing of G in linear time. For simplicity, it is assumed that every inner vertex has degree three or more in G. Otherwise, replace each maximal induced path not on $C_o(G)$ by a single edge joining its ends (the resulting simple graph G' satisfies Conditions (i)–(iii) of Theorem 1); then, find a convex drawing of G'; and finally, subdivide each edge substituting a maximal induced path.

They reduce the convex drawing of G to those of several subgraphs of G as follows: delete from G an arbitrary apex v of the outer convex polygon C_o^* together with the edges incident to v; divide the resulting graph $G' = G - v$ into blocks $B_1, B_2, \ldots, B_p$, $p \geq 1$ as illustrated in Fig. 3; determine an outer convex polygon C_i^* of each block B_i so that B_i with C_i^* satisfies Conditions (i)–(iii) of Theorem 1; and recursively apply the algorithm to each block B_i with C_i^* to determine the position of inner vertices of B_i.

The recursive algorithm described above can be implemented in linear time by ensuring that only the edges which newly appear on the outer face are traversed in each recursive step.

Convex Testing Algorithm

In this section, we describe the convex testing algorithm of Chiba, Yamanouchi, and Nishizeki [2] which implies a constructive proof of Theorem 2. They have modified the conditions in Theorem 1 into a form suitable for the convex testing, which is represented in terms of 3-connected components. Using the form, they have shown that the convex testing of a planar graph G can be reduced to the planarity testing of a certain graph obtained from G.



Convex Graph Drawing, Fig. 3 Reduction of the convex drawing of G into subproblems

To describe the convex testing algorithm, we need some definitions. A pair $\{x, y\}$ of vertices of a 2-connected graph $G = (V, E)$ is called a *separation pair* if there exists two subgraphs $G_1' = (V_1, E_1')$ and $G_2' = (V_2, E_2')$ satisfying the following conditions (a) and (b): (a) $V = V_1 \cup V_2, V_1 \cap V_2 = \{x, y\}$; and (b) $E = E_1' \cup E_2', E_1' \cap E_2' = \emptyset, |E_1'| \geq 2, |E_2'| \geq 2$. For a separation pair $\{x, y\}$ of G, $G_1 = (V_1, E_1' + (x, y))$ and $G_2 = (V_2, E_2' + (x, y))$ are called the *split graphs* of G. The new edges (x, y) added to G_1 and G_2 are called the *virtual edges*. Dividing a graph G into two split graphs G_1 and G_2 is called *splitting*. Reassembling the two split graphs G_1 and G_2 into G is called *merging*. Suppose that a graph G is split, the split graphs are split, and so on, until no more splits are possible. The graphs constructed in this way are called the *split components* of G. The split components are of three types: triple bonds (i.e., a set of three multiple edges), triangles, and 3-connected graphs. The *3-connected components* of G are obtained from the split components of G by merging triple bonds into a bond and triangles into a ring, as far as possible, where a *bond* is a set of multiple edges and a *ring* is a cycle. Note that the split components of G are not necessarily unique, but the 3-connected components of G are unique [5].

A separation pair $\{x, y\}$ is *prime* if x and y are the end vertices of a virtual edge contained in a 3-connected component. Suppose that $\{x, y\}$ is a prime separation pair of a graph G and that G is split at $\{x, y\}$, the split graphs are split, and so on, until no more splits are possible at $\{x, y\}$. A graph constructed in this way is called an $\{x, y\}$-*split component* of G if it has at least one real (i.e., non-virtual) edge.

In some cases, it can be easily known only from the $\{x, y\}$-split components for a single

separation pair $\{x, y\}$ that a graph G has no convex drawing. A prime separation pair $\{x, y\}$ of G is called a *forbidden separation pair* if there are either (a) at least four $\{x, y\}$-split components or (b) exactly three $\{x, y\}$-split components, each of which is neither a ring nor a bond. Note that an $\{x, y\}$-split component corresponds to an edge (x, y) if it is a bond and to a subdivision of an edge (x, y) if it is a ring. One can easily know that if a planar graph G has a forbidden separation pair, then any plane embedding of G has no convex drawing, that is, G has no extendible facial cycle. On the other hand, the converse of the fact above is not true. A prime separation pair $\{x, y\}$ is called a *critical separation pair* if there are either (i) exactly three $\{x, y\}$-split components including a bond or a ring or (ii) exactly two $\{x, y\}$-split components each of which is neither a bond nor a ring. When a planar graph G has no forbidden separation pair, two cases occur: if G has no critical separation pair either, then G is a subdivision of a 3-connected graph, and so every facial cycle of G is extendible; otherwise, that is, if G has critical separation pairs, then a facial cycle F of G may or may not be extendible, depending on the interaction of F and critical separation pairs.

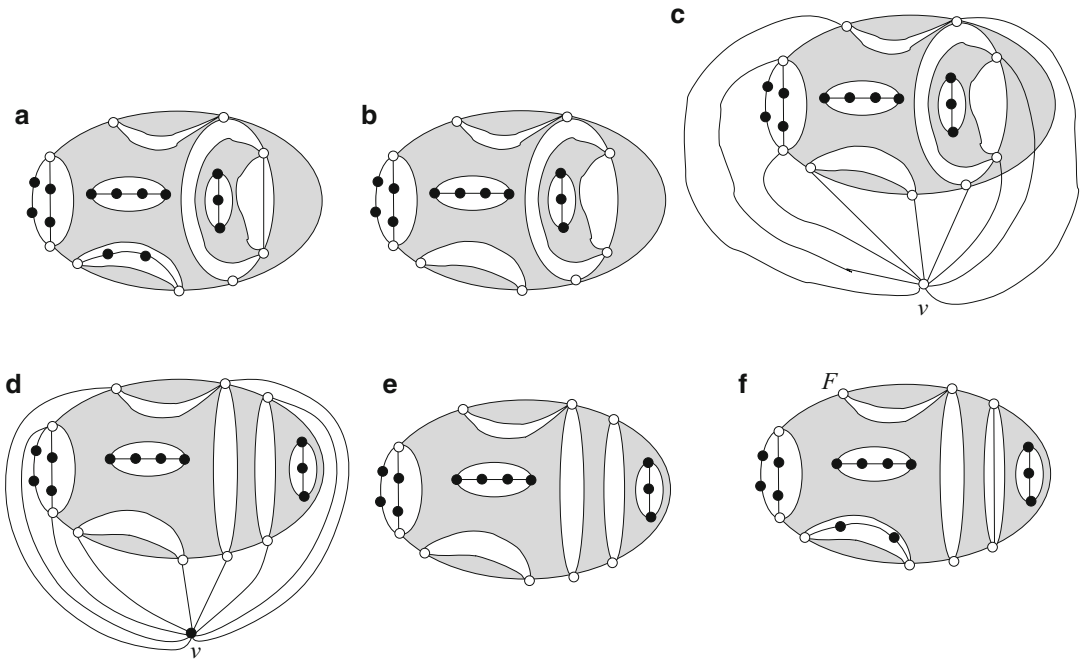
Using the concepts of forbidden separation pairs and critical separation pairs, Chiba et al. gave the following condition in Theorem 3 which is suitable for the testing algorithm. They proved that the condition in Theorem 3 is equivalent to the condition in Theorem 1 under a restriction that the outer convex polygon C_o^* is strict, that is, every vertex of $C_o(G)$ is an apex of C_o^* [2].

Theorem 3 *Let $G = (V, E)$ be a 2-connected plane graph with the outer facial cycle $F = C_o(G)$, and let C_o^* be an outer strict convex polygon of G. Then, C_o^* is extendible if and only if G and F satisfy the following conditions.*

(a) G has no forbidden separation pair.
(b) For each critical separation pair $\{x, y\}$ of G, there is at most one $\{x, y\}$-split component having no edge of F, and, if any, it is either a bond if $(x, y) \in E$ or a ring, otherwise.

The convex testing condition in Theorem 3 is given for a plane graph. Note that Condition (a) does not depend on a plane embedding. Thus, to test whether a planar graph G has a convex drawing, it is needed to test whether G satisfies Condition (a) or not and if G satisfies Condition (a) then test whether G has a plane embedding such that its outer face F satisfies Condition (b) in Theorem 3. With some simple observation, it is shown that every graph G having no forbidden separation pair has an embedding such that the outer face satisfies Condition (b) if G has at most one critical separation pair. Hence, every planar graph with no forbidden separation pair and at most one critical separation pair has a convex drawing.

The convex testing problem of G for the case where G has no forbidden separation pair and has two or more critical separation pairs is reduced to the planarity testing problem of a certain graph obtained from G. If G has a plane embedding which has a convex drawing, the outer face F of the embedding must satisfy Condition (b) of Theorem 3. Then, F contains every vertex of critical separation pairs and any split component which is neither a bond nor a ring must have an edge on the outer face. Observe that if a critical separation pair $\{x, y\}$ has exactly three $\{x, y\}$-split components, then two of them can have edges on F and one cannot have an edge on F; the $\{x, y\}$-split component which will not have an edge on F must be either a bond or a ring. Thus, to test whether G has such an embedding or not, a new graph from G is constructed as follows. For each critical separation pair $\{x, y\}$, if (x, y) is an edge of G, then delete the edge $\{x, y\}$ from G. If (x, y) is not an edge of G and exactly one $\{x, y\}$-split component is a ring, then delete the x-y path in the component from G. Let G_1 be the resulting graph, as illustrated in Fig. 4b. Let G_2 be the graph obtained from G_1 by adding a new vertex v and joining v to all vertices of critical separation pairs of G, as illustrated in Fig. 4c. If G_2 has a planar embedding Γ_2 such that v is embedded on the outer face of Γ_2 as illustrated in Fig. 4d, we get a planar embedding Γ_1 of G_1 from Γ_2 by deleting v from the embedding as illustrated in Fig. 4e. The outer facial cycle of Γ_1 will be the outer facial cycle F of a planar embedding of G as illustrated in Fig. 4f which satisfies the Condition (b) of Theorem 3. Thus,



Convex Graph Drawing, Fig. 4 Illustration for convex testing; (**a**) G, (**b**) G_1, (**c**) G_2, (**d**) Γ_2, (**e**) Γ_1, and (**f**) Γ

the strict convex polygon F^* of F is extendible. Hence, G has a convex drawing if G_2 has a planar embedding with v on the outer face. It is not difficult to show the converse implication. Hence, Theorem 2 holds.

Observe that F may not be the only extendible facial cycle of G, that is, Γ may not be the only planar embedding of G which has a convex drawing. Chiba et al. [2] also gave a linear algorithm which finds all extendible facial cycles of G.

Applications

Thomassen [10] showed applications of convex representations in proving a conjecture of Grünbaum and Shephard on convex deformation of convex graphs and for giving a short proof of the result of Mani-Levistka, Guigas, and Klee on convex representation of infinite doubly periodic 3-connected planar graphs. The research on convex drawing of planar graphs was motivated by the desire of finding aesthetic drawings of graphs [3]. Arkin et al. [1] showed that there is a monotone path in some direction between every pair of vertices in any strictly convex drawing of a planar graph.

Open Problems

A convex drawing is called a *convex grid drawing* if each vertex is drawn at a grid point of an integer grid. Using canonical ordering and shift method, Chrobak and Kant [4] showed that every 3-connected plane graph has a convex grid drawing on an $(n-2) \times (n-2)$ grid and such a grid drawing can be found in linear time. However, the question of whether every planar graph which has a convex drawing admits a convex grid drawing on a grid of polynomial size remained as an open problem. Several research works are concentrated in this direction [6, 7, 13]. For example, Zhou and Nishizeki showed that every internally triconnected plane graph G whose decomposition tree $T(G)$ has exactly four leaves has a convex grid drawing on a $2n \times 4n = O(n^2)$ grid and presented a linear algorithm to find such a drawing [13].

Recommended Reading

1. Arkin EM, Connelly R, Mitchell JS (1989) On monotone paths among obstacles with applications to planning assemblies. In: Proceedings of the fifth annual symposium on computational geom-

etry (SCG '89), Saarbrücken. ACM, New York, pp 334–343
2. Chiba N, Yamanouchi T, Nishizeki T (1984) Linear algorithms for convex drawings of planar graphs. In: Bondy JA, Murty USR (eds) Progress in graph theory. Academic, Toronto, pp 153–173
3. Chiba N, Onoguchi K, Nishizeki T (1985) Drawing planar graphs nicely. Acta Inform 22:187–201
4. Chrobak M, Kant G (1997) Convex grid drawings of 3-connected planar graphs. Int J Comput Geom Appl 7:221–223
5. Hopcroft JE, Tarjan RE (1973) Dividing a graph into triconnected components. SIAM J Comput 2:135–158
6. Miura K, Azuma M, Nishizeki T (2006) Convex drawings of plane graphs of minimum outer apices. Int J Found Comput Sci 17:1115–1127
7. Miura K, Nakano S, Nishizeki T (2006) Convex grid drawings of four connected plane graphs. Int J Found Comput Sci 17:1032–1060
8. Nishizeki T, Rahman MS (2004) Planar graph drawing. Lecture notes series on computing, vol 12. World Scientific, Singapore
9. Thomassen C (1980) Planarity and duality of finite and infinite graphs. J Comb Theory 29:244–271
10. Thomassen C (1984) Plane representations of graphs. In: Bondy JA, Murty USR (eds) Progress in graph theory. Academic, Toronto, pp 43–69
11. Tutte WT (1960) Convex representations of graphs. Proc Lond Math Soc. 10:304–320
12. Tutte WT (1963) How to draw a graph. Proc Lond Math Soc. 13:743–768
13. Zhou X, Nishizeki T (2010) Convex drawings of internally triconnected plane graphs on $o(n^2)$ grids. Discret Math Algorithms Appl 2:347–362

Convex Hulls

Michael Hemmer[1] and Christiane Schmidt[2]
[1]Department of Computer Science, TU Braunschweig, Braunschweig, Germany
[2]The Selim and Rachel Benin School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem, Israel

Keywords

Computational geometry; Point sets

Years and Authors of Summarized Original Work

1972; Graham
1973; Jarvis
1977; Preparata, Hong
1996; Chan

Problem Definition

The *convex hull* of a set P of n points in $\mathbb{R}^d$ is the intersection of all convex regions that contain P. While convex hulls are defined for arbitrary d, the focus here is on $d = 2$ (and $d = 3$). For a more general overview, we recommend reading [7, 9] as well as [3].

A frequently used visual description for a convex hull in 2D is a rubber band: when we imagine the points in the plane to be nails and put a rubber band around them, the convex hull is exactly the structure we obtain by a tight rubber band; see Fig. 1.

The above definition, though intuitive, is hard to use for algorithms to compute the convex hull – one would have to intersect all convex supersets of P. However, one can show that there is an alternative definition of the convex hull of P: it is the set of all convex combinations of P.

Notation

For a point set $P = \{p_1, \ldots, p_n\}$, a *convex combination* is of the form

$$\lambda_1 p_1 + \lambda_2 p_2 + \ldots \lambda_n p_n \quad \text{with} \quad \lambda_i \geq 0, \sum \lambda_i = 1. \tag{1}$$

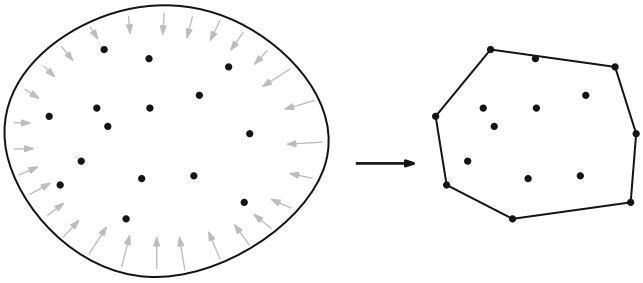
The convex hull, $CH(P)$, of P is the polygon that consists of all convex combinations of P. The *ordered* convex hull gives the ordered sequence of vertices on the boundary of $CH(P)$, instead of only the set of vertices that constitute the hull.

Key Results

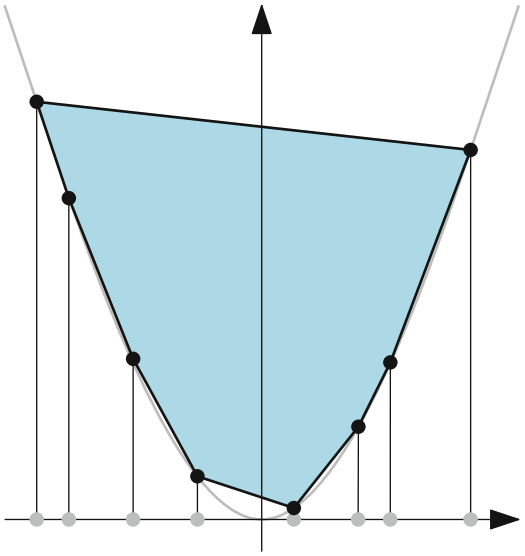
In the following, we present algorithms that compute the ordered convex hull of a given point set P in the plane. We start with a short proof for a lower bound of $\Omega(n \log n)$.

Lower Bound

Theorem 1 *Let P be a set of n points in the plane. An algorithm that computes the ordered convex hull is in $\Omega(n \log n)$.*



Convex Hulls, Fig. 1 The convex hull of a set of points in $\mathbb{R}^2$



Convex Hulls, Fig. 2 Set of numbers X in *gray*, the point set P in *black* and the convex hull $CH(P)$ in *blue*

Proof Given an unsorted list of numbers $X = \{x_1, x_2, \ldots, x_n\}$, we can lift these to a parabola as depicted in Fig. 2. Computing the convex hull $CH(P)$ of the resulting set $P = \{(x_i, x_i^2) \mid x_i \in X\}$ allows to output the sorted numbers by reading off the x-values of the vertices on the lower chain of $CH(P)$ in $O(n)$ time. Thus, a computation of $CH(P)$ in $o(n \log n)$ time would contradict the lower bound $\Omega(n \log n)$ for sorting.

Divide and Conquer by Preparata and Hong [8]

In the first step, the elements of P are sorted, and then the algorithm recursively divides P into subsets A and B of similar size. This is done until at most three points are left in each set, for which the convex hull is trivially computed. The sorting assures that the computed convex hulls are disjoint. Thus, in each step of the merge phase, the algorithm is given two ordered convex hulls $\mathrm{CH}(A)$ and $\mathrm{CH}(B)$, which are separated by a vertical line. To compute $\mathrm{CH}(A \cup B)$, it needs to find the two tangents supporting $\mathrm{CH}(A)$ and $\mathrm{CH}(B)$ from above and below, respectively. The procedure, which is often referred to as a "wobbly stick", is exemplified in Fig. 3.

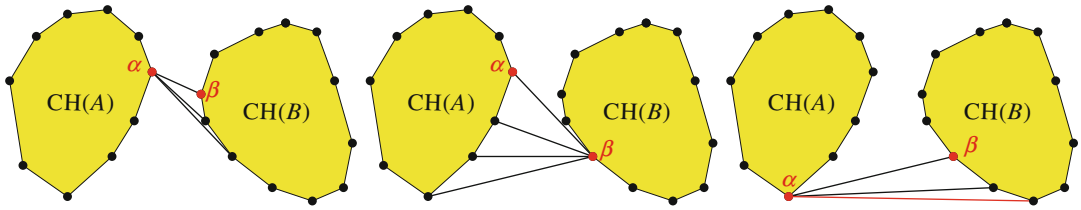
It is easy to see that each level of the merge phase requires $O(n)$ time, resulting in a total running time of $O(n \log n)$. A similar idea is also applicable in 3D.

Graham Scan [4]

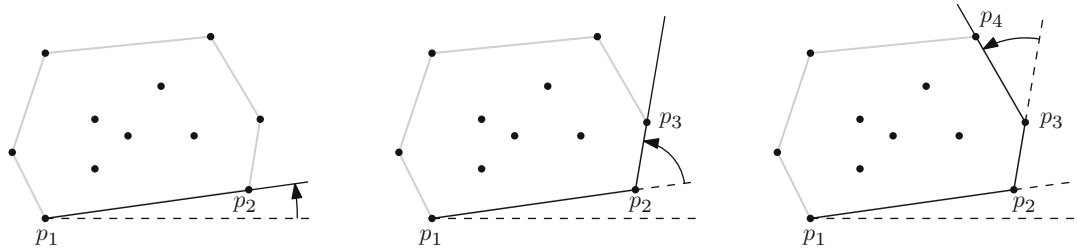
The Graham Scan starts with a known point on the hull as an anchor, the bottommost (rightmost) point p_1. The remaining $n - 1$ points are sorted according to the angles they form with the negative x-axis through p_1, from the largest angle to the smallest angle. The points p_i are processed using this angular order. For the next point p_i, the algorithm determines whether the last constructed hull edge xy and p_i form a left or a right turn. In case of a right turn, y is not part of the hull and is discarded. The discarding is continued as long as the last three points form a right turn. Once xy and p_i form a left turn, the next point in the angular order is considered. Because of the initial sorting step, the total running time of the Graham Scan is $O(n \log n)$.

Jarvis's March or Gift Wrapping [5]

Jarvis's March is an output-sensitive algorithm, i.e., the running time does depend on the size of the output. Its total running time is $O(nh)$, where h is the number of points on $\mathrm{CH}(P)$.



Convex Hulls, Fig. 3 Finding the tangent supporting CH(A) and CH(B) from below: considering CH(A) and CH(B) as obstacles, the algorithm iteratively tries to increment either α or β in clockwise (cw) and counterclockwise (ccw) order, respectively, while maintaining visibility of α and β. That is, it stops as soon as α does not see the ccw neighbor of β and β does not see the cw neighbor of α



Convex Hulls, Fig. 4 The first three steps of the gift wrapping algorithm. Starting at p_1, the algorithm acts as if it was wrapping a string around the point set

The algorithm starts with a known point on the hull, i.e., the bottommost (rightmost) point p_1. Just like wrapping a string around the point set, it then computes the next point on CH(P) in counterclockwise order: compare angles to all other points and choose the point with the largest angle to the negative x-axis. In general, the wrapping step is as follows: let p_{k-1} and p_k be the two previously computed hull vertices, then p_{k+1} is set to the point $p \in P$, $p \neq p_k$, that maximizes the angle $\angle p_{k-1} p_k p_{k+1}$; see Fig. 4. Each steps takes $O(n)$ time and finds one point on CH(P). Thus, the total running time is $O(nh)$.

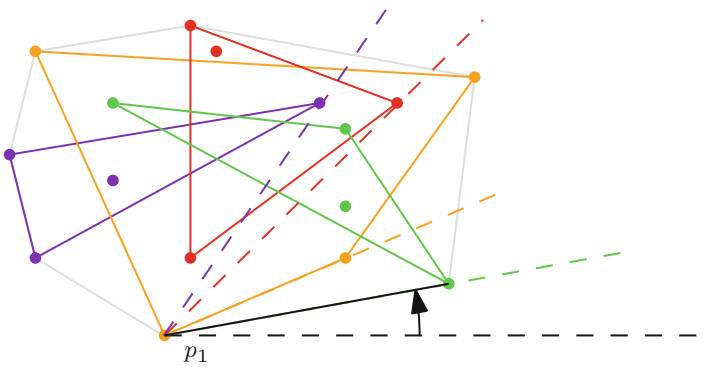
Chan's Algorithm [2]

In 1996, Chan presented an output-sensitive algorithm, with a worst-case optimal running time of $O(n \log h)$. This does not contradict the lower bound presented above, as it features n points on the hull. Let us for now assume that h is known, the number of points on the final convex hull. The algorithm runs in two phases.

Phase 1 splits P into $\lceil n/h \rceil$ groups of size at most h. Computing the convex hull for each set using, e.g., Graham Scan takes $O(h \log h)$. This results in $\lceil n/h \rceil$ (potentially overlapping) convex hulls and takes $O(\lceil n/h \rceil \cdot h \log h) = O(n \log h)$; see Fig. 5.

Phase 2 essentially applies Jarvis's March. Starting at the lowest leftmost point, it wraps a string around the set of convex hulls, i.e., for each hull it computes the proper tangent to the current point and chooses the tangent with the best angle in order to obtain the next point on the final convex hull. Computing the tangent for a hull of size h takes $O(\log h)$, which must be done for each of the $\lceil n/h \rceil$ hulls in each of the h rounds. Thus, the running time is $O(h \cdot \lceil n/h \rceil \log h) = O(n \log h)$.

Because h is not known, the algorithm does several such rounds for increasing values of h, until h is determined. In the initial round, it starts with a very small $h_0 = 4 = 2^{2^0}$ and continues with $h_i = 2^{2^i}$ in round i. As long as $h_i < h$, phase 1 is very quick. The second phase stops with an incomplete hull, knowing that h_i is still too small. That is, round i costs at most $n \log(h_i) = n2^i$. The algorithm terminates as soon as $h_i > h$. Thus, in total we obtain $\lceil \log \log h \rceil$ rounds. Therefore, the total



Convex Hulls, Fig. 5 Initial step of Jarvis's March in the first round of Chan's algorithm ($h_0 = 4$). Starting at p_1, the algorithm computes tangents to each convex hull (indicated in different colors) and selects the first tangent in counterclockwise order

cost is:

$$O\left(\sum_{t=1}^{\lceil \log\log h\rceil} n2^t\right) = O(n2^{\lceil \log\log h\rceil+1}) = O(n\log h). \quad (2)$$

Implementation

Like many geometric algorithms, the computation of the convex hull can be very sensitive to inconsistencies, due to rounding errors [6]. A well-maintained collection of exact implementations that eliminates problems due to rounding errors can be found in CGAL, the Computational Geometry Algorithms Library [1].

Cross-References

- ▸ Engineering Geometric Algorithms
- ▸ Robust Geometric Computation

Recommended Reading

1. CGAL (2014) Computational geometry algorithms library. http://www.cgal.org
2. Chan T (1996) Optimal output-sensitive convex hull algorithms in two and three dimensions. Discret Comput Geom 16(4):361–368. doi:10.1007/BF02712873
3. Devadoss SL, O'Rourke J (2011) Discrete and computational geometry. Princeton University Press, Princeton
4. Graham RL (1972) An efficient algorithm for determining the convex hull of a finite planar set. Inf Process Lett 1(4):132–133
5. Jarvis R (1973) On the identification of the convex hull of a finite set of points in the plane. Information Process Lett 2(1):18–21. doi:10.1016/0020-0190(73)90020-3
6. Kettner L, Mehlhorn K, Pion S, Schirra S, Yap CK (2004) Classroom examples of robustness problems in geometric computations. In: Proceedings of the 12th annual European symposium on algorithms, vol 3221, pp 702–713
7. O'Rourke J (1998) Computational geometry in C, 2nd edn. Cambridge University Press, New York
8. Preparata FP, Hong SJ (1977) Convex hulls of finite sets of points in two and three dimensions. Commun ACM 20(2):87–93
9. Sack JR, Urrutia J (eds) (2000) Handbook of computational geometry. North-Holland, Amsterdam

Coordinated Sampling

Edith Cohen
Tel Aviv University, Tel Aviv, Israel
Stanford University, Stanford, CA, USA

Keywords

Bottom-k sampling; Coordinated random sampling; Monotone estimation problems; Permanent random numbers

Years and Authors of Summarized Original Work

1972; Brewer, Early, Joice
1997; Cohen
1997; Broder

2013; Cohen, Kaplan
2014; Cohen

Problem Definition

Data is often sampled as a means of addressing resource constraints on storage, bandwidth, or processing – even when we have the resources to store the full data set, processing queries exactly over the data can be very expensive, and we therefore may opt for approximate fast answers obtained from the much smaller sample.

Our focus here is on data sets that have the form of a set of *keys* from some universe and multiple *instances*, which are assignments of nonnegative values to keys. We denote by v_{hi} the value of key h in instance i. Examples of data sets with this form include measurements of a set of parameters; snapshots of a state of a system; logs of requests, transactions, or activity; IP flow records in different time periods; and occurrences of terms in a set of documents. Typically, this matrix is very sparse – the vast majority of entries are $v_{ih} = 0$.

The sampling algorithms we apply can scan the full data set but retain the values v_{hi} of only a subset of the entries (h, i). From the sample, we would like to estimate functions (or statistics) specified over the original data. In particular, many common queries can be expressed as (or as a function of) the sum $\sum_{h\in H} f(v_{h\cdot})$ over selected keys $h \in H$ of a basic function f applied to the values of the key h in one or more instances. These include domain (subset) queries $\sum_{h\in H} v_{hi}$ which are the total weight of keys H in instance i, L_p distances between instances i, j which use $f(v_{h\cdot}) \equiv |v_{hi} - v_{hj}|^p$, one-sided distances which use $f(v_{h\cdot}) \equiv \max\{0, v_{hi} - v_{hj}\}^p$, and sums of quantiles (say maximum or median) of the tuple $(v_{h1}, \ldots, v_{hr})$.

The objective is to design a family of sampling scheme that is suitable for one or more query types. When the sampling scheme is specified, we are interested in designing estimators that use the information in the sample in the "best" way. The estimators are functions $\hat{f}$ that are applied to the sample and return an approximate answer to the query.

When sampling a single-instance i and aiming for approximation quality on domain queries and on sparse or skewed data, we use a *weighted sample*, meaning that the inclusion probability of an entry (h, i) in the sample depends (usually is increasing) with its weight v_{hi}. In particular, zero entries are never sampled. Two popular weighted sampling schemes are Poisson sampling, where entries are sampled independently, and bottom-k (order) sampling [12, 35, 36]. It is convenient for our purposes to specify these sampling schemes through a *rank function*, $r : [0, 1] \times V \to \mathbb{R}$, which maps seed-value pairs to a number $r(u, v)$ that is nonincreasing with u and nondecreasing with v. For each item h, we draw a *seed* $u(h) \sim U[0, 1]$ uniformly at random and compute the rank value $r(u(h), v_{hi})$. A Poisson sample includes a key $h \iff r(u(h), v_{hi}) \geq T_{hi}$, where T_{hi} are fixed thresholds. A bottom-k sample includes the k items with the highest ranks. (The term bottom-k is due to equivalently using the inverse rank function and lowest k ranks [12–14, 35, 36].)

Specifically, Poisson probability proportional to size (PPS) [28] samples include each key with probability proportion to its value. They are specified using the rank function $r(u, v) = v/u$ and a fixed $T_i \equiv T_{hi}$ across all keys in the instance. Priority (sequential Poisson) samples [22, 33, 38] are bottom-k samples utilizing the PPS ranks $r(u, v) = v/u$, and successive weighted samplings without replacement [12, 23, 35] are bottom-k samples with the rank function $r(u, v) = -v/\ln(1 - u)$. All these sampling algorithms can be easily implemented when the data is streamed or distributed.

Queries over a single instance can be estimated using inverse probabilities [29]. For each sampled key h, we can compute the probability $q(h)$ that it is included in the sample. With Poisson sampling, this probability is that of $u \sim U[0, 1]$ satisfying $r(u(h), v_{hi}) \geq T_{hi}$. With bottom-$k$ sampling, we use a conditioned version [13, 22, 38]: The probability $q(h)$ is that of $r(u(h), v_{hi})$ being larger than the kth largest value among $r(u(y), v_{yi})$, where y are all keys other than h and $u(y)$ is fixed to be as in the current sample. Note that this threshold value is available to us from the sample, and hence

keys:	1	2	3	4	5	6	7	8
Instance1:	1	3	2	0	1	4	0	1
Instance2:	0	1	3	2	0	1	2	3
PPS sampling probabilities for T=4 (sample of expected size 3):								
Instance1:	0.25	0.75	0.50	0.00	0.25	1.00	0.00	0.25
Instance2:	0.00	0.25	0.75	0.50	0.00	0.25	0.50	0.75

Coordinated Sampling, Fig. 1 Two instances with 8 keys and respective PPS sampling probabilities for threshold value 4, so a key with value v is sampled with probability $\min\{1, v/4\}$. To obtain two coordinated PPS samples of the instances, we associate an independent $u(i) \sim U[0,1]$ with each key $i \in [8]$. We then sample $i \in [8]$ in instance $h \in [2]$ if and only if $u(i) \leq v_{hi}/4$, where v_{hi} is the value of i in instance h. When coordinating the samples this way, we make them as similar as possible. In the example, key 8 will always (for any drawing of seeds) be sampled in instance 2 if it is sampled in instance 1 and vice versa for key 2

$q(h)$ can be computed for each sampled key. We then estimate $\sum_{h\in H} f(v_{hi})$ using the sum over sampled keys that are in H of $f(v_{hi})/q(h)$. This estimator is a *sum* estimator in that it is applied separately for each key: $\hat{f}(h) = 0$ when the key is not sampled and is $\hat{f}(h) = f(v_{hi})/q(h)$ otherwise. When the sampling scheme is such that $q(h) > 0$ whenever $f(v_{hi}) > 0$, this estimator is unbiased. Moreover, this is also the optimal sum estimator, in terms of minimizing variance. We note that tighter estimators can be obtained when the total weight of the instance is available to the estimator [13].

What Is Sample Coordination?

When the data has multiple instances, we distinguish between a data source that is *dispersed*, meaning that different entries of each key occur in different times or locations and *co-located* if all entries occur together. These scenarios [17] impose different constraints on the sampling algorithm. In particular, with co-located data, it is easy to include the complete tuple $v_{h\cdot}$ of each key h that is sampled in at least one instance, whereas with dispersed data, we would want the sampling of one entry v_{hi} not to depend on the values v_{hj} in other instances j.

When sampling, we can redraw a fresh set of random seed values $u(h)$ for each instance, which results in the samples being independent. Samples of different instances are *coordinated* when the set of seeds $u(h)$ is common in all instances. Scalable sharing of seeds when data is dispersed is facilitated through random hash functions $u(h)$, where the only requirement for our purposes is uniformity and pairwise independence.

Coordinated sampling has the property that the samples of different instances are more similar when the instances are more similar, a property also known as Locality Sensitive Hashing (LSH) [26, 30, 31]. Figure 1 contains an example data set of two instances and the PPS sampling probabilities of each item in each instance. When the samples are coordinated, a key sampled in one instance is always sampled in the instance with a higher inclusion probability.

Why Use Coordination?

Co-located Data

With co-located data, coordination allows us to minimize the sample size, which is the (expected) total number of included keys, while ensuring that the sample "contains" a desired Poisson or bottom-k sample of each instance.

For a key h, we can consider its respective inclusion probabilities in each of the instances (for bottom-k, inclusion may be conditioned on seeds of other keys). With coordination, the inclusion probability in the combined sample, $q(h)$, is always the maximum of these probabilities. Estimation with these samples is easy: We estimate $\sum_{h\in H} f(v_{h\cdot})$ using the Horvitz-Thompson estimator (inverse probabilities) [29]. The estimate is the sum $f(v_{h\cdot})/q(h)$ over sampled keys that are in H. Since the complete tuple $v_{h\cdot}$ is available for

each sampled key, we can compute $f(v_h.)$, $q(h)$, and thus the estimate.

When the query involves entries from a single-instance i, the variance of this estimate is at most that obtained from a respective sample of i. This is because the inclusion probability $q(h)$ of each key is at least as high as in the respective single-instance sample. Therefore, by coordinating the samples, we minimize the total number of keys included while ensuring estimation quality with respect to each instance.

Dispersed Data

With dispersed data, coordination is useful when we are interested in both domain queries over a single instance at a time and some queries that involve complex relation, such as similarity queries between multiple instances. Estimation of more complex relations, however, can be much more involved. Intuitively, this is because the sample can provide us with partial information on $f(v_h.)$ that is short of the exact value but still lower bounds it by a positive amount. Therefore, in this case, inverse probability estimators may not be optimal or even well defined – there could be zero probability of knowing the exact value, but the function f can have a nonnegative unbiased estimator. In the sequel, we overview estimators that are able to optimally use the available information.

Implicit Data

Another setting where coordination arises is when the input is not explicit, for example, expressed as relations in a graph, and coordinated samples can be obtained much more efficiently than independent samples. In this case, we work with coordinated samples even when we are interested in queries that are approximated well with independent samples.

Key Results

We now overview results on estimators that are applied to coordinated samples of dispersed data.

We first observe that coordinated PPS and bottom-k samples are mergeable, meaning that a respective sample of the union, or more generally, of a new instance whose weight of each key is the coordinate-wise maxima of several instances can be computed from the individual samples of each instance. This makes some estimation problems very simple. Even in these cases, however, better estimators of cardinalities of unions and intersections (key-wise maxima or minima) can be computed when we consider all sampled keys in the two sets rather than just the sample of the union. Such estimators for the 0/1 case are presented in [14] and for general nonnegative weights in [17].

The general question is to derive estimators for an arbitrary function $f \geq 0$ with respect to a coordinated sampling scheme. This problem can be formalized as a Monotone Estimation Problem (MEP) [9]: The smaller the seed $u(h) \in U[0, 1]$ is, the more information we have on the values $v_h.$ and therefore on $f(v_h.)$. We are interested in deriving estimators $\hat{f}$ that are nonnegative; this is because we are interested in nonnegative functions and estimates should be from the same range. We also seek unbiasedness, because we are ultimately estimating a sum over many keys, and bias accumulates even when sampling of different keys are independent. Other desirable properties are finite variance (for any $v_h.$ in the domain) or bounded estimates. A complete characterization of functions f for which estimators with subsets of these properties exist is given in [15].

We are also interested in deriving estimators that are *admissible*. Admissibility is Pareto optimality, meaning that any other estimator with lower variance on some data would have higher variance on another. Derivations of admissible estimators (for any MEP for which an unbiased nonnegative estimator exists) are provided in [9]. Of particular interest is the L* estimator, which is the unique admissible monotone estimator. By monotone, we mean that when there is more information, that is, when $u(h)$ is higher, the estimate $\hat{f}$ is at least as high.

A definition of *variance competitiveness* of MEP estimators is provided in [15]. The competitive ratio of an estimator $\hat{f}$ is the maximum, over all possible inputs (data values) $v_h.$, of the ratio of the integral of the square of $\hat{f}$ to the minimum possible by a nonnegative unbiased estimator. It turns out that the L* has ratio of at most 4 on any

MEP for which a nonnegative unbiased estimator with finite variances exists [9]. Two interesting remaining open problems, partially addressed in [10], are to design an estimator with minimum ratio for a given MEP and also to bound the maximum minimum ratio over all MEPs that admit an unbiased nonnegative estimator with finite variance.

Applications

We briefly discuss the history and some applications of coordinated sampling.

Sample coordination was proposed in 1972 by Brewer, Early, and Joice [2], as a method to maximize overlap and therefore minimize overhead in repeated surveys [34, 36, 37]: The values of keys change, and therefore, there is a new set of PPS sampling probabilities. With coordination, the sample of the new instance is as similar as possible to the previous sample, and therefore, the number of keys that need to be surveyed again is minimized. The term permanent random numbers (PRN) is used in the statistics literature for sample coordination.

Coordination was subsequently used by computer scientists to facilitate efficient processing of large data sets, as estimates obtained over coordinated samples are much more accurate than possible with independent samples [1, 3–6, 8, 13, 14, 16, 17, 21, 24, 25, 27, 32].

In some applications, the representation of the data is not explicit, and coordinated samples are much easier to compute than independent samples. One such example is computing all-distance sketches, which are coordinated samples of (all) d-neighborhoods of all nodes in a graph [6,7,11–13,32]. These sketches support centrality and similarity and influence queries useful in the analysis of massive graph data sets such as social networks or Web graphs [18–20].

Recommended Reading

1. Beyer KS, Haas PJ, Reinwald B, Sismanis Y, Gemulla R (2007) On synopses for distinct-value estimation under multiset operations. In: SIGMOD, Beijing. ACM, pp 199–210
2. Brewer KRW, Early LJ, Joyce SF (1972) Selecting several samples from a single population. Aust J Stat 14(3):231–239
3. Broder AZ (1997) On the resemblance and containment of documents. In: Proceedings of the compression and complexity of sequences, Salerno. IEEE, pp 21–29
4. Broder AZ (2000) Identifying and filtering near-duplicate documents. In: Proceedings of the 11th annual symposium on combinatorial pattern matching, Montreal. LNCS, vol 1848. Springer, pp 1–10
5. Byers JW, Considine J, Mitzenmacher M, Rost S (2004) Informed content delivery across adaptive overlay networks. IEEE/ACM Trans Netw 12(5):767–780
6. Cohen E (1997) Size-estimation framework with applications to transitive closure and reachability. J Comput Syst Sci 55:441–453
7. Cohen E (2013) All-distances sketches, revisited: HIP estimators for massive graphs analysis. Tech. Rep. cs.DS/1306.3284, arXiv http://arxiv.org/abs/1306.3284
8. Cohen E (2014) Distance queries from sampled data: accurate and efficient. In: ACM KDD, New York. Full version: http://arxiv.org/abs/1203.4903
9. Cohen E (2014) Estimation for monotone sampling: competitiveness and customization. In: ACM PODC, Paris. http://arxiv.org/abs/1212.0243, full version http://arxiv.org/abs/1212.0243
10. Cohen E (2014) Variance competitiveness for monotone estimation: tightening the bounds. Tech. Rep. cs.ST/1406.6490, arXiv http://arxiv.org/abs/1406.6490
11. Cohen E, Kaplan H (2007) Spatially-decaying aggregation over a network: model and algorithms. J Comput Syst Sci 73:265–288. Full version of a SIGMOD 2004 paper
12. Cohen E, Kaplan H (2007) Summarizing data using bottom-k sketches. In: ACM PODC, Portland
13. Cohen E, Kaplan H (2008) Tighter estimation using bottom-k sketches. In: Proceedings of the 34th international conference on very large data bases (VLDB), Auckland. http://arxiv.org/abs/0802.3448
14. Cohen E, Kaplan H (2009) Leveraging discarded samples for tighter estimation of multiple-set aggregates. In: ACM SIGMETRICS, Seattle
15. Cohen E, Kaplan H (2013) What you can do with coordinated samples. In: The 17th international workshop on randomization and computation (RANDOM), Berkeley. Full version: http://arxiv.org/abs/1206.5637
16. Cohen E, Wang YM, Suri G (1995) When piecewise determinism is almost true. In: Proceedings of the pacific rim international symposium on fault-tolerant systems, Newport Beach, pp 66–71
17. Cohen E, Kaplan H, Sen S (2009) Coordinated weighted sampling for estimating aggregates over multiple weight assignments. In: Proceedings of the VLDB endowment, Lyon, France, vol 2(1–2). Full version: http://arxiv.org/abs/0906.4560

18. Cohen E, Delling D, Fuchs F, Goldberg A, Goldszmidt M, Werneck R (2013) Scalable similarity estimation in social networks: closeness, node labels, and random edge lengths. In: ACM COSN, Boston
19. Cohen E, Delling D, Pajor T, Werneck RF (2014) Sketch-based influence maximization and computation: scaling up with guarantees. In: ACM CIKM, Shanghai. http://research.microsoft.com/apps/pubs/?id=226623, full version http://research.microsoft.com/apps/pubs/?id=226623
20. Cohen E, Delling D, Pajor T, Werneck RF (2014) Timed influence: computation and maximization. Tech. Rep. cs.SI/1410.6976, arXiv http://arxiv.org/abs/1410.06976
21. Das A, Datar M, Garg A, Rajaram S (2007) Google news personalization: scalable online collaborative filtering. In: WWW, Banff, Alberta, Canada
22. Duffield N, Thorup M, Lund C (2007) Priority sampling for estimating arbitrary subset sums. J Assoc Comput Mach 54(6)
23. Efraimidis PS, Spirakis PG (2006) Weighted random sampling with a reservoir. Inf Process Lett 97(5):181–185
24. Gibbons PB (2001) Distinct sampling for highly-accurate answers to distinct values queries and event reports. In: International conference on very large databases (VLDB), Roma, pp 541–550
25. Gibbons P, Tirthapura S (2001) Estimating simple functions on the union of data streams. In: Proceedings of the 13th annual ACM symposium on parallel algorithms and architectures, Crete Island. ACM
26. Gionis A, Indyk P, Motwani R (1999) Similarity search in high dimensions via hashing. In: Proceedings of the 25th international conference on very large data bases (VLDB'99), Edinburgh
27. Hadjieleftheriou M, Yu X, Koudas N, Srivastava D (2008) Hashed samples: selectivity estimators for set similarity selection queries. In: Proceedings of the 34th international conference on very large data bases (VLDB), Auckland
28. Hájek J (1981) Sampling from a finite population. Marcel Dekker, New York
29. Horvitz DG, Thompson DJ (1952) A generalization of sampling without replacement from a finite universe. J Am Stat Assoc 47(260):663–685
30. Indyk P (2001) Stable distributions, pseudorandom generators, embeddings and data stream computation. In: Proceedings of the 41st IEEE annual symposium on foundations of computer science, Redondo Beach. IEEE, pp 189–197
31. Indyk P, Motwani R (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the 30th annual ACM symposium on theory of computing, Texas. ACM, pp 604–613
32. Mosk-Aoyama D, Shah D (2006) Computing separable functions via gossip. In: ACM PODC, Denver
33. Ohlsson E (1998) Sequential poisson sampling. J Off Stat 14(2):149–162
34. Ohlsson E (2000) Coordination of PPS samples over time. In: The 2nd international conference on establishment surveys. American Statistical Association, pp 255–264
35. Rosén B (1972) Asymptotic theory for successive sampling with varying probabilities without replacement, I. Ann Math Stat 43(2):373–397. http://www.jstor.org/stable/2239977
36. Rosén B (1997) Asymptotic theory for order sampling. J Stat Plan Inference 62(2):135–158
37. Saavedra PJ (1995) Fixed sample size PPS approximations with a permanent random number. In: Proceedings of the section on survey research methods, Alexandria. American Statistical Association, pp 697–700
38. Szegedy M (2006) The DLT priority sampling is essentially optimal. In: Proceedings of the 38th annual ACM symposium on theory of computing, Seattle. ACM

Counting by ZDD

Shin-ichi Minato
Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan

Keywords

BDD; Binary decision diagram; Data compression; Dynamic programming; Enumeration; Graph algorithm; Indexing; ZDD; Zero-suppressed BDD

Years and Authors of Summarized Original Work

1986; R. E. Bryant
1993; S. Minato
2009; D. E. Knuth
2013; S. Minato
2013; T. Inoue, H. Iwashita, J. Kawahara, S. Minato

Problem Definition

We consider a framework of the problems to enumerate all the subsets of a given graph, each

subset of which satisfies a given constraint. For example, enumerating all Hamilton cycles, all spanning trees, all paths between two vertices, all independent sets of vertices, etc. When we assume a graph $G = (V, E)$ with the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and the edge set $E = \{e_1, e_2, \ldots, e_m\}$, a graph enumeration problem is to compute a subset of the power set 2^E (or 2^V), each element of which satisfies a given constraint. In this model, we can consider that each solution is a combination of edges (or vertices), and the problem is how to represent the set of solutions and how to generate it efficiently.

Constraints

Any kind of constraint for the graph edges and vertices can be considered. For example, we consider to enumerate all the simple (self-avoiding) paths connecting the two vertices s and t on the graph shown in Fig. 1. The constraint is described as:

1. At a terminal vertex (s and t), only one edge is selected and connected to the vertex.
2. At the other vertices, none or just two edges are selected and connected to the vertex, respectively.
3. The set of selected edges forms a connected component.

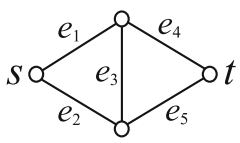
In this example, the set of solutions can be represented as a combination of the edges, $\{e_1e_3e_5,\ e_1e_4,\ e_2e_3e_4,\ e_2e_5\}$.

Key Results

A binary decision diagram (BDD) is a representation of a Boolean function, one of the most basic models of discrete structures. After the epoch-making paper [1] by Bryant in 1986, BDD-based methods have attracted a great deal of attention. BDD was originally invented for the efficient Boolean function manipulation required in VLSI logic design, but Boolean functions are also used for modeling many kinds of combinatorial problems. Zero-suppressed BDD (ZDD) [8] is a variant of BDD, customized for manipulating "sets of combinations." ZDDs have been successfully applied not only for VLSI design but also for solving various combinatorial problems, such as constraint satisfaction, frequent pattern mining, and graph enumeration.

Recently, D. E. Knuth presented a surprisingly fast algorithm "Simpath" [7] to construct a ZDD which represents all the paths connecting two points in a given graph structure. This work is important because many kinds of practical problems are efficiently solved by some variations of this algorithm. We generically call such ZDD construction method "frontier-based methods."

Counting by ZDD, Fig. 1
Example of a graph



BDDs/ZDDs for Graph Enumeration

A binary decision diagram (BDD) is a graph representation for a Boolean function, developed for VLSI design. A BDD is derived by reducing a binary decision tree, which represents a decision-making process by the input variables. If we fix the order of input variables and apply the following two reduction rules, then we have a compact canonical form for a given Boolean function:

1. Delete all redundant nodes whose two edges have the same destination.
2. Share all equivalent nodes having the same child nodes and the same variable.

The compression ratio achieved by using a BDD instead of a decision tree depends on the property of Boolean function to be represented, but it can be 10–100 times in some practical cases.

A zero-suppressed BDD (ZDD) is a variant of BDD, customized for manipulating sets of combinations. This data structure was first introduced by Minato [8]. ZDDs are based on the special reduction rules different from ordinary ones, as follows:

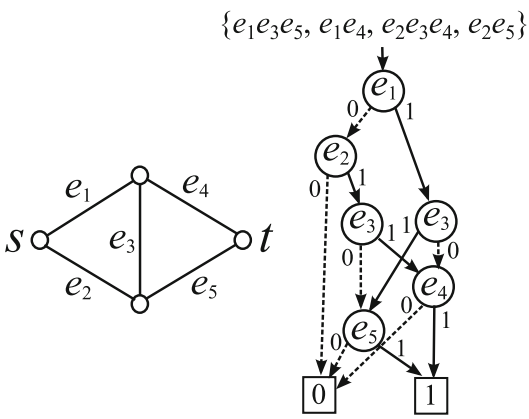
1. Delete all nodes whose 1-edge directly points to the 0-terminal node, but do not delete the nodes which were deleted in ordinary BDDs.

This new reduction rule is extremely effective, when it is applied to a set of sparse combinations. If each item appears in 1 % of combinations in average, ZDDs are possibly more compact than ordinary BDDs, by up to 100 times. Such situations often appear in real-life problems, for example, in a supermarket, the number of items in a customer's basket is usually much less than all the items displayed there. Because of such an advantage, ZDDs are now widely recognized as the most important variant of BDD.

ZDDs can be utilized for enumerating and indexing the solutions of a graph problem. For example, Fig. 2 shows the ZDD enumerating all the simple paths of the graph the same as Fig. 1. The ZDD has four paths from the root node to the 1-terminal node, and each path corresponds to the solution of the problem, where $e_i = 1$ means to use the edge e_i and $e_i = 0$ means not to use e_i.

Frontier-Based Method

In 2009, Knuth published the surprisingly fast algorithm "Simpath" [7] (Vol. 4, Fascicle 1, p. 121, or p. 254 of Vol. 4A) to construct a ZDD which represents all the simple (or self-avoiding) paths connecting two points s and t in a given graph (not necessarily the shortest ones but ones not passing through the same point twice). This work is important because many kinds of practical problems can be efficiently solved by some variations of this algorithm. Knuth provides his own C source codes on his web page for public access, and the program is surprisingly fast. For example, in a 14×14 grid graph (420 edges in total), the number of self-avoiding paths between opposite corners is exactly 227449714676812739631826459327989863387613323440 ($\approx 2.27 \times 10^{47}$) ways. By applying the Simpath algorithm, the set of paths can be compressed into a ZDD with only 144759636 nodes, and the computation time is only a few minutes.



Counting by ZDD, Fig. 2 ZDD representing the paths from s to t

The Simpath algorithm is minutely written in Knuth's book, and his source codes are also provided, but it is not easy to read. The survey paper [9] will be helpful for understanding the basic mechanism of the Simpath algorithm.

The Simpath algorithm belongs to the method of dynamic programming, by scanning the given graph from left to right like a moving frontier line. If the frontier grows larger in the computation process, more intermediate states appear and more computation time is required. Thus, it is important to keep the frontier small. The maximum size of the frontier depends on the given graph structures and the order of the edges. Planar and narrow graphs tend to have small frontier.

Knuth described in his book [7] that the Simpath algorithm can easily be modified to generate not only $s - t$ paths but also Hamilton paths, directed paths, some kinds of cycles, and many other problems by slightly changing the mate data structure. We generically call such ZDD construction method "frontier-based methods."

Applications

Here we list graph problems which can be enumerated and indexed by a ZDD using a frontier-based method.

All $s - t$ paths, $s - t$ paths with length k, k-pairs of $s - t$ paths, all cycles, cycles with length k, Hamilton paths/cycles, directed

paths/cycles, all connected components, spanning trees/forests, Steiner trees, all cutsets, k-partitioning, calculating probability of connectivity, all cliques, all independent sets, graph colorings, tilings, and perfect/imperfect matching.

These problems are strongly related to many kinds of real-life problems. For example, path enumeration is of course important in geographic information systems and is also used for dependency analysis of a process flow chart, fault analysis of industrial systems, etc. Recently, Inoue et al. [5] discussed the design of electric power distribution systems. Such civil engineering systems are usually near to planar graphs, so the frontier-based method is very effective in many cases. They succeeded in generating a ZDD to enumerate all the possible switching patterns in a realistic benchmark of an electric power distribution system with 468 switches. The obtained ZDD represents as many as 10^{60} of the valid switching patterns, but the actual ZDD size is less than 100 MB, and computation time is around 30 min. After generating the ZDD, all valid switching patterns are compactly represented, and we can efficiently discover the switching patterns with maximum, minimum, and average cost. We can also efficiently apply additional constraints to the current solutions. In this way, frontier-based methods can be utilized for many kinds of real-life problems.

Open Problems

Frontier-based method is a general framework of the algorithm, and we have to develop particular algorithm for enumerating graphs to satisfy a given constraint. It is sometimes time consuming, and it is not clearly understood which kind of graphs can be generated easily and which are hard or impossible.

Experimental Results

It is an interesting problem how large n is possible to count the number of simple paths included in an $n \times n$ grid graph with s and t at the opposite corners. We have worked for this problems and succeeded in counting the total number of self-avoiding $s - t$ paths for the 26×26 grid graph. The number is exactly:

173699315862792729311754404212364989003

722295882881406046637037209103424132 7

613476278921819349800610708229622314338

049134829002672193112962770873889085 3

908108906396.

This is the current world record and is officially registered in the On-Line Encyclopedia of Integer Sequences [10] in November 2013. The detailed techniques for solving larger problems are presented in the report by Iwashita et al. [6]

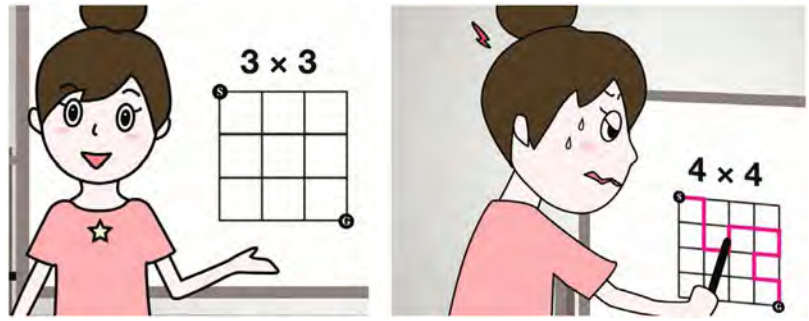
A Related YouTube Video

In 2012, Minato supervised a short educational animation video (Fig. 3). The video is mainly designed for junior high school to college students, to show the power of combinatorial explosion and the state-of-the-art techniques for solving such hard problems. This video uses the simple path enumeration problem for $n \times n$ grid graphs. The story is that the teacher counts the total number of paths for children starting from $n = 1$, but she will be faced with a difficult situation, since the number grows unbelievably fast. She would spend 250,000 years to count the paths for the 10×10 grid graph by using a supercomputer if she used a naive method. The story ends by telling that a state-of-the-art algorithm can finish the same problem in a few seconds. The video is now shown in YouTube [2] and received more than 1.5 million views, which is an extraordinary case in the scientific educational contents. We hear that Knuth also enjoyed this video and shared it to several of his friends.

Graphillion: Open Software Library

The above techniques of data structures and algorithms have been implemented and published as an open software library, named “Graphillion”

Counting by ZDD, Fig. 3 Screenshots of the animation video [2]



[3, 4]. Graphillion is a library for manipulating very large sets of graphs, based on ZDDs and frontier-based method. Traditional graph libraries maintain each graph individually, which causes poor scalability, while Graphillion handles a set of graphs collectively without considering individual graph. Graphillion is implemented as a Python extension in C++, to encourage easy development of its applications without introducing significant performance overhead.

URLs to Code and Data Sets

The open software library "Graphillion" can be found on the web page at http://graphillion.org, the YouTube video http://www.youtube.com/watch?v=Q4gTV4r0zRs, and the On-Line Encyclopedia of Integer Sequences (OEIS) on the self-avoiding path enumeration problem https://oeis.org/A007764.

Recommended Reading

1. Bryant RE (1986) Graph-based algorithms for Boolean function manipulation. IEEE Trans Comput C-35(8):677–691
2. Doi S et al (2012) Time with class! let's count! (the art of 10^{64} – understanding vastness –). YouTube video, http://www.youtube.com/watch?v=Q4gTV4r0zRs
3. Inoue T et al (2013) Graphillion. http://graphillion.org/
4. Inoue T, Iwashita H, Kawahara J, Minato S (2013) Graphillion: ZDD-based software library for very large sets of graphs. In: Proceedings of the workshop on synthesis and simulation meeting and international interchange (SASIMI-2013), R4-6, Sapporo, pp 237–242
5. Inoue T, Takano K, Watanabe T, Kawahara J, Yoshinaka R, Kishimoto A, Tsuda K, Minato S, Hayashi Y (2014) Distribution loss minimization with guaranteed error bound. IEEE Trans Smart Grid 5(1):102–111
6. Iwashita H, Nakazawa Y, Kawahara J, Uno T, Minato S (2013) Efficient computation of the number of paths in a grid graph with minimal perfect hash functions. TCS technical reports TCS-TR-A-10-64, Division of Computer Science, Hokkaido University
7. Knuth DE (2009) Bitwise tricks & techniques; binary decision diagrams. The art of computer programming, vol 4, fascicle 1. Addison-Wesley, Upper Saddle River
8. Minato S (1993) Zero-suppressed BDDs for set manipulation in combinatorial problems. In: Proceedings of 30th ACM/IEEE design automation conference (DAC'93), Dallas, pp 272–277
9. Minato S (2013) Techniques of BDD/ZDD: brief history and recent activity. IEICE Trans Inf Syst E96-D(7):1419–1429
10. OEIS-simpath (2013) Number of nonintersecting (or self-avoiding) rook paths joining opposite corners of an $n \times n$ grid, the on-line encyclopedia of integer sequences. https://oeis.org/A007764

Counting Triangles in Graph Streams

Madhav Jha
Sandia National Laboratories, Livermore, CA, USA
Zenefits, San Francisco, CA, USA

Keywords

Counting triangles; Streaming algorithms

Years and Authors of Summarized Original Work

2013; Jha, Seshadhri, Pinar

Problem Definition

A *graph stream* is a sequence of unordered pairs of elements from a set V implicitly describing an underlying graph G on vertex set. The unordered pairs represent edges of the graph. A *triangle* is a triple of vertices $u, v, w \in V$ which form a 3-clique, that is, every unordered pair of vertices of the set $\{u, v, w\}$ is connected by an edge. In this article, we investigate the problem of counting the number of triangles in an input graph G given as a graph stream. Furthermore, we restrict our attention to algorithms which are severely limited in total space (in particular, they cannot store the entire stream) and are allowed only a single scan over the stream of edges.

Next we describe the streaming setting more formally. Consider a sequence of *distinct* unordered pairs or, equivalently, edges $e_1, e_2, \ldots, e_m$ on the set V. Let G_t be the graph formed by the first t edges of the stream where $t \in \{1, \ldots, m\}$. Denoting the empty graph by G_0, we see that graph G_t is obtained from G_{t-1} by *inserting* edge e_t for all $t \in \{1, \ldots, m\}$. An edge $\{u, v\}$ of the stream implicitly introduces vertex labels u and v. New vertices are therefore implicitly added as new labels. In this article, we do not consider edge or vertex deletions, nor do we allow repeated edges. The problem of counting triangles even in this simple setting has received a lot of attention [1, 2, 6, 12–15, 17].

A streaming algorithm has a small working memory M and gets to scan the input stream in a sequential fashion at most a few times. In this article, we only consider algorithms which make a single pass over the input stream. Thus, the algorithm proceeds by sequentially reading each edge e_t from the input graph stream and updating its data structures in M using e_t. The algorithm cannot read an edge that has already passed again. (It may remember it in M.) Since the size of M is much smaller than m, the algorithm must work with only a "sketch" of the input graph stored in M. The streaming algorithm has access to random coins and typically maintains a random sketch of the input graph (e.g., a random subsample of the input edge stream).

The aim is to output an accurate estimate of the number of triangles in graph G_m. In fact, we require that the algorithm output a running estimate of the number of triangles T_t in graph G_t seen so far as it is reading the edge stream. It is also of interest to output an estimate of another quantity, related to the number of triangles, called *transitivity*. The transitivity of a graph, denoted κ, is the fraction of length-2 paths which form triangles. A path of length 2 is also called a *wedge*. A wedge $\{\{u, v\}, \{u, w\}\}$ is *open* (respectively, *closed*) if the edge $\{v, w\}$ is absent (respectively, present) in the graph. Every closed wedge is part of a triangle, and every triangle has exactly three closed wedges. This immediately gives a formula for transitivity: $\kappa = 3T/W$, where T and W are the total number of triangles and wedges in the graph, respectively. We use the subscript t, as in T_t, W_t, and κ_t, to denote corresponding quantities for graph G_t. The key result described in this article is a small space single-pass streaming algorithm for maintaining a running estimate for each of T_t, W_t, and κ_t.

Outline of the Rest of the Article

The bulk of this article is devoted to the description of results and algorithms of [12]. This is complemented by a section on related work where we briefly describe some of the other algorithms for triangle counting. This is followed by applications of triangle counting, an open problem, a section describing experimental results, and, finally, references used in this article.

Key Results

The main result (from [12]) presented in this article is a single pass, $O(m/\sqrt{T})$-space streaming algorithm to estimate the transitivity and the number of triangles of the input graph with small additive error.

The Algorithm of [12] and Its Analysis

The starting point of the algorithm is the wedge sampling idea from [19]. The transitivity of a

graph is precisely the probability that a uniformly random wedge from the graph is closed. Thus, estimating transitivity amounts to approximating the bias of a coin flip simulated by the following probabilistic experiment: sample a wedge $\{\{u, v\}, \{u, w\}\}$ uniformly from the set of wedges and output "Heads" if it is closed and "Tails" otherwise. One can check whether a wedge is closed by checking if $\{v, w\}$ is an edge in the graph. If, in addition, we have an accurate estimate of W, the triangle count can be estimated using $T = \kappa/3 \cdot W$.

If we adopt the described strategy, we need a way to sample a wedge uniformly from the *graph stream*. While this task by itself appears rather challenging, we note that sampling an edge uniformly from the graph stream can be done easily via an adaptation of reservoir sampling. (See below for details.) Can we use an edge sampling primitive to sample wedges uniformly from a graph stream? This is exactly what [12] achieves. Before we describe the algorithm of [12], we present a key primitive which is also used in other works on counting triangles in graph streams.

A key Algorithmic Tool: A Reservoir of Uniform Edges

This algorithmic tool allows one to maintain a set R_t of k edges while reading the edge stream with the following guarantee for every t: each of the k edges in R_t is selected uniformly from the edges of G_t and all edges are mutually independent. The idea is to adapt the classic reservoir sampling [21]. More precisely, at the beginning of the stream, R_0 consists of k empty slots. While reading the edge stream, on observing edge e_t, do the following probabilistic experiment independently for each of the k slots of R_{t-1} to yield R_t: (i) sample a random number x uniformly from [0, 1], and (ii) if $x \leq 1/t$, replace the slot with e_t. Otherwise, keep the slot unchanged.

How does a reservoir of edges R_t help in sampling wedges from the graph stream? When the edge reservoir R_t is large enough, there are many pairs of edges in R_t sharing a common vertex, yielding many wedges. Further, if the transitivity of the input graph is high, many of these wedges will in fact be closed wedges, aka, triangles. This is the idea behind the algorithm of [12]. The key theoretical insight is that there is a *birthday paradox*-like situation here: many uniform edges result in sufficiently many "collisions" to form wedges. Recall (e.g., from Chap.II.3 of [10]) that the classic birthday paradox states that if we choose 23 random people, the probability that 2 of them share a birthday is at least 1/2. In our setting, edges correspond to people and "sharing a birthday" corresponds to sharing a common vertex.

A Key Analytical Tool: The Birthday Paradox

Suppose $R_1, \ldots, R_k$ are i.i.d. samples from the uniform distribution on the edges of G. Then for distinct $i, j \in [k]$, $\Pr(\{R_i, R_j\}$ forms a wedge$) = 2W/m^2$. By linearity of expectation, the expected number of expected wedges is $\binom{k}{2}(2W/m^2)$. In particular, setting k to be c times a large constant multiple of $m/\sqrt{W}$ results in expectation of at least c^2 wedges. Even better is the fact that these wedges are uniformly (but not independently) distributed in the set of all wedges. Similarly, one can show that when k is $\Omega(m/\sqrt{T})$, one obtains many closed wedges. This is the heart of the argument in the analysis of the algorithm of [12].

While we do not wish to present all the technical details of the algorithm of [12] here, we do provide some high-level ideas from [12]. The algorithm maintains a reservoir of edges R_t as explained above. Further, it maintains a set C_t of wedges $\{e_a, e_b\}$ with $e_a, e_b \in \mathrm{R}_t$ such that (i) $\{e_a, e_b\}$ is a closed wedge in G_t and (ii) the closing edge appears after edges e_a and e_b in the stream. In other words, C_t are the wedges in R_t which can be *detected to be closed*. The algorithm outputs a random bit b_t whose expectation is close to κ_t.

For each edge e_t of the graph stream:

(a) Update reservoir R_t of k edges as described above.
(b) Let W_t be the set of wedges in R_t. Let N_t be the set of wedges in R_t which have e_t as their closing edge.

(c) Update C_t to $C_{t-1} \cap W_t \cup N_t$.
(d) If there are no wedges in R_t, output $b_t = 0$. Otherwise, sample a uniform wedge in R_t and output $b_t = 1$ if this wedge is in C_t and $b_t = 0$ otherwise.

The next theorem shows that the expectation of b_m is close to $\kappa/3$. Moreover, it shows that $|W_m|$ can be used to estimate W. Getting a good estimate on $\mathbf{E}[b_m]$ and W allows one to estimate T by multiplying the two estimates together. We note that while the theorem is stated for the final index $t = m$, it holds for any large enough t.

Theorem 1 ([12]) *Assume $W \geq m$ and fix $\beta \in (0,1)$. Suppose $k = |R_m|$ is $\Omega(m/(\beta^3\sqrt{T}))$. Set $\hat{W} = m^2|W_m|/(k(k-1))$. Then $|\kappa/3 - \mathbf{E}[b_m]| < \beta$ and $\Pr[|W - \hat{W}| < \beta W]$ are at least $1-\beta$.*

Related Work

The problem of counting triangles in graphs has been studied extensively in a variety of different settings: exact, sampling, streaming, and MapReduce. We refer the reader to references in [12] for a comprehensive list of these works. Here we focus on a narrow topic: single-pass streaming algorithms for estimating triangle counts. In particular, we do not discuss streaming algorithms that make multiple passes (e.g., [11,13]) or algorithms that compute triangles incident on every vertex (e.g., Becchetti et al. [4]).

Bar-Yossef et al. [3] were the first to study the problem of triangle counting in the streaming setting. Since then there have been a long line of work improving the guarantees of the algorithm in various ways [1, 2, 6, 12–15, 17]. Specifically, Buriol et al. [6] gave an $O(mn/T)$ space algorithm. The algorithm maintains samples of the form $((u,v),w)$ where (u,v) is a uniform edge in the stream and w is a uniform node label other than u and v. The algorithm checks for presence of edges (u,w) and (v,w) to detect triangle. They also give an implementation of their algorithm which is practical but relative error in triangle estimates can be high. The $O(m/\sqrt{T})$ algorithm described in this algorithm is from Jha et al. [12]. In parallel with [12], Pavan et al. [17] independently gave an $O(m\Delta/T)$ space algorithm where Δ is the maximum degree of the graph. Their algorithm is based on a sampling technique they introduced called *neighborhood sampling*. Neighborhood sampling maintains edge pair samples of the form (r_1, r_2). In each pair, edge r_1 is sampled uniformly from the edges observed so far. Edge r_2 is sampled uniformly from the set $N(r_1)$ where $N(r_1)$ consists of the edges adjacent to edge r_1 that appear after edge r_1 in the stream. When r_2 is nonempty, the pair (r_1, r_2) forms a wedge which can be monitored to see if it forms a triangle. Observe that a triangle formed by edges e_{t_1}, e_{t_2}, and e_{t_3} appearing in this order is detected as a closed triangle with probability $1/|N(e_{t_1})| \cdot m$. Accounting for this bias by keeping track of the quantity $N(r_1)$ for each sample (r_1, r_2), one gets an unbiased estimator for triangle counts. In a recent work, Ahmed et al. [1] give practical algorithms for triangle counting which seem to empirically improve on some of the previous results. In this work, the authors present a generic technique called *graph sample and hold* and use it for estimating triangle counts. At a high level, graph sample and hold associates a nonzero probability p_{e_t} to each edge e_t which corresponds to the probability with which edge e_t is independently sampled. Importantly, the probability p_{e_t} may depend on the graph sampled so far. But the actual probability with which the edge is sampled is *recorded*. Now estimates about the original graph can be obtained from the sampled graph using the selection estimator. For example, the number of triangles is estimated by summing $1/(p_{e_{t_1}} \cdot p_{e_{t_2}} \cdot p_{e_{t_3}})$ over all sampled triangles $\{e_{t_1}, e_{t_2}, e_{t_3}\}$. This can be seen as a generalization of neighborhood sampling.

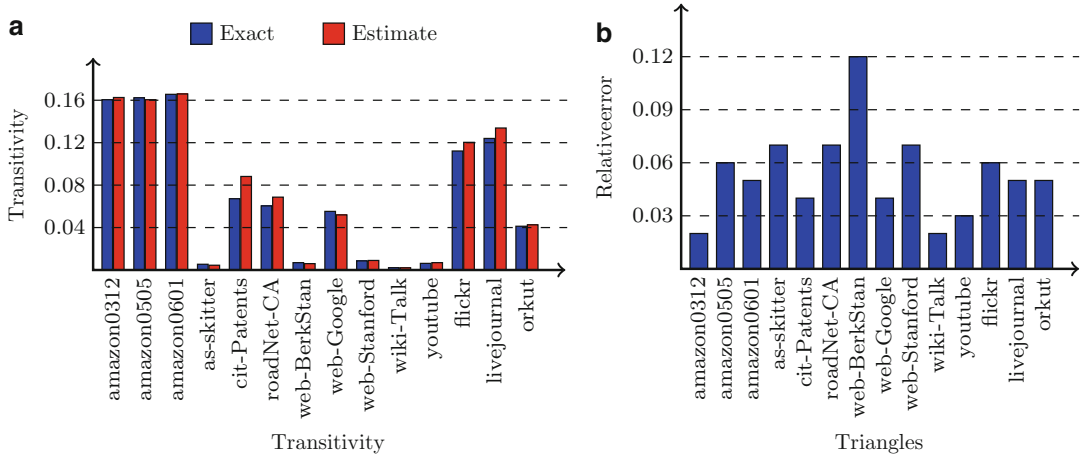
On the lower bound side, Braverman et al. [5] show that any single-pass streaming algorithm which gives a good *multiplicative* approximation of triangle counts must use $\Omega(m)$ bits of storage even if the input graph has $\Omega(n)$ triangles. This improves lower bounds from [3, 13]. For algorithms making a constant c number of passes, for every constant c, the lower bound is shown to be $\Omega(m/T)$ in the same work.

Applications

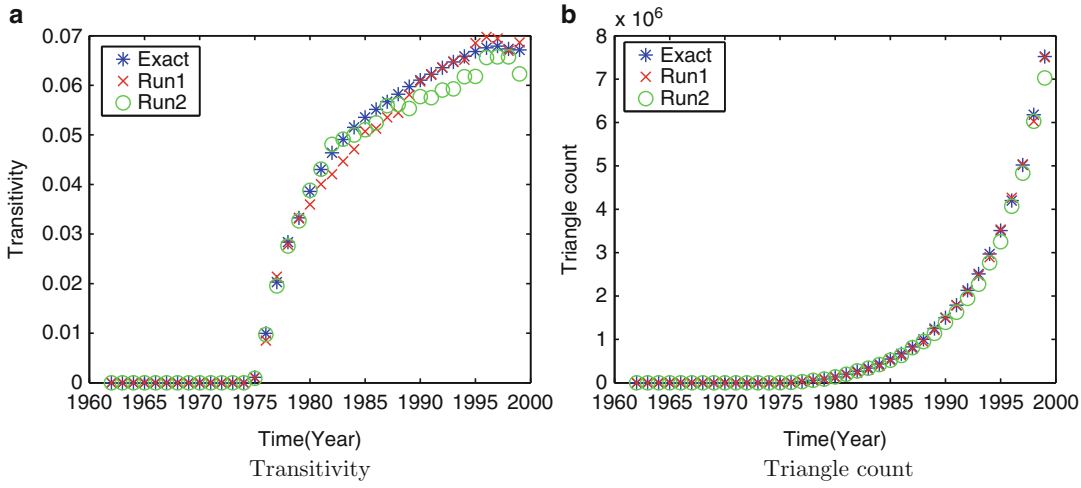
The number of triangles in a graph and the transitivity of a graph are important measures used widely in the study of networks across many different domains. For example, these measures appear in social science research [7, 8, 18, 22], in data mining applications such as spam detection and finding common topics on the World Wide Web [4, 9], and in bioinformatics for motif counting [16]. For a more detailed list of applications, we point the reader to introductory sections of references in the related work.

Open Problems

Give a tight lower bound on the space required by a single-pass streaming algorithm for estimating the number of triangles in graph stream with additive error. The lower bound for multiplicative approximation is known to be $\Omega(m)$ [3, 5].



Counting Triangles in Graph Streams, Fig. 1 Output of STREAMING-TRIANGLES algorithm of [12] on a variety of real datasets while storing at most 40 K edges. (**a**) Gives the estimated transitivity values alongside their exact values. (**b**) Gives the relative error of STREAMING-TRIANGLES's estimates of triangles T. Observe that the relative error for T is mostly below 8 % and often below 4 %



Counting Triangles in Graph Streams, Fig. 2 Real-time tracking of number of triangles and transitivities on cit-Patents (16 M edges), storing only 200 K edges (**a**) Transitivity. (**b**) Triangle count

Experimental Results

Figures 1 and 2 give a glimpse of experimental results from [12] on performance of algorithm STREAMING-TRIANGLES. Specifically, Fig. 1 shows the result of running the algorithm on a variety of graph datasets obtained from [20]. This includes run on graph datasets such as Orkut social network consisting of 200 M edges. The relative errors on κ and T are mostly less than 5 % (except for graphs with tiny κ). The storage used by the algorithm stated in terms of number of edges is at most 40 K. (The storage roughly corresponds to the size of edge reservoir used in the algorithm of [12] described in this article.)

An important aspect of the algorithm presented in [12] is that it can track the quantities κ_t and T_t for all values of t in real time. This is exhibited in Fig. 2.

Cross-References

- ▸ Data Stream Verification
- ▸ Graph Sketching
- ▸ Sliding Window Algorithms

Recommended Reading

1. Ahmed N, Duffield N, Neville J, Kompella R (2014) Graph sample and hold: a framework for big-graph analytics. In: The 20th ACM SIGKDD international conference on knowledge discovery and data mining (KDD '14), New York, 24–27 Aug 2014, pp 1446–1455. doi:10.1145/2623330.2623757. http://doi.acm.org/10.1145/2623330.2623757
2. Ahn KJ, Guha S, McGregor A (2012) Graph sketches: sparsification, spanners, and subgraphs. In: Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, PODS 2012, Scottsdale, 20–24 May 2012, pp 5–14. doi:10.1145/2213556.2213560. http://doi.acm.org/10.1145/2213556.2213560
3. Bar-Yossef Z, Kumar R, Sivakumar D (2002) Reductions in streaming algorithms, with an application to counting triangles in graphs. In: Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms, San Francisco, 6–8 Jan 2002, pp 623–632. http://dl.acm.org/citation.cfm?id=545381.545464
4. Becchetti L, Boldi P, Castillo C, Gionis A (2008) Efficient semi-streaming algorithms for local triangle counting in massive graphs. In: Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining, Las Vegas, 24–27 Aug 2008, pp 16–24. doi:10.1145/1401890.1401898. http://doi.acm.org/10.1145/1401890.1401898
5. Braverman V, Ostrovsky R, Vilenchik D (2013) How hard is counting triangles in the streaming model? In: Proceedings of the 40th international colloquium automata, languages, and programming (ICALP 2013) Part I, Riga, 8–12 July 2013, pp 244–254. doi:10.1007/978-3-642-39206-1_21. http://dx.doi.org/10.1007/978-3-642-39206-1_21
6. Buriol LS, Frahling G, Leonardi S, Marchetti-Spaccamela A, Sohler C (2006) Counting triangles in data streams. In: Proceedings of the twenty-fifth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, Chicago, 26–28 June 2006, pp 253–262. doi:10.1145/1142351.1142388. http://doi.acm.org/10.1145/1142351.1142388
7. Burt RS (2004) Structural holes and good ideas. Am J Soc 110(2):349–399. http://www.jstor.org/stable/10.1086/421787
8. Coleman JS (1988) Social capital in the creation of human capital. Am J Soc 94:S95–S120. http://www.jstor.org/stable/2780243
9. Eckmann JP, Moses E (2002) Curvature of co-links uncovers hidden thematic layers in the World Wide Web. Proc Nat Acad Sci (PNAS) 99(9):5825–5829. doi:10.1073/pnas.032093399
10. Feller W (1968) An Introduction to probability theory and applications, vol I, 3rd edn. John Wiley & Sons, New York
11. García-Soriano D, Kutzkov K (2014) Triangle counting in streamed graphs via small vertex covers. In: Proceedings of the 2014 SIAM international conference on data mining, Philadelphia, 24–26 Apr 2014, pp 352–360. doi:10.1137/1.9781611973440.40. http://dx.doi.org/10.1137/1.9781611973440.40
12. Jha M, Seshadhri C, Pinar A (2013) A space efficient streaming algorithm for triangle counting using the birthday paradox. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining (KDD '13), Chicago. ACM, New York, pp 589–597. doi:10.1145/2487575.2487678. http://doi.acm.org/10.1145/2487575.2487678
13. Jowhari H, Ghodsi M (2005) New streaming algorithms for counting triangles in graphs. In: Proceedings of the 11th annual international conference computing and combinatorics (COCOON 2005), Kunming, 16–29 Aug 2005, pp 710–716. doi:10.1007/11533719_72. http://dx.doi.org/10.1007/11533719_72
14. Kane DM, Mehlhorn K, Sauerwald T, Sun H (2012) Counting arbitrary subgraphs in data streams. In: Proceedings of the 39th international colloquium

automata, languages, and programming (ICALP 2012), Warwick, 9–13 July 2012, pp 598–609. doi:10.1007/978-3-642-31585-5_53. http://dx.doi.org/10.1007/978-3-642-31585-5_53
15. Kutzkov K, Pagh R (2014) Triangle counting in dynamic graph streams. In: Proceedings of the 14th Scandinavian symposium and workshops algorithm theory (SWAT 2014), Copenhagen, 2–4 July 2014, pp 306–318. doi:10.1007/978-3-319-08404-6_27. http://dx.doi.org/10.1007/978-3-319-08404-6_27
16. Milo R, Shen-Orr S, Itzkovitz S, Kashtan N, Chklovskii D, Alon U (2002) Network motifs: simple building blocks of complex networks. Science 298(5594):824–827
17. Pavan A, Tangwongsan K, Tirthapura S, Wu KL (2013) Counting and sampling triangles from a graph stream. PVLDB 6(14):1870–1881. http://www.vldb.org/pvldb/vol6/p1870-aduri.pdf
18. Portes A (1998) Social capital: its origins and applications in modern sociology. Ann Rev Soc 24(1):1–24. doi:10.1146/annurev.soc.24.1.1
19. Seshadhri C, Pinar A, Kolda TG (2014) Wedge sampling for computing clustering coecients and triangle counts on large graphs. Stat Anal Data Mining 7(4):294–307. doi:10.1002/sam.11224. http://dx.doi.org/10.1002/sam.11224
20. SNAP (2013) Stanford network analysis project. Available at http://snap.stanford.edu/
21. Vitter J (1985) Random sampling with a reservoir. ACM Trans Math Softw 11(1):37–57. doi:10.1145/3147.3165. http://doi.acm.org/10.1145/3147.3165
22. Welles BF, Devender AV, Contractor N (2010) Is a friend a friend? Investigating the structure of friendship networks in virtual worlds. In: Proceedings of the 28th international conference on human factors in computing systems (CHI 2010), extended abstracts volume, Atlanta, 10–15 April 2010, pp 4027–4032. doi:10.1145/1753846.1754097. http://doi.acm.org/10.1145/1753846.1754097

Count-Min Sketch

Graham Cormode
Department of Computer Science, University of Warwick, Coventry, UK

Keywords

Approximate counting; Frequent items; Sketch; Streaming algorithms

Years and Authors of Summarized Original Work

2004; Cormode, Muthukrishnan

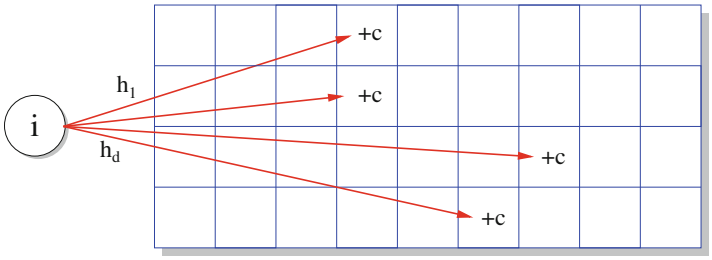
Problem Definition

The problem of *sketching* a large mathematical object is to produce a compact data structure that approximately represents it. The Count-Min (CM) sketch is an example of a sketch that allows a number of related quantities to be estimated with accuracy guarantees, including point queries and dot product queries. Such queries are at the core of many computations, so the structure can be used in order to answer a variety of other queries, such as frequent items (heavy hitters), quantile finding, join size estimation, and more. Since the sketch can process updates in the form of additions or subtractions to dimensions of the vector (which may correspond to insertions or deletions or other transactions), it is capable of working over streams of updates, at high rates.

The data structure maintains the linear projection of the vector with a number of other random vectors. These vectors are defined implicitly by simple hash functions. Increasing the range of the hash functions increases the accuracy of the summary, and increasing the number of hash functions decreases the probability of a bad estimate. These trade-offs are quantified precisely below. Because of this linearity, CM sketches can be scaled, added, and subtracted, to produce summaries of the corresponding scaled and combined vectors.

Key Results

The Count-Min sketch was first proposed in 2003 [4], following several other sketch techniques, such as the Count sketch [2] and the AMS sketch [1]. The sketch is similar to a counting Bloom filter or multistage filter [7].



Count-Min Sketch, Fig. 1 Each item i is mapped to one cell in each row of the array of counts: when an update of c to item i_t arrives, c_t is added to each of these cells

Data Structure Description

The CM sketch is simply an array of counters of width w and depth d, $CM[1,1] \ldots CM[d,w]$. Each entry of the array is initially zero. Additionally, d hash functions

$$h_1 \ldots h_d : \{1 \ldots n\} \rightarrow \{1 \ldots w\}$$

are chosen uniformly at random from a pairwise-independent family. Once w and d are chosen, the space required is fixed: the data structure is represented by wd counters and d hash functions (which can each be represented in $O(1)$ machine words [12]).

Update Procedure

A vector $\boldsymbol{a}$ of dimension n is described incrementally. Initially, $\boldsymbol{a}(0)$ is the zero vector, $\mathbf{0}$, so $a_i(0)$ is 0 for all i. Its state at time t is denoted $\boldsymbol{a}(t) = [a_1(t), \ldots a_i(t), \ldots a_n(t)]$. Updates to individual entries of the vector are presented as a stream of pairs. The tth update is (i_t, c_t), meaning that

$$\begin{aligned} a_{i_t}(t) &= a_{i_t}(t-1) + c_t \\ a_{i'}(t) &= a_{i'}(t-1) \qquad i' \neq i_t. \end{aligned}$$

For convenience, the subscript t is dropped, and the current state of the vector simply referred to as $\boldsymbol{a}$. For simplicity of description, it is assumed here that although values of a_i increase and decrease with updates, each $a_i \geq 0$. However, the sketch can also be applied to the case where a_is can be less than zero with some increase in costs [4].

When an update (i_t, c_t) arrives, c_t is added to one count in each row of the Count-Min sketch; the counter is determined by h_j. Formally, given (i_t, c_t), the following modifications are performed:

$$\forall 1 \leq j \leq d : CM[j, h_j(i_t)] \leftarrow CM[j, h_j(i_t)] + c_t.$$

This procedure is illustrated in Fig. 1. Because computing each hash function takes constant time, the total time to perform an update is $O(d)$, independent of w. Since d is typically small in practice (often less than 10), updates can be processed at high speed.

Point Queries

A *point query* is to estimate the value of an entry in the vector a_i. Given a query point i, an estimate is found from the sketch as $\hat{a}_i = \min_{1 \leq j \leq d} CM[j, h_j(i)]$. The approximation guarantee is that if $w = \lceil \frac{e}{\varepsilon} \rceil$ and $d = \lceil \ln \frac{1}{\delta} \rceil$, the estimate $\hat{a}_i$ obeys $a_i \leq \hat{a}_i$; and, with probability at least $1 - \delta$,

$$\hat{a}_i \leq a_i + \varepsilon \|\boldsymbol{a}\|_1.$$

Here, $\|\boldsymbol{a}\|_1$ is the L_1 norm of $\boldsymbol{a}$, i.e., the sum of the (absolute) values. The proof follows by using the Markov inequality to limit the error in each row and then using the independence of the hash functions to amplify the success probability.

This analysis makes no assumption about the distribution of values in $\boldsymbol{a}$. In many applications, there are Zipfian, or power law, distributions of item frequencies. Here, the (relative) frequency of the ith most frequent item is proportional to i^{-z}, for some parameter z, where z is typically in the range 1–3. Here, the skew in the distribution can be used to show a stronger space/accuracy trade-off: for a Zipf distribution with parameter z, the space required to answer point queries with

error $\varepsilon\|\boldsymbol{a}\|_1$ with probability at least $1-\delta$ is given by $O(\varepsilon^{-\min\{1,1/z\}} \ln 1/\delta)$ [5].

Range, Heavy Hitter, and Quantile Queries

A *range query* is to estimate $\sum_{i=l}^{r} a_i$ for a range $[l \ldots r]$. For small ranges, the range sum can be estimated as a sum of point queries; however, as the range grows, the error in this approach also grows linearly. Instead, $\log n$ sketches can be kept, each of which summarizes a derived vector a^k where

$$a^k[j] = \sum_{i=j2^k}^{(j+1)2^k-1} a_i$$

for $k = 1 \ldots \log n$. A range of the form $j2^k \ldots (j+1)2^k - 1$ is called a *dyadic range*, and any arbitrary range $[l \ldots r]$ can be partitioned into at most $2 \log n$ dyadic ranges. With appropriate rescaling of accuracy bounds, it follows that Count-Min sketches can be used to find an estimate $\hat{r}$ for a range query on $l \ldots r$ such that

$$\hat{r} - \varepsilon\|\boldsymbol{a}\|_1 \leq \sum_{i=l}^{r} a_i \leq \hat{r}.$$

The right inequality holds with certainty, and the left inequality holds with probability at least $1-\delta$. The total space required is $O(\frac{\log^2 n}{\varepsilon} \log \frac{1}{\delta})$ [4]. The closely related *ϕ-quantile query* is to find a point j such that

$$\sum_{i=1}^{j} a_i \leq \phi\|\boldsymbol{a}\|_1 \leq \sum_{i=1}^{j+1} a_i.$$

Range queries can be used to (binary) search for a j which satisfies this requirement approximately (i.e., tolerates up to $\varepsilon\|\boldsymbol{a}\|_1$ error in the above expression) given ϕ. The overall cost is space that depends on $1/\varepsilon$, with further log factors for the rescaling necessary to give the overall guarantee [4]. The time for each insert or delete operation and the time to find any quantile are logarithmic in n, the size of the domain.

Heavy hitters are those points i such that $a_i \geq \phi\|\boldsymbol{a}\|_1$ for some specified ϕ. The range query primitive based on Count-Min sketches can again be used to find heavy hitters, by recursively splitting dyadic ranges into two and querying each half to see if the range is still heavy, until a range of a single, heavy item is found. The cost of this is similar to that for quantiles, with space dependent on $1/\varepsilon$ and $\log n$. The time to update the data structure and to find approximate heavy hitters is also logarithmic in n. The guarantee is that every item with frequency at least $(\phi+\varepsilon)\|\boldsymbol{a}\|_1$ is output, and with probability $1-\delta$ no item whose frequency is less than $\phi\|\boldsymbol{a}\|_1$ is output.

Inner Product Queries

The Count-Min sketch can also be used to estimate the inner product between two vectors. The inner product $\boldsymbol{a}\cdot\boldsymbol{b}$ can be estimated by treating the Count-Min sketch as a collection of d vectors of length w and finding the minimum inner product between corresponding rows of sketches of the two vectors. With probability $1-\delta$, this estimate is at most an additive quantity $\varepsilon\|\boldsymbol{a}\|_1\|\boldsymbol{b}\|_1$ above the true value of $\boldsymbol{a}\cdot\boldsymbol{b}$. This is to be compared with AMS sketches which guarantee $\varepsilon\|\boldsymbol{a}\|_2\|\boldsymbol{b}\|_2$ additive error but require space proportional to $\frac{1}{\varepsilon^2}$ to make this guarantee.

Conservative Update

If only positive updates arrive, then the "conservative update" process (due to Estan and Varghese [7]) can be used. For an update (i,c), $\hat{a}_i$ is computed, and the counts are modified according to $\forall 1 \leq j \leq d : CM[j,h_j(i)] \leftarrow \max(CM[j,h_j(i)], \hat{a}_i + c)$. This procedure still ensures for point queries that $a_i \leq \hat{a}_i$, and that the error is no worse than in the normal update procedure; it has been observed that conservative update can improve accuracy "up to an order of magnitude" [7]. However, deletions or negative updates can no longer be processed, and the new update procedure is slower than the original one.

Applications

The Count-Min sketch has found a number of applications.

- Indyk [9] used the Count-Min sketch to estimate the residual mass after removing a set of items, that is, given a (small) set of indices I, to estimate $\sum_{i \notin I} a_i$. This supports clustering over streaming data.
- The *entropy* of a data stream is a function of the relative frequencies of each item or character within the stream. Using Count-Min sketches within a larger data structure based on additional hashing techniques, B. Lakshminath and Ganguly [8] showed how to estimate this entropy to within relative error.
- Sarlós et al. [14] gave approximate algorithms for personalized page rank computations which make use of Count-Min sketches to compactly represent web-sized graphs.
- In describing a system for building selectivity estimates for complex queries, Spiegel and Polyzotis [15] use Count-Min sketches in order to allow clustering over a high-dimensional space.
- Sketches that reduce the amount of information stored seem like a natural candidate to preserve privacy of information. However, proving privacy requires more care. Roughan and Zhang use the Count-Min sketch to allow private computation of a sketch of a vector [13]. Dwork et al. show that the Count-Min sketch can be made *pan-private*, meaning that information about individuals contributing to the data structure is held private.

Experimental Results

There have been a number of experimental studies of COUNT-MIN and related algorithms for a variety of computing models. These have shown that the algorithm is accurate and fast to execute [3, 11]. Implementations on desktop machines achieve many millions of updates per second, primarily limited by IO throughput. Other implementations have incorporated Count-Min sketch into high-speed streaming systems such as Gigascope [6] and tuned it to process packet streams of multi-gigabit speeds. Lai and Byrd report on an implementation of Count-Min sketches on a low-power stream processor [10] capable of processing 40 byte packets at a throughput rate of up to 13 Gbps. This is equivalent to about 44 million updates per second.

URLs to Code and Data Sets

Sample implementations are widely available in a variety of languages.

C code is given by the MassDal code bank: http://www.cs.rutgers.edu/~muthu/massdal-code-index.html.

C++ code by Marios Hadjieleftheriou is available from http://hadjieleftheriou.com/sketches/index.html.

The MADlib project has SQL implementations for Postgres/Greenplum http://madlib.net/.

OCaml implementation is available via https://github.com/ezyang/ocaml-cminsketch.

Cross-Reference

▸ AMS Sketch

Recommended Reading

1. Alon N, Matias Y, Szegedy M (1996) The space complexity of approximating the frequency moments. In: ACM symposium on theory of computing, Philadelphia, pp 20–29
2. Charikar M, Chen K, Farach-Colton M (2002) Finding frequent items in data streams. In: Procedings of the international colloquium on automata, languages and programming (ICALP), Málaga
3. Cormode G, Hadjieleftheriou M (2009) Finding the frequent items in streams of data. Commun ACM 52(10):97–105
4. Cormode G, Muthukrishnan S (2005) An improved data stream summary: the Count-Min sketch and its applications. J Algorithms 55(1):58–75
5. Cormode G, Muthukrishnan S (2005) Summarizing and mining skewed data streams. In: SIAM conference on data mining, Newport Beach
6. Cormode G, Korn F, Muthukrishnan S, Johnson T, Spatscheck O, Srivastava D (2004) Holistic UDAFs at streaming speeds. In: ACM SIGMOD international conference on management of data, Paris, pp 35–46
7. Estan C, Varghese G (2002) New directions in traffic measurement and accounting. In: Proceedings of

ACM SIGCOMM, computer communication review, vol 32, 4, Pittsburgh, PA, pp 323–338
8. Ganguly S, Lakshminath B (2006) Estimating entropy over data streams. In: European symposium on algorithms (ESA), Zurich
9. Indyk P (2003) Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In: ACM-SIAM symposium on discrete algorithms, Baltimore
10. Lai YK, Byrd GT (2006) High-throughput sketch update on a low-power stream processor. In: Proceedings of the ACM/IEEE symposium on architecture for networking and communications systems, San Jose
11. Manerikar N, Palpanas T (2009) Frequent items in streaming data: an experimental evaluation of the state-of-the-art. Data Knowl Eng 68(4): 415–430
12. Motwani R, Raghavan P (1995) Randomized algorithms. Cambridge University Press, Cambridge/New York
13. Roughan M, Zhang Y (2006) Secure distributed data mining and its application in large-scale network measurements. In: ACM SIGCOMM computer communication review (CCR), Pisa
14. Sarlós T, Benzúr A, Csalogány K, Fogaras D, Rácz B (2006) To randomize or not to randomize: space optimal summaries for hyperlink analysis. In: International conference on World Wide Web (WWW), Edinburgh
15. Spiegel J, Polyzotis N (2006) Graph-based synopses for relational selectivity estimation. In: ACM SIGMOD international conference on management of data, Chicago

CPU Time Pricing

Li-Sha Huang
Department of Compurter Science and Technology, Tsinghua University, Beijing, China

Keywords

Competitive auction; Market equilibrium; Resource scheduling

Years and Authors of Summarized Original Work

2005; Deng, Huang, Li

Problem Definition

This problem is concerned with a Walrasian equilibrium model to determine the prices of CPU time. In a market model of a CPU job scheduling problem, the owner of the CPU processing time sells time slots to customers and the prices of each time slot depends on the seller's strategy and the customers' bids (valuation functions). In a Walrasian equilibrium, the market is clear and each customer is most satisfied according to its valuation function and current prices. The work of Deng, Huang, and Li [1] establishes the existence conditions of Walrasion equilibrium, and obtains complexity results to determine the existence of equilibrium. It also discusses the issues of excessive supply of CPU time and price dynamics.

Notations

Consider a combinatorial auction (Ω, I, V):

- Commodities: The seller sells m kinds of indivisible commodities. Let $\Omega = \{\omega_1 \times \delta_1, \ldots, \omega_m \times \delta_m\}$ denote the set of commodities, where δ_j is the available quantity of the item ω_j.
- Agents: There are n agents in the market acting as buyers, denoted by $I = \{1, 2, \ldots, n\}$.
- Valuation functions: Each buyer $i \in I$ has a valuation function $v_i : 2^{\Omega} \to \mathbb{R}^+$ to submit the maximum amount of money he is willing to pay for a certain bundle of items. Let $V = \{v_1, v_2, \ldots, v_n\}$.

An XOR combination of two valuation functions v_1 and v_2 is defined by:

$$(v_1 \text{ XOR } v_2)(S) = \max\{v_1(S), v_2(S)\}$$

An *atomic bid* is a valuation function v denoted by a pair (S, q), where $S \subset \Omega$ and $q \in \mathbb{R}^+$:

$$v(T) = \begin{cases} q\,, & \text{if } S \subset T \\ 0\,, & \text{otherwise} \end{cases}$$

Any valuation function v_i can be expressed by an XOR combination of atomic bids,

$$v_i = (S_{i1}, q_{i1}) \text{XOR } (S_{i2}, q_{i2}) \ldots \text{ XOR } (S_{in}, q_{in})$$

Given (Ω, I, V) as the input, the seller will determine an *allocation* and a *price vector* as the output:

- An *allocation* $X = \{X_0, X_1, X_2, \ldots, X_n\}$ is a partition of Ω, in which X_i is the bundle of commodities assigned to buyer i and X_0 is the set of unallocated commodities.
- A *price* vector p is a non-negative vector in $\mathbb{R}^m$, whose jth entry is the price of good $\omega_j \in \Omega$.

For any subset $T = \{\omega_1 \times \sigma_1, \ldots, \omega_m \times \sigma_m\} \subset \Omega$, define $p(T)$ by $p(T) = \sum_{j=1}^{m} \sigma_j p_j$. If buyer i is assigned to a bundle X_i, his *utility* is $u_i(X_i, p) = v_i(X_i) - p(X_i)$.

Definition A *Walrasian equilibrium* for a combinatorial auction (Ω, I, V) is a tuple (X, p), where $X = \{X_0, X_1, \ldots, X_n\}$ is an allocation and p is a price vector, satisfying that:

(1) $p(X_0) = 0$;

(2) $u_i(X_i, p) \geq u_i(B, p), \quad \forall B \subset \Omega, \quad \forall 1 \leq i \leq n$

Such a price vector is also called a market clearing price, or Walrasian price, or equilibrium price.

The CPU Job-Scheduling Problem

There are two types of players in a market-driven CPU resource allocation model: a resource provider and n consumers. The provider sells to the consumers CPU time slots and the consumers each have a job that requires a fixed number of CPU time, and its valuation function depends on the time slots assigned to the job, usually the last assigned CPU time slot. Assume that all jobs are released at time $t = 0$ and the ith job needs s_i time units. The jobs are interruptible without preemption cost, as is often modeled for CPU jobs.

Translating into the language of combinatorial auctions, there are m commodities (time units), $\Omega = \{\omega_1, \ldots, \omega_m\}$, and n buyers (jobs), $I = \{1, 2, \ldots, n\}$, in the market. Each buyer has a valuation function v_i, which only depends on the completion time. Moreover, if not explicitly mentioned, every job's valuation function is non-increasing w.r.t. the completion time.

Key Results

Consider the following linear programming problem:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{k_i} q_{ij} x_{ij}$$

$$\text{s.t.} \sum_{i,j \mid \omega_k \in S_{ij}} x_{ij} \leq \delta_k \,, \quad \forall \omega_k \in \Omega$$

$$\sum_{j=1}^{r_i} x_{ij} \leq 1 \,, \quad \forall 1 \leq i \leq n$$

$$0 \leq x_{ij} \leq 1 \,, \quad \forall i, j$$

Denote the problem by **LPR** and its integer restriction by **IP**. The following theorem shows that a non-zero gap between the integer programming problem **IP** and its linear relaxation implies the non-existence of the Walrasian equilibrium.

Theorem 1 *In a combinatorial auction, the Walrasian equilibrium exists if and only if the optimum of **IP** equals the optimum of **LPR**. The size of the LP problem is linear to the total number of XOR bids.*

Theorem 2 *Determination of the existence of Walrasian equilibrium in a CPU job scheduling problem is strong NP-hard.*

Now consider a job scheduling problem in which the customers' valuation functions are all linear. Assume n jobs are released at the time $t = 0$ for a single machine, the jth job's time span is $s_j \in \mathbb{N}^+$ and weight $w_j \geq 0$. The goal of the scheduling is to minimize the weighted completion time: $\sum_{i=1}^{n} w_i t_i$, where t_i is the completion time of job i. Such a problem is called an MWCT (Minimal Weighted Completion Time) problem.

Theorem 3 *In a single-machine MWCT job scheduling problem, Walrasian equilibrium always exists when $m \geq EM + \Delta$, where m is the total number of processor time, $EM = \sum_{i=1}^{n} s_i$ and $\Delta = \max_k \{s_k\}$. The equilibrium can be computed in polynomial time.*

The following theorem shows the existence of a non-increasing price sequence if Walrasian equilibrium exists.

Theorem 4 *If there exists a Walrasian equilibrium in a job scheduling problem, it can be adjusted to an equilibrium with consecutive allocation and a non-increasing equilibrium price vector.*

Applications

Information technology has changed people's lifestyles with the creation of many digital goods, such as word processing software, computer games, search engines, and online communities. Such a new economy has already demanded many theoretical tools (new and old, of economics and other related disciplines) be applied to their development and production, marketing, and pricing. The lack of a full understanding of the new economy is mainly due to the fact that digital goods can often be re-produced at no additional cost, though multi-fold other factors could also be part of the difficulty. The work of Deng, Huang, and Li [1] focuses on CPU time as a product for sale in the market, through the Walrasian pricing model in economics. CPU time as a commercial product is extensively studied in grid computing. Singling out CPU time pricing will help us to set aside other complicated issues caused by secondary factors, and a complete understanding of this special digital product (or service) may shed some light on the study of other goods in the digital economy.

The utilization of CPU time by multiple customers has been a crucial issue in the development of operating system concept. The rise of grid computing proposes to fully utilize computational resources, e.g., CPU time, disk space, bandwidth. Market-oriented schemes have been proposed for efficient allocation of computational grid recourses, by [2, 5]. Later, various practical and simulation systems have emerged in grid resource management. Besides the resource allocation in grids, an economic mechanism has also been introduced to TCP congestion control problems, see Kelly [4].

Cross-References

- Adwords Pricing
- Competitive Auction
- Incentive Compatible Selection
- Price of Anarchy

Recommended Reading

1. Deng X, Huang L-S, Li M (2007) On Walrasian price of CPU time. In: Proceedings of COCOON'05, Knming, 16–19 Aug 2005, pp 586–595. Algorithmica 48(2):159–172
2. Ferguson D, Yemini Y, Nikolaou C (1988) Microeconomic algorithms for load balancing in distributed computer systems. In: Proceedings of DCS'88, San Jose, 13–17 June 1988, pp 419–499
3. Goldberg AV, Hartline JD, Wright A (2001) Competitive auctions and digital goods. In: Proceedings of SODA'01, Washington, DC, 7–9 Jan 2001, pp 735–744
4. Kelly FP (1997) Charging and rate control for elastic traffic. Eur Trans Telecommun 8:33–37
5. Kurose JF, Simha R (1989) A microeconomic approach to optimal resource allocation in distributed computer systems. IEEE Trans Comput 38(5):705–717
6. Nisan N (2000) Bidding and allocation in combinatorial auctions. In: Proceedings of EC'00, Minneapolis, 17–20 Oct 2000, pp 1–12

Critical Range for Wireless Networks

Chih-Wei Yi
Department of Computer Science, National Chiao Tung University, Hsinchu City, Taiwan

Keywords

Connectivity; Delaunay triangulations; Gabriel graphs; Greedy forward routing; Isolated nodes; Monotonic properties; Random geometric graphs

Years and Authors of Summarized Original Work

2004; Wan, Yi

Problem Definition

Given a point set V, a graph of the vertex set V in which two vertices have an edge if and only if the distance between them is at most r for some positive real number r is called a r-disk graph over the vertex set V and denoted by $G_r(V)$. If $r_1 \leq r_2$, obviously $G_{r_1}(V) \subseteq G_{r_2}(V)$. A graph property is monotonic (increasing) if a graph is with the property, then every supergraph with the same vertex set also has the property. The critical-range problem (or critical-radius problem) is concerned with the minimal range r such that $G_r(V)$ is with some monotonic property. For example, graph connectivity is monotonic and crucial to many applications. It is interesting to know whether $G_r(V)$ is connected or not. Let $\rho_{\text{con}}(V)$ denote the minimal range r such that $G_r(V)$ is connected. Then, $G_r(V)$ is connected if $r \geq \rho_{\text{con}}(V)$, and otherwise not connected. Here $\rho_{\text{con}}(V)$ is called the critical range for connectivity of V. Formally, the critical-range problem is defined as follows.

Definition 1 The critical range for a monotonic graph property π over a point set V, denoted by $\rho_\pi(V)$, is the smallest range r such that $G_r(V)$ has property π.

From another aspect, for a given geometric property, a corresponding geometric structure is usually embedded. In many cases, the critical-range problem for graph properties is related or equivalent to the longest-edge problem of corresponding geometric structures. For example, if $G_r(V)$ is connected, it contains a Euclidean minimal spanning tree (EMST), and $\rho_{\text{con}}(V)$ is equal to the largest edge length of the EMST. So the critical range for connectivity problem is equivalent to the longest edge of the EMST problem, and the critical range for connectivity is the smallest r such that $G_r(V)$ contains the EMST.

In most cases, given an instance, the critical range can be calculated by polynomial time algorithms. So it is not a hard problem to decide the critical range. Researchers are interested in the probabilistic analysis of the critical range, especially asymptotic behaviors of r-disk graphs over random point sets. Random geometric graphs [8] is a general term for the theory about r-disk graphs over random point sets.

Key Results

In the following, problems are discussed in a 2D plane. Let $X_1, X_2, \cdots$ be independent and uniformly distributed random points on a bounded region A. Given a positive integer n, the point process $\{X_1, X_2, \ldots, X_n\}$ is referred to as the uniform n-point process on A, and is denoted by $\mathrm{X}_n(A)$. Given a positive number λ, let $Po(\lambda)$ be a Poisson random variable with parameter λ, independent of $\{X_1, X_2, \ldots\}$. Then the point process $\{X_1, X_2, \ldots, X_{Po(n)}\}$ is referred to as the Poisson point process with mean n on A, and is denoted by $\mathrm{P}_n(A)$. A is called a deployment region. An event is said to be asymptotic almost sure if it occurs with a probability that converges to 1 as $n \to \infty$.

In a graph, a node is "isolated" if it has no neighbor. If a graph is connected, there exists no isolated node in the graph. The asymptotic

distribution of the number of isolated nodes is given by the following theorem [2, 6, 14].

Theorem 1 *Let* $r_n = \sqrt{\frac{\ln n + \xi}{\pi n}}$ *and* Ω *be a unit-area disk or square. The number of isolated nodes in* $G_r(\mathrm{X}_n(\Omega))$ *or* $G_r(\mathrm{P}_n(\Omega))$ *is asymptotically Poisson with mean* $\mathrm{e}^{-\xi}$.

According to the theorem, the probability of the event that there is no isolated node is asymptotically equal to $\exp\left(-\mathrm{e}^{-\xi}\right)$. In the theory of random geometric graphs, if a graph has no isolated node, it is almost surely connected. Thus, the next theorem follows [6, 8, 9].

Theorem 2 *Let* $r_n = \sqrt{\frac{\ln n + \xi}{\pi n}}$ *and* Ω *be a unit-area disk or square. Then,*
$\Pr[G_r(\mathrm{X}_n(\Omega)) \text{ is connected}] \to \exp\left(-\mathrm{e}^{-\xi}\right)$, and

$$\Pr[G_r(\mathrm{P}_n(\Omega)) \text{ is connected}] \to \exp\left(-\mathrm{e}^{-\xi}\right).$$

In wireless sensor networks, the deployment region is k-covered if every point in the deployment region is within the coverage ranges of at least k sensors (vertices). Assume the coverage ranges are disks of radius r centered at the vertices. Let k be a fixed non-negative integer, and Ω be the unit-area square or disk centered at the origin $\mathbf{o}$. For any real number t, let $t\Omega$ denote the set $\{tx : x \in \Omega\}$, i. e., the square or disk of area t^2 centered at the origin. Let $C_{n,r}$ (respectively, $C'_{n,r}$) denote the event that Ω is $(k+1)$-covered by the (open or closed) disks of radius r centered at the points in $\mathrm{P}_n(\Omega)$ (respectively, $\mathrm{X}_n(\Omega)$). Let $K_{s,n}$ (respectively, $K'_{s,n}$) denote the event that $\sqrt{s}\Omega$ is $(k+1)$-covered by the unit-area (closed or open) disks centered at the points in $\mathrm{P}_n(\sqrt{s}\Omega)$ (respectively, $\mathrm{X}_n(\sqrt{s}\Omega)$). To simplify the presentation, let η denote the peripheral of Ω, which is equal to 4 (respectively, $2\sqrt{\pi}$) if Ω is a square (respectively, disk). For any $\xi \in \mathbb{R}$, let

$$\alpha(\xi) = \begin{cases} \dfrac{\left(\frac{\sqrt{\pi}\eta}{2} + \mathrm{e}^{-\frac{\xi}{2}}\right)^2}{16\left(2\sqrt{\pi}\eta + \mathrm{e}^{-\frac{\xi}{2}}\right)} \mathrm{e}^{-\frac{\xi}{2}}, & \text{if } k = 0; \\ \dfrac{\sqrt{\pi}\eta}{2^{k+6}(k+2)!} \mathrm{e}^{-\frac{\xi}{2}}, & \text{if } k \geq 1. \end{cases}$$

and

$$\beta(\xi) = \begin{cases} 4\mathrm{e}^{-\xi} + 2\left(\sqrt{\pi} + \frac{1}{\sqrt{\pi}}\right)\eta \mathrm{e}^{-\frac{\xi}{2}}, & \text{if } k = 0; \\ \dfrac{\sqrt{\pi} + \frac{1}{\sqrt{\pi}}}{2^{k-1}k!}\eta \mathrm{e}^{-\frac{\xi}{2}}, & \text{if } k \geq 1. \end{cases}$$

The asymptotics of $\Pr[C_{n,r}]$ and $\Pr\left[C'_{n,r}\right]$ as n approaches infinity, and the asymptotics of $\Pr[K_{s,n}]$ and $\Pr\left[K'_{s,n}\right]$ as s approaches infinity are given in the following two theorems [4, 10, 16].

Theorem 3 *Let* $r_n = \sqrt{\frac{\ln n + (2k+1)\ln\ln n + \xi_n}{\pi n}}$. *If* $\lim_{n\to\infty} \xi_n = \xi$ *for some* $\xi \in \mathbb{R}$, *then*

$$1 - \beta(\xi) \leq \lim_{n\to\infty} \Pr[C_{n,r_n}] \leq \frac{1}{1 + \alpha(\xi)}, \text{ and}$$

$$1 - \beta(\xi) \leq \lim_{n\to\infty} \Pr\left[C'_{n,r_n}\right] \leq \frac{1}{1 + \alpha(\xi)}.$$

If $\lim_{n\to\infty} \xi_n = \infty$, *then*

$$\lim_{n\to\infty} \Pr[C_{n,r_n}] = \lim_{n\to\infty} \Pr\left[C'_{n,r_n}\right] = 1.$$

If $\lim_{n\to\infty} \xi_n = -\infty$, *then*

$$\lim_{n\to\infty} \Pr[C_{n,r_n}] = \lim_{n\to\infty} \Pr\left[C'_{n,r_n}\right] = 0.$$

Theorem 4 *Let* $\mu(s) = \ln s + 2(k+1)\ln\ln s + \xi(s)$. *If* $\lim_{s\to\infty} \xi(s) = \xi$ *for some* $\xi \in \mathbb{R}$, *then*

$$1 - \beta(\xi) \leq \lim_{s\to\infty} \Pr\left[K_{s,\mu(s)s}\right] \leq \frac{1}{1 + \alpha(\xi)}, \text{ and}$$

$$1 - \beta(\xi) \leq \lim_{s\to\infty} \Pr\left[K'_{s,\mu(s)s}\right] \leq \frac{1}{1 + \alpha(\xi)}.$$

If $\lim_{s\to\infty} \xi(s) = \infty$, *then*

$$\lim_{s\to\infty} \Pr\left[K_{s,\mu(s)s}\right] = \lim_{s\to\infty} \Pr\left[K'_{s,\mu(s)s}\right] = 1.$$

If $\lim_{s\to\infty} \xi(s) = -\infty$, *then*

$$\lim_{s\to\infty} \Pr\left[K_{s,\mu(s)s}\right] = \lim_{s\to\infty} \Pr\left[K'_{s,\mu(s)s}\right] = 0.$$

In Gabriel graphs (GG), two nodes have an edge if and only if there is no other node in the

disk using the segment of these two nodes as its diameter. If V is a point set and l is a positive real number, we use $\rho_{\mathrm{GG}}(V)$ to denote the largest edge length of the GG over V, and $N(V,l)$ denotes the number of GG edges over V whose length is at least l. Wan and Yi (2007) [11] gave the following theorem.

Theorem 5 *Let Ω be a unit-area disk. For any constant ξ, $N\left(\mathrm{P}_n(\Omega), 2\sqrt{\frac{\ln n+\xi}{\pi n}}\right)$ is asymptotically Poisson with mean $2\mathrm{e}^{-\xi}$, and*

$$\lim_{n\to\infty}\Pr\left[\rho_{\mathrm{GG}}\left(\mathrm{P}_n(\Omega)\right) < 2\sqrt{\frac{\ln n+\xi}{\pi n}}\right] = \exp\left(-2\mathrm{e}^{-\xi}\right).$$

Let $\rho_{\mathrm{Del}}(V)$ denote the largest edge length of the Delaunay triangulation over a point set V. The following theorem is given by Kozma et al. [3].

Theorem 6 *Let Ω be a unit-area disk. Then,*

$$\rho_{\mathrm{Del}}\left(\mathrm{X}_n(\Omega)\right) = O\left(\sqrt[3]{\frac{\ln n}{n}}\right).$$

In wireless networks with greedy forward routing (GFR), each node discards a packet if none of its neighbors is closer to the destination of the packet than itself, or otherwise forwards the packet to the neighbor that is the closest to the destination. Since each node only needs to maintain the locations of its one-hop neighbors and each packet should contain the location of the destination node, GFR can be implemented in a localized and memoryless manner. Because of the existence of local minima where none of the neighbors is closer to the destination than the current node, a packet may be discarded before it reaches its destination. To ensure that every packet can reach its destination, all nodes should have sufficiently large transmission radii to avoid the existence of local minima. Applying the r-disk model, we assume every node has the same transmission radius r, and each pair of nodes with distance at most r has a link. For a point set V, the *critical transmission radius* for GFR is given by

$$\rho_{\mathrm{GFR}}(V) = \max_{(u,v)\in V^2, u\neq v}\left(\min_{\|w-v\|<\|u-v\|}\|w-u\|\right).$$

In the definition, (u,v) is a source–destination pair and w is a node that is closer to v than u. If every node is with a transmission radius not less than $\rho_{\mathrm{GFR}}(V)$, GFR can guarantee the deliverability between any source–destination pair [12].

Theorem 7 *Let Ω be a unit-area convex compact region with bounded curvature, and $\beta_0 = 1/\left(2/3-\sqrt{3}/2\pi\right) \approx 1.6^2$. Suppose that $n\pi r_n^2 = (\beta + o(1))\ln n$ for some $\beta > 0$. Then,*

1. *If $\beta > \beta_0$, then $\rho_{\mathrm{GFR}}\left(\mathrm{P}_n(\Omega)\right) \le r_n$ is asymptotically almost sure.*
2. *If $\beta < \beta_0$, then $\rho_{\mathrm{GFR}}\left(\mathrm{P}_n(\Omega)\right) > r_n$ is asymptotically almost sure.*

Applications

In the literature, r-disk graphs (or unit disk graphs by proper scaling) are widely used to model homogeneous wireless networks in which each node is equipped with an omnidirectional antenna. According to the path loss of radio frequency, the transmission ranges (radii) of wireless devices depend on transmission powers. For simplicity, the power assignment problem usually is modeled by a corresponding transmission range assignment problem. Recently, wireless ad-hoc networks have attracted attention from a lot of researchers because of various possible applications. In many of the possible applications, since wireless devices are powered by batteries, transmission range assignment has become one of the most important tools for prolonging system lifetime. By applying the theory of critical ranges, a randomly deployed wireless ad-hoc network may have good properties in high probability if the transmission range is larger than some critical value.

One application of critical ranges is to connectivity of networks. A network is k-vertex-

connected if there exist k node-disjoint paths between any pair of nodes. With such a property, at least k distinct communication paths exist between any pair of nodes, and the network is connected even if $k-1$ nodes fail. Thus, with a higher degree of connectivity, a network may have larger bandwidth and higher fault tolerance capacity. In addition, in [9, 14], and [15], networks with node or link failures were considered.

Another application is in topology control. To efficiently operate wireless ad-hoc networks, subsets of network topology will be constructed and maintained. The related topics are called topology control. A spanner is a subset of the network topology in which the minimal total cost of a path between any pair of nodes, e.g., distance or energy consumption, is only a constant fact larger than the minimal total cost in the original network topology. Hence spanners are good candidates for virtual backbones. Geometric structures, including Euclidean minimal spanning trees, relative neighbor graphs, Gabriel graphs, Delaunay triangulations, Yao's graphs, etc., are widely used ingredients to construct spanners [1, 5, 13]. By applying the knowledge of critical ranges, the complexity of algorithm design can be reduced, e.g., [3, 11].

Open Problems

A number of problems related to critical ranges remain open. Most problems discussed here apply 2-D plane geometry. In other words, the point set is in the plane. The first direction for future work is to study those problems in high-dimension spaces. Another open research area is on the longest-edge problems for other geometric structures, e.g., relative neighbor graphs and Yao's graphs. A third direction for future work involves considering relations between graph properties. A well-known result in random geometric graphs is that vanishment of isolated nodes asymptotically implies connectivity of networks. But for the wireless networks with unreliable links, this property is still open. In addition, in wireless sensor networks, the relations between connectivity and coverage are also interesting.

Cross-References

- Applications of Geometric Spanner Networks
- Connected Dominating Set
- Dilation of Geometric Networks
- Geometric Spanners
- Minimum Geometric Spanning Trees
- Minimum k-Connected Geometric Networks
- Randomized Broadcasting in Radio Networks
- Randomized Gossiping in Radio Networks

Recommended Reading

1. Cartigny J, Ingelrest F, Simplot-Ryl D, Stojmenovic I (2004) Localized LMST and RNG based minimum-energy broadcast protocols in ad hoc networks. Ad Hoc Netw 3(1):1–16
2. Dette H, Henze N (1989) The limit distribution of the largest nearest-neighbour link in the unit d-cube. J Appl Probab 26:67–80
3. Kozma G, Lotker Z, Sharir M, Stupp G (2004) Geometrically aware communication in random wireless networks. In: Proceedings of the twenty-third annual ACM symposium on principles of distributed computing, 25–28 July 2004, pp 310–319
4. Kumar S, Lai TH, Balogh J (2004) On k-coverage in a mostly sleeping sensor network. In: Proceedings of the 10th annual international conference on Mobile Computing and Networking (MobiCom'04), 26 Sept–1 Oct 2004
5. Li N, Hou JC, Sha L (2003) Design and analysis of a MST-based distributed topology control algorithm for wireless ad-hoc networks. In: 22nd annual joint conference of the IEEE computer and communications societies (INFOCOM 2003), vol 3, 1–3 Apr 2003, pp 1702–1712
6. Penrose M (1997) The longest edge of the random minimal spanning tree. Ann Appl Probab 7(2):340–361
7. Penrose M (1999) On k-connectivity for a geometric random graph. Random Struct Algorithms 15(2):145–164
8. Penrose M (2003) Random geometric graphs. Oxford University Press, Oxford
9. Wan P-J, Yi C-W (2005) Asymptotic critical transmission ranges for connectivity in wireless ad hoc networks with Bernoulli nodes. In: IEEE wireless communications and networking conference (WCNC 2005), 13–17 Mar 2005

10. Wan P-J, Yi C-W (2005) Coverage by randomly deployed wireless sensor networks. In: Proceedings of the 4th IEEE international symposium on Network Computing and Applications (NCA 2005), 27–29 July 2005
11. Wan P-J, Yi C-W (2007) On the longest edge of Gabriel graphs in wireless ad hoc networks. Trans Parallel Distrib Syst 18(1):1–16
12. Wan P-J, Yi C-W, Yao F, Jia X (2006) Asymptotic critical transmission radius for greedy forward routing in wireless ad hoc networks. In: Proceedings of the 7th ACM international symposium on mobile ad hoc networking and computing, 22–25 May 2006, pp 25–36
13. Wang Y, Li X-Y (2003) Localized construction of bounded degree and planar spanner for wireless ad hoc networks, In: Proceedings of the 2003 joint workshop on foundations of mobile computing (DIALM-POMC'03), 19 Sept 2003, pp 59–68
14. Yi C-W, Wan P-J, Li X-Y, Frieder O (2003) Asymptotic distribution of the number of isolated nodes in wireless ad hoc networks with Bernoulli nodes. In: IEEE wireless communications and networking conference (WCNC 2003), March 2003
15. Yi C-W, Wan P-J, Lin K-W, Huang C-H (2006) Asymptotic distribution of the number of isolated nodes in wireless ad hoc networks with unreliable nodes and links. In: The 49th annual IEEE GLOBECOMTechnical conference (GLOBECOM2006), 27 Nov–1 Dec 2006
16. Zhang H, Hou J (2004) On deriving the upper bound of α-lifetime for large sensor networks. In: Proceedings of the 5th ACM international symposium on mobile ad hoc networking & computing (MobiHoc 2004), 24–26 Mar 2004

Cryptographic Hardness of Learning

Adam Klivans
Department of Computer Science, University of Texas, Austin, TX, USA

Keywords

Representation-independent hardness for learning

Years and Authors of Summarized Original Work

1994; Kearns, Valiant

Problem Definition

This entry deals with proving negative results for distribution-free PAC learning. The crux of the problem is proving that a polynomial-time algorithm for learning various concept classes in the PAC model implies that several well-known cryptosystems are insecure. Thus, if we assume a particular cryptosystem is secure, we can conclude that it is impossible to efficiently learn a corresponding set of concept classes.

PAC Learning

We recall here the PAC learning model. Let C be a concept class (a set of functions over n variables), and let D be a distribution over the input space $\{0,1\}^n$. With C we associate a size function *size* that measures the complexity of each $c \in C$. For example, if C is a class of Boolean circuits, then $size(c)$ is equal to the number of gates in c. Let A be a randomized algorithm that has access to an oracle which returns labeled examples $(x, c(x))$ for some unknown $c \in C$; the examples x are drawn according to D. Algorithm A PAC learns concept class C by hypothesis class H if for any $c \in C$, for any distribution D over the input space, and any $\epsilon, \delta > 0$, A runs in time $poly(n, 1/\epsilon, 1/\delta, size(c))$ and produces a hypothesis $h \in H$ such that with probability at least $(1 - \delta)$, $Pr_D[c(x) \neq h(x)] < \epsilon$. This probability is taken over the random coin tosses of A as well as over the random labeled examples seen from distribution D. When $H = C$ (the hypothesis must be some concept in C), then A is a *proper* PAC learning algorithm. In this entry it is not assumed $H = C$, i.e., hardness results for *representation-independent* learning algorithms are discussed. The only assumption made on H is that for each $h \in H$, h can be evaluated in polynomial time for every input of length n.

Cryptographic Primitives

Also required is knowledge of various cryptographic primitives such as public-key cryptosystems, one-way functions, and one-way trapdoor functions. For a formal treatment of these primitives, refer to Goldreich [3].

Informally, a function f is *one way* if, after choosing a random x of length n and giving an adversary A only $f(x)$, it is computationally intractable for A to find y such that $f(y) = f(x)$. Furthermore, given x, $f(x)$ can be evaluated in polynomial time. That is, f is easy to compute one way, but there is no polynomial-time algorithm for finding pre-images of f on randomly chosen inputs. Say a function f is *trapdoor* if f is one way, but if an adversary A is given access to a secret "trapdoor" d, then A *can* efficiently find random pre-images of f.

Trapdoor functions that are permutations are closely related to *public-key cryptosystems*: imagine a person Alice who wants to allow others to secretly communicate with her. She publishes a one-way trapdoor permutation f so that it is publicly available to everyone, but keeps the "trapdoor" d to herself. Then Bob can send Alice a secret message x by sending her $f(x)$. Only Alice is able to invert f (recall f is a permutation) and recover x because only she knows d.

Key Results

The main insight in Kearns and Valiant's work is the following: if f is a trapdoor one-way function, and C is a circuit class containing the set of functions capable of inverting f given access to the trapdoor, then C is not efficiently PAC learnable, i. e., assuming the difficulty of inverting trap-door function f, there is a distribution on $\{0,1\}^n$ where no learning algorithm can succeed in learning f's associated decryption function.

The following theorem is stated in the (closely related) language of public-key cryptosystems:

Theorem 1 (Cryptography and learning; cf. Kearns and Valiant [4]) *Consider a public-key cryptosystem for encrypting individual bits into n-bit strings. Let C be a concept class that contains all the decryption functions $\{0,1\}^n \to \{0,1\}$ of the cryptosystem. If C is PAC learnable in polynomial time, then there is a polynomial-time distinguisher between the encryptions of 0 and 1.*

The intuition behind the proof is as follows: fix an encryption function f, associated secret key d, and let C be a class of functions such that the problem of inverting $f(x)$ given d can be computed by an element c of C; notice that knowledge of d is not necessary to generate a polynomial-size sample of $(x, f(x))$ pairs.

If C is PAC learnable, then given a relatively small number of encrypted messages $(x, f(x))$, a learning algorithm A can find a hypothesis h that will approximate c and thus have a non-negligible advantage for decrypting future *randomly* encrypted messages. This violates the security properties of the cryptosystem.

A natural question follows: "what is the simplest concept class that can compute the decryption function for secure public-key cryptosystems?" For example, if a public-key cryptosystem is proven to be secure, and encrypted messages can be decrypted (given the secret key) by polynomial-size DNF formulas, then, by Theorem 1, one could conclude that polynomial-size DNF formulas cannot be learned in the PAC model.

Kearns and Valiant do not obtain such a hardness result for learning DNF formulas (it is still an outstanding open problem), but they do obtain a variety of hardness results assuming the security of various well-known public-key cryptosystems based on the hardness of number-theoretic problems such as factoring.

The following list summarizes their main results:

- Let C be the class of polynomial-size Boolean formulas (not necessarily DNF formulas) or polynomial-size circuits of logarithmic depth. Assuming that the RSA cryptosystem is secure, or recognizing quadratic residues is intractable, or factoring Blum integers is intractable, C is not PAC learnable.
- Let C be the class of polynomial-size deterministic finite automata. Under the same assumptions as above, C is not PAC learnable.
- Let C be the class of constant depth threshold circuits of polynomial size. Under the same assumptions as above, C is not PAC learnable.

The depth of the circuit class is not specified but it can be seen to be at most 4.

Kearns and Valiant also prove the intractability of finding optimal solutions related to coloring problems assuming the security of the above cryptographic primitives (e.g., breaking RSA).

Relationship to Hardness Results for Proper Learning

The key results above should not be confused with the extensive literature regarding hardness results for *properly* PAC learning concept classes. For example, it is known [1] that, unless RP = NP, it is impossible to properly PAC learn polynomial-size DNF formulas (i.e., require the learner to learn DNF formulas by outputting a DNF formula as its hypothesis). Such results are *incomparable* to the work of Kearns and Valiant, as they require something much stronger from the learner but take a much weaker assumption (RP $\neq$ NP is a weaker assumption than the assumption that RSA is secure).

Applications and Related Work

Valiant [10] was the first to observe that the existence of a particular cryptographic primitive (pseudorandom functions) implies hardness results for PAC learning concept classes. The work of Kearns and Valiant has subsequently found many applications. Klivans and Sherstov have recently shown [7] that the problem of PAC learning intersections of halfspaces (a very simple depth-2 threshold circuit) is intractable unless certain lattice-based cryptosystems due to Regev [9] are not secure. Their result makes use of the Kearns and Valiant approach. Angluin and Kharitonov [2] have extended the Kearns and Valiant paradigm to give cryptographic hardness results for learning concept classes even if the learner has *query access* to the unknown concept. Kharitonov [6] has given hardness results for learning polynomial-size, constant depth circuits that assumes the existence of secure pseudorandom generators rather than the existence of public-key cryptosystems.

Open Problems

The major open problem in this line of research is to prove a cryptographic hardness result for PAC learning polynomial-size DNF formulas. Currently, polynomial-size DNF formulas seem far too weak to compute cryptographic primitives such as the decryption function for a well-known cryptosystem. The fastest known algorithm for PAC learning DNF formulas runs in time $2^{\tilde{O}(n^{1/3})}$ [8].

Cross-References

▸ PAC Learning

Recommended Reading

1. Alekhnovich M, Braverman M, Feldman V, Klivans AR, Pitassi T (2004) Learnability and automatizability. In: Proceedings of the 45th symposium on foundations of computer science, Rome
2. Angluin D, Kharitonov M (1995) When won't membership queries help? J Comput Syst Sci 50(2):336-355
3. Goldreich O (2001) Foundations of cryptography: basic tools. Cambridge University Press, Cambridge/New York
4. Kearns M, Valiant L (1994) Cryptographic limitations on learning boolean formulae and finite automata. J ACM 41(1):67–95
5. Kearns M, Vazirani U (1994) An introduction to computational learning theory. MIT, Cambridge
6. Kharitonov M (1993) Cryptographic hardness of distribution-specific learning. In: Proceedings of the twenty-fifth annual symposium on theory of computing, San Diego, pp 372–381
7. Klivans A, Sherstov AA (2006) Cryptographic hardness for learning intersections of halfspaces. In: Proceedings of the 47th symposium on foundations of computer science, Berkeley
8. Klivans A, Servedio R (2001) Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. In: Proceedings of the 33rd annual symposium on theory of computing, Heraklion
9. Regev O (2004) New lattice-based cryptographic constructions. J ACM 51:899–942
10. Valiant L (1984) A theory of the learnable. Commun ACM 27(11):1134–1142

Cuckoo Hashing

Rasmus Pagh
Theoretical Computer Science, IT University of Copenhagen, Copenhagen, Denmark

Keywords

Dictionary; Hash table; Key-value store; Open addressing

Years and Authors of Summarized Original Work

2001; Pagh, Rodler

Problem Definition

A *dictionary* (also known as an *associative array*) is an abstract data structure capable of storing a set S of elements, referred to as *keys*, and information associated with each key. The operations supported by a dictionary are insertion of a key (and associated information), deletion of a key, and lookup of a key (retrieving the associated information). In case a lookup is made on a key that is not in S, this must be reported by the data structure.

The *hash table* is a class of data structures use to implement dictionaries in the RAM model of computation. *Open addressing* hash tables are a particularly simple type of hash table, where the data structure is an array such that each entry either contains a key of S or is marked "empty." *Cuckoo hashing* addresses the problem of implementing an open addressing hash table with *worst-case* constant lookup time. Specifically, a constant number of entries in the hash table should be associated with each key x, such that x is present in one of these entries if $x \in S$.

In the following it is assumed that a key and the information associated with a key are single machine words. This is essentially without loss of generality: If more associated data is wanted, it can be referred to using a pointer. If keys are longer than one machine word, they can be mapped down to a single (or a few) machine words using universal hashing [4] and the described method used on the hash values (which are unique to each key with high probability). The original key must then be stored as associated data. Let n denote an upper bound on the size of S. To allow the size of the set to grow beyond n, *global rebuilding* can be used.

Key Results

Prehistory

It has been known since the advent of universal hashing [4] that if the hash table has $r \geq n^2$ entries, a lookup can be implemented by retrieving just a single entry in the hash table. This is done by storing a key x in entry $h(x)$ of the hash table, where h is a function from the set of machine words to $\{1, \ldots, n^2\}$. If h is chosen at random from a *universal family* of hash functions [4], then with probability at least 1/2 every key in S is assigned a unique entry. The same behavior would be seen if h was a random function, but in contrast to random functions, there are universal families that allow efficient storage and evaluation of h (constant number of machine words and constant evaluation time).

This overview concentrates on the case where the space used by the open- addressing hash table is linear, $r = O(n)$. It was shown by Azar et al. [1] that it is possible to combine linear space with worst-case constant lookup time. It was not considered how to construct the data structure. Since randomization is used, all schemes discussed have a probability of error. However, this probability is small, $O(1/n)$ or less for all schemes, and an error can be handled by *rehashing* (changing the hash functions and rebuilding the hash table). The result of [1] was shown under the assumption that the algorithm is given free access to a number of truly random hash functions. In many of the subsequent papers, it is shown how to achieve the bounds using explicitly defined hash functions. However, no attempt is made here to cover these constructions.

In the following, let ε denote an arbitrary positive constant. Pagh [11] showed that retrieving two entries from the hash table suffices when

$r \geq (2+\varepsilon)n$. Specifically, lookup of a key x can be done by retrieving entries $h_1(x)$ and $h_2(x)$ of the hash table, where h_1 and h_2 are random hash functions mapping machine words to $\{1,\ldots,r\}$. The same result holds if h_1 has range $\{1,\ldots,r/2\}$ and h_2 has range $\{r/2+1,\ldots,r\}$, that is, if the two lookups are done in disjoint parts of memory.

It follows from [11] that it is not possible to perform lookup by retrieving a *single* entry in the worst case unless the hash table is of size $n^{2-o(1)}$.

Cuckoo Hashing

Pagh and Rodler [12] showed how to maintain the data structure of Pagh [11] under insertions. They considered the variant in which the lookups are done in disjoint parts of the hash table. It will be convenient to think of these as separate arrays, T_1 and T_2. Let $\perp$ denote the contents of an empty hash table entry, and let $x \leftrightarrow y$ express that the values of variables x and y are swapped. The proposed dynamic algorithm, called *cuckoo hashing*, performs insertions by the following procedure:

```
procedure insert(x)
  i := 1;
  repeat
    x ↔ T_i[h_i(x)]; i := 3 − i;
  until x = ⊥
  end
```

At any time the variable x holds a key that needs to be inserted in the table, or $\perp$. The value of i changes between 1 and 2 in each iteration, so the algorithm is alternately exchanging the contents of x with a key from Table 1 and Table 2. Conceptually, what happens is that the algorithm moves a sequence of zero or more keys from one table to the other to make room for the new key. This is done in a greedy fashion, by kicking out any key that may be present in the location where a key is being moved. The similarity of the insertion procedure and the nesting habits of the European cuckoo is the reason for the name of the algorithm.

The pseudocode above is slightly simplified. In general the algorithm needs to make sure not to insert the same key twice and handle the possibility that the insertion may not succeed (by rehashing if the loop takes too long).

Theorem 1 *Assuming that* $r \geq (2+\varepsilon)n$, *the expected time for the cuckoo hashing insertion procedure is O(1).*

Generalizations of Cuckoo Hashing

Cuckoo hashing has been generalized in several directions. Kirsh et al. [8] showed that keeping a small *stash* of memory locations that are inspected at every lookup can significantly reduce the error probability of cuckoo hashing.

More generally the case of $k > 2$ hash functions has been considered. Also, the hash table may be divided into "buckets" of size b, such that the lookup procedure searches an entire bucket for each hash function. Let (k, b)-cuckoo denote a scheme with k hash functions and buckets of size b. What was described above is a (2,1)-*cuckoo* scheme. Already in 1999, (4,1)-cuckoo was described in a patent application by David A. Brown (US patent 6,775,281). Fotakis et al. described and analyzed a $(k, 1)$-cuckoo scheme in [7], and a (2,b)-cuckoo scheme was described and analyzed by Dietzfelbinger and Weidling [5]. In both cases, it was shown that space utilization arbitrarily close to 100 % is possible and that the necessary fraction of unused space decreases exponentially with k and b. The insertion procedure considered in [5, 7] is a breadth-first search for the shortest sequence of key moves that can be made to accommodate the new key. Panigrahy [13] studied (2,2)-cuckoo schemes in detail, showing that a space utilization of 83 % can be achieved dynamically, still supporting constant time insertions using breadth-first search. In a *static* setting with no updates, thresholds for general (k, b)-cuckoo hashing have been established (see LeLarge [10] and its references).

Applications

Dictionaries (sometimes referred to as key-value stores) have a wide range of uses in computer science and engineering. For example, dictionaries arise in many applications in string algorithms

and data structures, database systems, data compression, and various information retrieval applications. Also, cuckoo hashing has been used in oblivious RAM simulations and other cryptographic constructions [2, 14].

Open Problems

The results above provide a good understanding of the properties of open-addressing schemes with worst-case constant lookup time. However, several aspects are still not understood satisfactorily.

First of all, there is no practical class of hash functions for which the above results can be shown. The only explicit classes of hash functions that are known to make the methods work either have evaluation time $\Theta(\log n)$ or use space $n^{\Omega(1)}$. It is an intriguing open problem to construct a class having constant evaluation time and space usage.

For the generalizations of cuckoo hashing, the use of breadth-first search is not so attractive in practice, due to the associated overhead in storage. A simpler approach that does not require any storage is to perform a random walk where keys are moved to a random, alternative position. (This generalizes the cuckoo hashing insertion procedure, where there is only one alternative position to choose.) Panigrahy [13] showed that this works for (2,2)-cuckoo when the space utilization is low. However, it is unknown whether this approach works well as the space utilization approaches 100 %.

Finally, many of the analyses that have been given are not tight. In contrast, most classical open addressing schemes have been analyzed very precisely. It seems likely that precise analysis of cuckoo hashing and its generalizations is possible using techniques from analysis of algorithms and tools from the theory of random graphs. In particular, the relationship between space utilization and insertion time is not well understood. A precise analysis of the probability that cuckoo hashing fails has been given by Kutzelnigg [9].

Experimental Results

All experiments on cuckoo hashing and its generalizations so far presented in the literature have been done using simple, heuristic hash functions. Pagh and Rodler [12] presented experiments showing that, for space utilization 1/3, cuckoo hashing is competitive with open addressing schemes that do not give a worst-case guarantee. Zukowski et al. [15] showed how to implement cuckoo hashing such that it runs very efficiently on pipelined processors with the capability of processing several instructions in parallel. For hash tables that are small enough to fit in cache, cuckoo hashing was 2 to 4 times faster than chained hashing in their experiments. Erlingsson et al. [6] considered (k, b)-cuckoo schemes for various combinations of small values of k and b, showing that very high space utilization is possible even for modestly small values of k and b. For example, a space utilization of 99.9 % is possible for $k = b = 4$. It was further found that the resulting algorithms were very robust. Experiments in [7] indicate that the random walk insertion procedure performs as well as one could hope for.

Cross-References

- Dictionary Matching
- Online Load Balancing of Temporary Tasks

Recommended Reading

1. Azar Y, Broder AZ, Karlin AR, Upfal E (1999) Balanced allocations. SIAM J Comput 29(1):180–200
2. Berman I, Haitner I, Komargodski I, Naor M (eds) (2013) Hardness preserving reductions via cuckoo hashing. In: Theory of cryptography. Springer, Berlin/New York, pp 40–59
3. Cain JA, Sanders P, Wormald N (2007) The random graph threshold for k-orientability and a fast algorithm for optimal multiple-choice allocation. In: Proceedings of the 18th annual ACM-SIAM symposium on discrete algorithms (SODA), New Orleans. ACM, pp 469–476
4. Carter JL, Wegman MN (1979) Universal classes of hash functions. J Comput Syst Sci 18(2):143–154

5. Dietzfelbinger M, Weidling C (2005) Balanced allocation and dictionaries with tightly packed constant size bins. In: ICALP, Lisbon. Lecture notes in computer science, vol 3580. Springer, pp 166–178
6. Erlingsson Ú, Manasse M, McSherry F (2006) A cool and practical alternative to traditional hash tables. In: Proceedings of the 7th workshop on distributed data and structures (WDAS), Santa Clara
7. Fotakis D, Pagh R, Sanders P, Spirakis PG (2005) Space efficient hash tables with worst case constant access time. Theory Comput Syst 38(2):229–248
8. Kirsch A, Mitzenmacher M, Wieder U (2009) More robust hashing: cuckoo hashing with a stash. SIAM J Comput 39(4):1543–1561
9. Kutzelnigg R (2006) Bipartite random graphs and cuckoo hashing. In: Proceedings of 4th colloquium on mathematics and computer science, Nancy
10. LeLarge M (2012) A new approach to the orientation of random hypergraphs. In: Proceedings of the 23rd annual ACM-SIAM symposium on discrete algorithms (SODA), Kyoto. ACM, pp 251–264
11. Pagh R (2001) On the cell probe complexity of membership and perfect hashing. In: Proceedings of the 33rd annual ACM symposium on theory of computing (STOC), Heraklion. ACM, pp 425–432
12. Pagh R, Rodler FF (2004) Cuckoo hashing. J Algorithms 51:122–144
13. Panigrahy R (2005) Efficient hashing with lookups in two memory accesses. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms (SODA), Vancouver. SIAM, pp 830–839
14. Pinkas B, Reinman T (2010) Oblivious RAM revisited. In: Advances in cryptology–CRYPTO 2010, Santa Barbara. Springer, pp 502–519
15. Zukowski M, Heman S, Boncz PA (2006) Architecture-conscious hashing. In: Proceedings of the international workshop on data management on new hardware (DaMoN), Chicago. ACM, Article No 6

Current Champion for Online Bin Packing

Rob van Stee
University of Leicester, Leicester, UK

Keywords

Bin packing; Champion; Competitive analysis; Online algorithms

Years and Authors of Summarized Original Work

2002; Seiden

Problem Definition

In the online bin packing problem, a sequence of *items* with sizes in the interval $(0, 1]$ arrive one by one and need to be packed into *bins*, so that each bin contains items of total size at most 1. Each item must be irrevocably assigned to a bin before the next item becomes available. The algorithm has no knowledge about future items. There is an unlimited supply of bins available, and the goal is to minimize the total number of used bins (bins that receive at least one item).

The most common performance measure for online bin packing algorithms is the asymptotic performance ratio, or asymptotic competitive ratio, which is defined as

$$R_{\mathrm{ASY}}(A) := \limsup_{n\to\infty} \left\{ \max_{L} \left\{ \frac{A(L)}{n} \middle| \mathrm{OPT}(L)=n \right\} \right\}. \tag{1}$$

Hence, for any input L, the number of bins used by an online algorithm A is compared to the optimal number of bins needed to pack the same input. Note that calculating the optimal number of bins might take exponential time; moreover, it requires that the entire input is known in advance.

Key Results

This paper presents a new framework for analyzing online bin packing algorithms. It can be used to analyze all known versions of the well-known Harmonic algorithm, including a new version introduced in this paper.

The Harmonic algorithm [4] partitions the input into types depending on its size and packs each type separately. Harmonic-k has k types, and type i consists of items of size in the interval $(1/(i+1), 1/i]$. Harmonic has k open bins at

all times, one for each type, and packs i items of type i in one bin. Thus it achieves an asymptotic performance ratio of 1.691.

For some inputs, this algorithm wastes a lot of space in some bins, for instance, if many items of size $1/2 + \varepsilon$ arrive for some small $\varepsilon > 0$. Several authors improved on the basic Harmonic algorithm by combining some items of different types together into bins. Typically this is done by partitioning the intervals $(1/2, 1]$ and $(1/3, 1/2]$ further, guaranteeing that items can be combined. As a simple example, by introducing intervals $(1/2, 0.6]$ and $(1/3, 0.4]$, we can guarantee that items of these two new types can always be packed together in a single bin. Furthermore, the remaining intervals $(0.6, 1]$ and $(0.4, 0.5]$ now give better area guarantees than before: bins with items of these types will be at least 0.6 and at least 0.8 full, respectively.

Seiden builds on this idea and gives a new algorithm, Harmonic++, which beats all previously known algorithms and is still the best algorithm known. He used a computer-assisted search to set the many parameters of this algorithm. The algorithm partitions the intervals $(1/2, 1]$ and $(1/3, 1/2]$ in ten matching subintervals (in the sense described above) and also partitions intervals of several smaller types, using no less than 70 intervals in total. Also using a computer search, he proves that the asymptotic performance ratio of this algorithm is at most 1.58889.

Seiden also showed that the asymptotic performance ratio of a similar algorithm presented earlier, Harmonic+1, is at least 1.5972, disproving a claim by Richey [6] that Harmonic+1 is 1.58872-competitive.

The framework introduced by Seiden was used later in other contexts, for instance, for two-dimensional bin packing, where a set of rectangles needs to be packed into square bins. Han et al. [3] presented an algorithm with asymptotic performance ratio 2.5545. For the special case of packing squares, Han et al. [2] presented an algorithm with asymptotic performance ratio 2.1187.

Open Problems

The algorithm is very close to optimal for Harmonic-type algorithms, for which Ramanan et al. [5] showed a lower bound of 1.58333.... However, the general lower bound for this problem is only 1.54037 [1]. It is very difficult to see how either result can be improved, and this remains a challenging open problem which will require new ideas. There has been essentially no improvement in this area in over a decade.

Cross-References

▸ Harmonic Algorithm for Online Bin Packing
▸ Lower Bounds for Online Bin Packing

Recommended Reading

1. Balogh J, Békési J, Galambos G (2012) New lower bounds for certain bin packing algorithms. Theor Comput Sci 440–441:1–13
2. Han X, Ye D, Zhou Y (2010) A note on online hypercube packing. Cent Eur J Oper Res 18:221–239
3. Han X, Chin FYL, Ting H-F, Zhang G, Zhang Y (2011) A new upper bound 2.5545 on 2D online bin packing. ACM Trans Algorithms 7(4):50
4. Lee CC, Lee DT (1985) A simple online bin packing algorithm. J ACM 32:562–572
5. Ramanan P, Brown DJ, Lee CC, Lee DT (1989) Online bin packing in linear time. J Algorithms 10:305–326
6. Richey MB (1991) Improved bounds for harmonic-based bin packing algorithms. Discret Appl Math 34:203–227

Curve Reconstruction

Stefan Funke
Department of Computer Science, Universität Stuttgart, Stuttgart, Germany

Keywords

Computational geometry; Curve reconstruction

Years and Authors of Summarized Original Work

1998; Amenta, Bern, Eppstein
1999; Dey, Kumar
1999; Dey, Mehlhorn, Ramos
1999; Giesen
2001; Funke, Ramos
2003; Cheng, Funke, Golin, Kumar, Poon, Ramos

Problem Definition

Given a set S of sample points from a collection Γ of simple (nonintersecting) curves in the Euclidean plane, *curve reconstruction* is the problem of computing the graph $G(S, \Gamma)$, called the *correct reconstruction*, whose vertex set is S and that has an edge between two vertices if and only if the respective samples are adjacent on a curve in Γ; see Fig. 1.

Obviously, it is not possible to correctly reconstruct a given collection of curves from an arbitrary sample set from it. Therefore, some restriction on the sample set S – a so-called sampling condition – is required which specifies how dense a sampling has to be to guarantee a correct output of an algorithm. The difficulty for an algorithm to solve the curve reconstruction problem (and to come up with a suitable sampling condition) varies with the classes of allowed curves in Γ and whether the set S is actually sampled from the curves or noisy.

Key Results

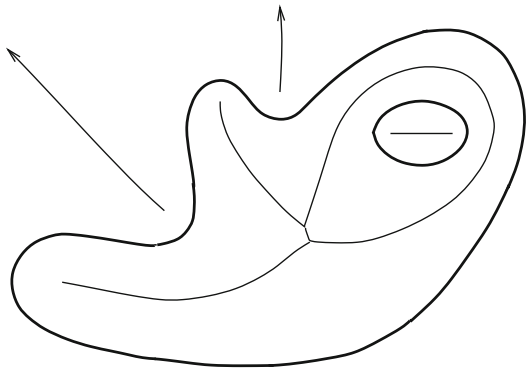
If the curves are closed, smooth, and uniformly sampled – that is, with a uniform maximum distance between adjacent sample points – several methods for the curve reconstruction problem are known to work ranging over minimum spanning trees,α-shapes, β-skeletons [KR85], and r-regular shapes; see the survey by Edelsbrunner [7]. The focus of this section are approaches which can deal with *nonuniform sampling conditions*, that is, conditions which allow sparser sampling in areas of low detail and require higher sampling only in areas of high detail.

Closed Smooth Curves

Amenta, Bern, and Epstein [2] introduced the concept of the *local feature size* $\mathrm{lfs}(p)$ of a point $p \in \Gamma$ which is defined as the distance to the *medial axis* of Γ. The medial axis of a collection of curves Γ is defined as the set of points in the plane which have more than one closest point on a curve in Γ; see Fig. 2. Roughly speaking, a neighborhood of a point of size equal to its local feature size is intersected by the curves in a single piece that winds up only a small angle.

The introduction of the local feature size allowed for a very elegant sampling condition: A sample set S is called an ϵ-sampling for a collection of curves Γ, if for all $p \in \Gamma$, $\exists s \in S$ with $|ps| \leq \epsilon \mathrm{lfs}(p)$. This condition naturally captures the intuition that "complicated" areas of the curve require higher sampling density than areas of low detail.

Curve Reconstruction, Fig. 1 A collection of curves Γ, a sample S from Γ, and the correct reconstruction G(S,Γ)



Curve Reconstruction, Fig. 2 The light curves are the medial axis of the heavy curves (Courtesy of N. Amenta, M. Bern, and D. Eppstein)

For small enough ϵ, the Voronoi nodes in the Voronoi diagram $VD(S)$ of an ϵ-sampling S for Γ approximate the medial axis of Γ. Based on that intuition, the *CRUST* algorithm in [1] outputs as correct reconstruction the edges of the Delaunay triangulation of S having a ball empty of Voronoi vertices in the Voronoi diagram $VD(S)$. For $\epsilon < 0.252$ CRUST provably outputs the correct reconstruction of an ϵ-sampling S with respect to a collection of closed smooth curves Γ.

In the same paper, the authors could also show that a known algorithm – the β-skeleton – for suitable choice of β also correctly reconstructs a collection of closed smooth curves for $\epsilon < 0.297$.

Later, Dey and Kumar [4] presented an extremely simple and straightforward algorithm connecting essentially the nearest neighbors on opposite sides. They could prove this algorithm to be correct under the local feature size sampling condition for $\epsilon \leq 1/3$. What is particularly interesting about this algorithm is the fact that decisions which points to connect are made based on a very local neighborhood of the respective points. This idea later nicely translated to algorithms for the 3- and higher-dimensional manifold reconstruction problem.

Open Smooth Curves

When considering the larger class of *open* and closed smooth curves, there is a little caveat. While one can guarantee for sufficiently dense samplings, i.e., small enough values of ϵ, that all edges of the correct reconstruction are present in the output of a reconstruction algorithm, one cannot always avoid the inclusion of additional edges in the output of an algorithm. Essentially, the problem is that a sample set, S set, might be an ϵ-sampling for two collections of curves Γ and Γ' with different correct reconstructions $G(S, \Gamma)$ and $G(S, \Gamma')$ irrespectively how small ϵ is chosen.

Dey, Mehlhorn, and Ramos in [5] introduced the concept of a *witness curve* Γ^*, proving the following guarantee for their *CONSERVATIVE CRUST* algorithm: If S is an ϵ-sampling for a collection of open and closed smooth curves Γ, their algorithm returns a reconstruction H such that $H \supseteq G(S, \Gamma)$, that is, H contains all edges of the correct reconstruction. Furthermore, the algorithm outputs a curve Γ^* such that S is an $\epsilon' \approx \epsilon$-sampling for Γ^* and $H = G(S, \Gamma^*)$. Their algorithm is similar to the CRUST algorithm in that it identifies a subcomplex of the Delaunay triangulation.

Closed Curves with Corners

Another natural extension of the allowed classes of curves in Γ is the inclusion of curves with corners, that is, points where left and right tangent do not coincide. Unfortunately, in this case, one cannot use the sampling condition based on the local feature size since the medial axis actually touches the corners, requiring an infinitely dense sampling near corners. The first algorithms to deal with a *single* closed curve possibly with corners were by Giesen [9] and Althaus/Mehlhorn [2] based on the construction of a travelling salesman tour. In their sampling condition, areas of the curve nearby a corner were exempt from the ϵ-sampling condition. [2] could even prove that the respective TSP instance can be solved in polynomial time for sufficiently dense sample sets. Dey and Wenger [6] proposed a non-TSP-based approach for collections of closed curves with corners.

Open and Closed Curves with Corners

Finally Funke/Ramos [8] considered the case of collections of open and closed curves. While their algorithm also comes with a guarantee for

some variant of an ϵ-sampling condition (with special condition near corners like [9] and [2]), they also propose a sampling condition which is expressed with respect to the correct reconstruction $G(S, \Gamma)$. Not being based on a travelling salesman tour computation, their algorithm can also handle collections containing several open curves. As [5] the algorithm also produces a collection of witness curves Γ'.

Noisy Sample Sets

A generalization in a different direction is the consideration of sample sets S which do not consist of points exactly *on* the curves in Γ but – e.g., due to measurement errors – lie only "nearby." In [3] the authors considered such noisy sample sets from a collection of disjoint smooth closed curves and could prove for a perturbed locally uniform sample set that their algorithm computes as output a set of polygonal curves converging to Γ as the sampling density increases.

Cross-References

▸ Manifold Reconstruction

Recommended Reading

1. Amenta N, Bern M, Eppstein D (1998) The Crust and the beta-skeleton: combinatorial curve reconstruction. Graph Models Image Process 60/2:2:125–135
2. Althaus E, Mehlhorn K (2000) TSP-based curve reconstruction in polynomial time. In: Proceedings of the 11th annual ACM-SIAM symposium on discrete algorithms, San Francisco, pp 686–695
3. Cheng S-W, Funke S, Golin MJ, Kumar P, Poon S-H, Ramos E A (2003) Curve reconstruction from noisy samples. In: Proceedings of the 19th ACM symposium on computational geometry, San Diego, pp 302–311
4. Dey TK, Kumar P (1999) A simple provable algorithm for curve reconstruction. In: Proceedings of the 10th ACM-SIAM symposium on discrete algorithms, Baltimore, pp 893–894
5. Dey TK, Mehlhorn K, Ramos EA (1999) Curve reconstruction: connecting dots with good reason. In: Proceedings of the 15th annual acm symposium on computational geometry, Miami Beach, pp 197–206
6. Dey TK, Wenger R (2000) Reconstruction curves with sharp corners. In: Proceedings of the 16th annual ACM symposium on computational geometry, Hong Kong, pp 233–241
7. Edelsbrunner H (1998) Shape reconstruction with delaunay complex. In: Proceedings of the 2nd Latin American theoretical informatics symposium, Campinas. LNCS, vol 1380, pp 119–132
8. Funke S, Ramos EA (2001) Reconstructing a collection of curves with corners and endpoints. In: Proceedings of the 12th annual ACM-SIAM symposium on discrete algorithms, Washington, DC, pp 344–353
9. Giesen J (1999) Curve reconstruction, the traveling salesman problem and Menger's theorem on length. In: Proceedings of the 15th annual ACM symposium on computational geometry, Miami Beach, pp 207–216

D

Data Migration

Yoo-Ah Kim
Computer Science and Engineering Department, University of Connecticut, Storrs, CT, USA

Keywords

Data movements; File transfers

Years and Authors of Summarized Original Work

2004; Khuller, Kim, Wan

Problem Definition

The problem is motivated by the need to manage data on a set of storage devices to handle dynamically changing demand. To maximize utilization, the data layout (i.e., a mapping that specifies the subset of data items stored on each disk) needs to be computed based on disk capacities as well as the demand for data. Over time as the demand for data changes, the system needs to create new data layout. The data migration problem is to compute an efficient schedule for the set of disks to convert an initial layout to a target layout.

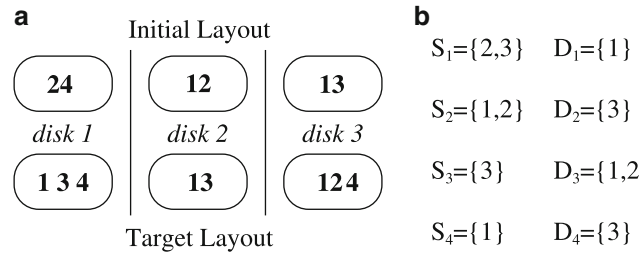
The problem is defined as follows. Suppose that there are N disks and Δ data items, and an initial layout and a target layout are given (see Fig. 1a for an example). For each item i, source disks S_i is defined to be a subset of disks which have item i in the initial layout. Destination disks D_i is a subset of disks that want to receive item i. In other words, disks in D_i have to store item i in the target layout but do not have to store it in the initial layout. Figure 1b shows the corresponding S_i and D_i. It is assumed that $S_i \neq \emptyset$ and $D_i \neq \emptyset$ for each item i. Data migration is the transfer of data to have all D_i receive data item i residing in S_i initially, and the goal is to minimize the total amount of time required for the transfers.

Assume that the underlying network is fully connected and the data items are all the same size. In other words, it takes the same amount of time to migrate an item from one disk to another. Therefore, migrations are performed in rounds. Consider the half-duplex model, where each disk can participate in the transfer of only one item – either as a sender or as a receiver. The objective is to find a migration schedule using the minimum number of rounds. No bypass nodes (A bypass node is a node that is not the target of a move operation, but is used as an intermediate holding point for a data item.) can be used and therefore all data items are sent only to disks that desire them.

Key Results

Khuller et al. [11] developed a 9.5-approximation for the data migration problem, which was later improved to $6.5 + o(1)$. In the next subsection,

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4



Data Migration, Fig. 1
Left An example of initial and target layout and *right* their corresponding S_i's and D_i's

the lower bounds of the problem are first examined.

Notations and Lower Bounds

1. **Maximum in-degree (β):** Let β_j be the number of data items that a disk j has to receive. In other words, $\beta_j = |\{i \,|\, j \in D_i\}|$. Then $\beta = \max_j \beta_j$ is a lower bound on the optimal as a disk can receive only one data item in one round.
2. **Maximum number of items that a disk may be a source or destination for (α):** For each item i, at least one disk in S_i should be used as a source for the item, and this disk is called a *primary source*. A unique primary source $s_i \in S_i$ for each item i that minimizes $\alpha = \max_{j=1,\ldots,N}(|\{i \,|\, j = s_i\}| + \beta_j)$ can be found using a network flow. Note that $\alpha \geq \beta$, and α is also a lower bound on the optimal solution.
3. **Minimum time required for cloning (M):** Let a disk j make a copy of item i at the kth round. At the end of the mth round, the number of copies that can be created from the copy is at most 2^{m-k} as in each round the number of copies can only be doubled. Also note that each disk can make a copy of only one item in one round. Since at least $|D_i|$ copies of item i need to be created, the minimum m that satisfies the following linear program gives a lower bound on the optimal solution: **L(m):**

$$\sum_j \sum_{k=1}^{m} 2^{m-k} x_{ijk} \geq |D_i| \quad \text{for all } i \qquad (1)$$

$$\sum_i x_{ijk} \leq 1 \quad \text{for all } j,k \qquad (2)$$

$$0 \leq x_{ijk} \leq 1 \qquad (3)$$

Data Migration Algorithm

A 9.5-approximation can be obtained as follows. The algorithm first computes representative sets for each item and sends the item to the representative sets first, which in turn send the item to the remaining set. Representative sets are computed differently depending on the size of D_i.

Representatives for Big Sets

For sets with size at least β, a *disjoint* collection of *representative sets* $R_i, i = 1 \ldots \Delta$ has to satisfy the following properties: Each R_i should be a subset of D_i and $|R_i| = \lfloor |D_i|/\beta \rfloor$. The representative sets can be found using a network flow.

Representatives for Small Sets

For each item i, let $\gamma_i = |D_i| \bmod k$. A *secondary representative*$_i$ in D_i for the items with $\gamma_i \neq 0$ needs to be computed. A disk j can be a secondary representative r_i for several items as long as $\sum_{i \in I_j} \gamma_i \leq 2\beta - 1$, where I_j is a set of items for which j is a secondary representative. This can be done by applying the Shmoys–Tardos algorithm [17] for the generalized assignment problem.

Scheduling Migrations

Given representatives for all data items, migrations can be done in three steps as follows:

1. **Migration to R_i:** Each item i is first sent to the set R_i. By converting a fractional solution given in $L(M)$, one can find a migration schedule from s_i to R_i and it requires at most $2M + \alpha$ rounds.
2. **Migration to r_i:** Item i is sent from primary source s_i to r_i. The migrations can be done in 1.5α rounds, using an algorithm for edge coloring [16].

3. **Migration to the remaining disks:** A transfer graph from representatives to the remaining disks can now be created as follows. For each item i, add directed edges from disks in R_i to $(\beta - 1)\lfloor \frac{|D_i|}{\beta} \rfloor$ disks in $D_i \setminus R_i$ such that the out-degree of each node in R_i is at most $\beta - 1$ and the in-degree of each node in $D_i \setminus R_i$ from R_i is 1. A directed edge is also added from the secondary representative r_i of item i to the remaining disks in D_i which do not have an edge coming from R_i. It has been shown that the maximum degree of the transfer graph is at most $4\beta - 5$ and the multiplicity is $\beta + 2$. Therefore, migration for the transfer graph can be done in $5\beta - 3$ rounds using an algorithm for multigraph edge coloring [18].

Analysis

Note that the total number of rounds required in the algorithm described in "Data Migration Algorithm" is at most $2M + 2.5\alpha + 5\beta - 3$. As α, β and M are lower bounds on the optimal number of rounds, the abovementioned algorithm gives a 9.5-approximation.

Theorem 1 ([11]) *There is a 9.5-approximation algorithm for the data migration problem.*

Khuller et al. [10] later improved the algorithm and obtained a $(6.5 + o(1))$-approximation.

Theorem 2 ([10]) *There is a (6.5 + o(1))-approximation algorithm for the data migration problem.*

Applications

Data Migration in Storage Systems

Typically, a large storage server consists of several disks connected using a dedicated network, called a *storage area network*. To handle high demand, especially for multimedia data, a common approach is to replicate data objects within the storage system. Disks typically have constraints on storage as well as the number of clients that can access data from a single disk simultaneously. Approximation algorithms have been developed to map known demand for data to a specific data layout pattern to maximize utilization (The utilization is the total number of clients that can be assigned to a disk that contains the data they want.) [4, 8, 14, 15]. In the layout, they compute not only how many copies of each item need to be created, but also a layout pattern that specifies the precise subset of items on each disk. The problem is NP-hard, but there are polynomial-time approximation schemes [4, 8, 14]. Given the relative demand for data, the algorithm computes an almost optimal layout.

Over time as the demand for data changes, the system needs to create new data layouts. To handle high demand for popular objects, new copies may have to be dynamically created and stored on different disks. The data migration problem is to compute a specific schedule for the set of disks to convert an initial layout to a target layout. Migration should be done as quickly as possible since the performance of the system will be suboptimal during migration.

Gossiping and Broadcasting

The data migration problem can be considered as a generalization of gossiping and broadcasting. The problems of gossiping and broadcasting play an important role in the design of communication protocols in various kinds of networks and have been extensively studied (see for example [6, 7] and the references therein). The gossip problem is defined as follows. There are n individuals and each individual has an item of gossip that he/she wish to communicate to everyone else. Communication is typically done in rounds, where in each round an individual may communicate with at most one other individual. Some communication models allow for the full exchange of all items of gossip known to each individual in a single round. In addition, there may be a communication graph whose edge indicates which pairs of individuals are allowed to communicate directly in each round. In the broadcast problem, one individual needs to convey an item of gossip to every other individual. The data migration problem generalizes the gossiping and broadcasting in three ways: (1) each item of gossip needs to be communicated to only a subset of individuals; (2) several items of gossip may be known to an

individual; (3) a single item of gossip can initially be shared by several individuals.

Open Problems

The data migration problem is NP-hard by reduction from the edge coloring problem. However, no inapproximability results are known for the problem. As the current best approximation factor is relatively high $(6.5 + o(1))$, it is an interesting open problem to narrow the gap between the approximation guarantee and the inapproximability.

Another open problem is to combine data placement and migration problems. This question was studied by Khuller et al. [9]. Given the initial layout and the new demand pattern, their goal was to find a set of data migrations that can be performed in a specific number of rounds and gives the best possible layout to the current demand pattern. They showed that even one-round migration is NP-hard and presented a heuristic algorithm for the one-round migration problem. The experiments showed that performing a few rounds of one-round migration consecutively works well in practice. Obtaining nontrivial approximation algorithms for this problem would be interesting future work.

Data migration in a heterogeneous storage system is another interesting direction for future research. Most research on data migration has focused mainly on homogeneous storage systems, assuming that disks have the same fixed capabilities and the network connections are of the same fixed bandwidth. In practice, however, large-scale storage systems may be heterogenous. For instance, disks tend to have heterogeneous capabilities as they are added over time owing to increasing demand for storage capacity. Lu et al. [13] studied the case when disks have variable bandwidth owing to the loads on different disks. They used a control-theoretic approach to generate adaptive rates of data migrations which minimize the degradation of the quality of the service. The algorithm reduces the latency experienced by clients significantly compared with the previous schemes. However, no theoretical bounds on the efficiency of data migrations were provided. Coffman et al. [2] studied the case when each disk i can handle p_i transfers simultaneously and provided approximation algorithms. Some papers [2, 12] considered the case when the lengths of data items are heterogenous (but the system is homogeneous), and present approximation algorithms for the problem.

Experimental Results

Golubchik et al. [3] conducted an extensive study of the performance of data migration algorithms under different changes in user-access patterns. They compared the 9.5-approximation [11] and several other heuristic algorithms. Some of these heuristic algorithms cannot provide constant approximation guarantees, while for some of the algorithms no approximation guarantees are known. Although the worst-case performance of the algorithm by Khuller et al. [11] is 9.5, in the experiments the number of rounds required was less than 3.25 times the lower bound.

They also introduced the *correspondence problem*, in which a matching between disks in the initial layout with disks in the target layout is computed so as to minimize changes. A good solution to the correspondence problem can improve the performance of the data migration algorithms by a factor of 4.4 in their experiments, relative to a bad solution.

URL to Code

http://www.cs.umd.edu/projects/smart/data-migration/

Cross-References

▸ Broadcasting in Geometric Radio Networks
▸ Deterministic Broadcasting in Radio Networks

Recommended Reading

A special case of the data migration problem was studied by Anderson et al. [1] and Hallet al. [5].

They assumed that a data transfer graph is given, in which a node corresponds to each disk and a directed edge corresponds to each move operation that is specified (the creation of new copies of data items is not allowed). Computing a data movement schedule is exactly the problem of edge-coloring the transfer graph. Algorithms for edge-coloring multigraphs can now be applied to produce a migration schedule since each color class represents a matching in the graph that can be scheduled simultaneously. Computing a solution with the minimum number of rounds is NP-hard, but several good approximation algorithms are available for edge coloring. With space constraints on the disk, the problem becomes more challenging. Hall et al. [5] showed that with the assumption that each disk has one spare unit of storage, very good constant factor approximations can be developed. The algorithms use at most $4\lceil\Delta/4\rceil$ colors with at most $n/3$ bypass nodes, or at most $6\lceil\Delta/4\rceil$ colors without bypass nodes.

Most of the results on the data migration problem deal with the half-duplex model. Another interesting communication model is the full-duplex model where each disk can act as a sender and a receiver in each round for a single item. There is a $(4 + o(1))$-approximation algorithm for the full-duplex model [10].

1. Anderson E, Hall J, Hartline J, Hobbes M, Karlin A, Saia J, Swaminathan R, Wilkes J (2001) An experimental study of data migration algorithms. In: Workshop on algorithm engineering
2. Coffman E, Garey M, Jr, Johnson D, Lapaugh A (1985) Scheduling file transfers. SIAM J Comput 14(3):744–780
3. Golubchik L, Khuller S, Kim Y, Shargorodskaya S, Wan Y (2004) Data migration on parallel disks. In: 12th annual european symposium on algorithms (ESA)
4. Golubchik L, Khanna S, Khuller S, Thurimella R, Zhu A (2000) Approximation algorithms for data placement on parallel disks. In: Symposium on discrete algorithms. Society for industrial and applied mathematics, Philadelphia, pp 223–232
5. Hall J, Hartline J, Karlin A, Saia J, Wilkes J (2001) On algorithms for efficient data migration. In: SODA. Society for industrial and applied mathematics, Philadelphia, pp 620–629
6. Hedetniemi SM, Hedetniemi ST, Liestman A (1988) A survey of gossiping and broadcasting in communication networks. Networks 18: 129–134
7. Hromkovic J, Klasing R, Monien B, Peine R (1996) Dissemination of information in interconnection networks (broadcasting and gossiping). In: Du DZ, Hsu F (eds) Combinatorial network theory. Kluwer Academic, Dordrecht, pp 125–212
8. Kashyap S, Khuller S (2003) Algorithms for non-uniform size data placement on parallel disks. In: Conference on FST&TCS conference. LNCS, vol 2914. Springer, Heidelberg, pp 265–276
9. Kashyap S, Khuller S, Wan Y-C, Golubchik L (2006) Fast reconfiguration of data placement in parallel disks. In: Workshop on algorithm engineering and experiments
10. Khuller S, Kim Y, Malekian A (2006) Improved algorithms for data migration. In: 9th international workshop on approximation algorithms for combinatorial optimization problems
11. Khuller S, Kim Y, Wan Y-C (2004) Algorithms for data migration with cloning. SIAM J Comput 33(2):448–461
12. Kim Y-A (2005) Data migration to minimize the average completion time. J Algorithms 55:42–57
13. Lu C, Alvarez GA, Wilkes J (2002) Aqueduct:online datamigration with performance guarantees. In: Proceedings of the conference on file and storage technologies
14. Shachnai H, Tamir T (2001) Polynomial time approximation schemes for class-constrained packing problems. J Sched 4(6):313–338
15. Shachnai H, Tamir T (2001) On two class-constrained versions of the multiple knapsack problem. Algorithmica 29(3):442–467
16. Shannon CE (1949) A theorem on colouring lines of a network. J Math Phys 28:148–151
17. Shmoys DB, Tardos E (1993) An approximation algorithm for the generalized assignment problem. Math Program 62(3):461–474
18. Vizing VG (1964) On an estimate of the chromatic class of a p-graph (Russian). Diskret Analiz 3:25–30

Data Reduction for Domination in Graphs

Rolf Niedermeier
Department of Mathematics and Computer Science, University of Jena, Jena, Germany
Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Berlin, Germany

Keywords

Dominating set; Kernelization; Reduction to a problem kernel

Years and Authors of Summarized Original Work

2004; Alber, Fellows, Niedermeier

Problem Definition

The NP-complete DOMINATING SET problem is a notoriously hard problem:

Problem 1 (Dominating Set)
INPUT: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
QUESTION: Is there an $S \subseteq V$ with $|S| \leq k$ such that every vertex $v \in V$ is contained in S or has at least one neighbor in S?

For instance, for an n-vertex graph its optimization version is known to be polynomial-time approximable only up to a factor of $\Theta(\log n)$ unless some standard complexity-theoretic assumptions fail [9]. In terms of parametrized complexity, the problem is shown to be W[2]-complete [8]. Although still NP-complete when restricted to planar graphs, the situation much improves here. In her seminal work, Baker showed that there is an efficient polynomial-time approximation scheme (PTAS) [6], and the problem also becomes fixed-parameter tractable [2, 4] when restricted to planar graphs. In particular, the problem becomes accessible to fairly effective data reduction rules and a kernelization result (see [16] for a general description of data reduction and kernelization) can be proven. This is the subject of this entry.

Key Results

The key idea behind the data reduction is preprocessing based on locally acting simplification rules. Exemplary, here we describe a rule where the local neighborhood of each graph vertex is considered. To this end, we need the following definitions.

We partition the neighborhood $N(v)$ of an arbitrary vertex $v \in V$ in the input graph into three disjoint sets $N_1(v)$, $N_2(v)$, and $N_3(v)$ depending on local neighborhood structure. More specifically, we define

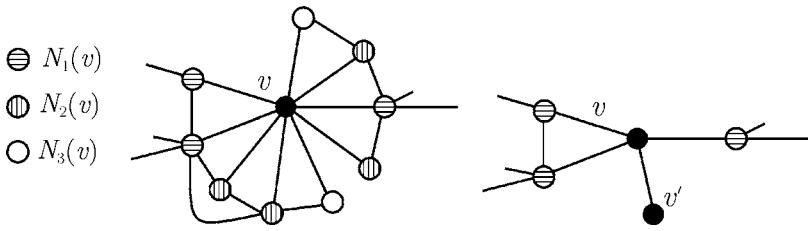
- $N_1(v)$ to contain all neighbors of v that have edges to vertices that are not neighbors of v;
- $N_2(v)$ to contain all vertices from $N(v) \setminus N_1(v)$ that have edges to at least one vertex from $N_1(v)$;
- $N_3(v)$ to contain all neighbors of v that are neither in $N_1(v)$ nor in $N_2(v)$.

An example which illustrates such a partitioning is given in Fig. 1 (left-hand side). A helpful and intuitive interpretation of the partition is to see vertices in $N_1(v)$ as *exits* because they have direct connections to the world outside the closed neighborhood of v, vertices in $N_2(v)$ as *guards* because they have direct connections to exits, and vertices in $N_3(v)$ as *prisoners* because they do not see the world outside $\{v\} \cup N(v)$.

Now consider a vertex $w \in N_3(v)$. Such a vertex only has neighbors in $\{v\} \cup N_2(v) \cup N_3(v)$. Hence, to dominate w, at least one vertex of $\{v\} \cup N_2(v) \cup N_3(v)$ *must* be contained in a dominating set for the input graph. Since v can dominate all vertices that would be dominated by choosing a vertex from $N_2(v) \cup N_3(v)$ into the dominating set, we obtain the following data reduction rule.

If $N_3(v) \neq \emptyset$ for some vertex v, then remove
$N_2(v)$ and $N_3(v)$ from G
and add a new vertex v'
with the edge $\{v, v'\}$ to G.

Note that the new vertex v' can be considered as a "gadget vertex" that "enforces" v to be chosen into the dominating set. It is not hard to verify the correctness of this rule, that is, the original graph has a dominating set of size k iff the reduced graph has a dominating set of size k. Clearly, the data reduction can be executed in polynomial time [5]. Note, however, that there are particular "diamond" structures that are not amenable to this reduction rule. Hence, a second, somewhat more complicated rule based on considering the joint neighborhood of *two* vertices has been introduced [5].



Data Reduction for Domination in Graphs, Fig. 1 The *left-hand side* shows the partitioning of the neighborhood of a single vertex v. The *right-hand side* shows the result of applying the presented data reduction rule to this particular (sub)graph

Altogether, the following core result could be shown [5].

Theorem 1 *A planar graph $G = (V, E)$ can be reduced in polynomial time to a planar graph $G' = (V', E')$ such that G has a dominating set of size k iff G' has a dominating set of size k and $|V'| = O(k)$.*

In other words, the theorem states that the DOMINATING SET in planar graphs has a linear-size problem kernel. The upper bound on $|V'|$ was first shown to be $335k$ [5] and was then further improved to $67k$ [7]. Moreover, the results can be extended to graphs of bounded genus [10]. In addition, similar results (linear kernelization) have been recently obtained for the FULL-DEGREE SPANNING TREE problem in planar graphs [13]. Very recently, these results have been generalized into a methodological framework [12].

Applications

DOMINATING SET is considered to be one of the most central graph problems [14, 15]. Its applications range from facility location to bioinformatics.

Open Problems

The best lower bound for the size of a problem kernel for DOMINATING SET in planar graphs is $2k$ [7]. Thus, there is quite a gap between known upper and lower bounds. In addition, there have been some considerations concerning a generalization of the above-discussed data reduction rules [3]. To what extent such extensions are of practical use remains to be explored. Finally, a study of deeper connections between Baker's PTAS results [6] and linear kernelization results for DOMINATING SET in planar graphs seems to be worthwhile for future research. Links concerning the class of problems amenable to both approaches have been detected recently [12]. The research field of data reduction and problem kernelization as a whole together with its challenges is discussed in a recent survey [11].

Experimental Results

The above-described theoretical work has been accompanied by experimental investigations on synthetic as well as real-world data [1]. The results have been encouraging in general. However, note that grid structures seem to be a hard case where the data reduction rules remained largely ineffective.

Cross-References

▸ Connected Dominating Set

Recommended Reading

1. Alber J, Betzler N, Niedermeier R (2006) Experiments on data reduction for optimal domination in networks. Ann Oper Res 146(1):105–117
2. Alber J, Bodlaender HL, Fernau H, Kloks T, Niedermeier R (2002) Fixed parameter algorithms for Dominating Set and related problems on planar graphs. Algorithmica 33(4):461–493

3. Alber J, Dorn B, Niedermeier R (2006) A general data reduction scheme for domination in graphs. In: Proceedings of 32nd SOFSEM. LNCS, vol 3831. Springer, Berlin, pp 137–147
4. Alber J, Fan H, Fellows MR, Fernau H, Niedermeier R, Rosamond F, Stege U (2005) A refined search tree technique for dominating Set on planar graphs. J Comput Syst Sci 71(4):385–405
5. Alber J, Fellows MR, Niedermeier R (2004) Polynomial time data reduction for Dominating Set. J ACM 51(3):363–384
6. Baker BS (1994) Approximation algorithms for NP-complete problems on planar graphs. J ACM 41(1):153–180
7. Chen J, Fernau H, Kanj IA, Xia G (2007) Parametric duality and kernelization: lower bounds and upper bounds on kernel size. SIAM J Comput 37(4):1077–1106
8. Downey RG, Fellows MR (1999) Parameterized complexity. Springer, New York
9. Feige U (1998) A threshold of ln n for approximating set cover. J ACM 45(4):634–652
10. Fomin FV, Thilikos DM (2004) Fast parameterized algorithms for graphs on surfaces: linear kernel and exponential speed-up. In: Proceedings of 31st ICALP. LNCS, vol 3142. Springer, Berlin, pp 581–592
11. Guo J, Niedermeier R (2007) Invitation to data reduction and problemkernelization. ACM SIGACT News 38(1):31–45
12. Guo J, Niedermeier R (2007) Linear problem kernels for NPhard problems on planar graphs. In: Proceedings of 34th ICALP. LNCS, vol 4596. Springer, Berlin, pp 375–386
13. Guo J, Niedermeier R, Wernicke S (2006) Fixed-parameter tractability results for full-degree spanning tree and its dual. In: Proceedings of 2nd IWPEC. LNCS, vol 4196. Springer, Berlin, pp 203–214
14. Haynes TW, Hedetniemi ST, Slater PJ (1998) Domination in graphs: advanced topics. Pure and applied mathematics, vol 209. Marcel Dekker, New York
15. Haynes TW, Hedetniemi ST, Slater PJ (1998) Fundamentals of domination in graphs. Pure and applied mathematics, vol 208. Marcel Dekker, New York
16. Niedermeier R (2006) Invitation to fixed-parameter algorithms. Oxford University Press, New York

Data Stream Verification

Justin Thaler
Yahoo! Labs, New York, NY, USA

Keywords

Delegation; Interactive proofs; Streaming algorithms; Verification

Years and Authors of Summarized Original Work

2008; Goldwasser, Kalai, Rothblum
2011; Chung, Kalai, Liu, Raz
2012; Cormode, Thaler, Yi
2013; Gur, Raz
2014; Chakrabarti, Cormode, McGregor, Thaler

Problem Definition

The problem is concerned with the following setting. A computationally limited client wants to compute some property of a massive input, but lacks the resources to store even a small fraction of the input, and hence cannot perform the desired computation locally. The client therefore accesses a powerful but untrusted service provider (e.g., a commercial cloud computing service), who not only performs the requested computation but also proves that the answer is correct. An array of closely related models have been introduced to capture this scenario. The following section provides a unified presentation of these models, emphasizing their common features before delineating their differences.

Streaming Verification Model

Let $\sigma = \langle a_1, a_2, \ldots, a_m \rangle$ be a data stream, where each a_i comes from a data universe $\mathcal{U}$ of size n, and let F be a function mapping data streams to a finite range $\mathcal{R}$. A stream verification protocol for F involves two parties: a *prover* $\mathcal{P}$ and a (randomized) *verifier* $\mathcal{V}$. The protocol consists of two stages: a stream observation stage and a proof verification stage.

In the stream observation stage, $\mathcal{V}$ processes the stream σ, subject to the standard constraints of the data-stream model, i.e., sequential access to σ and limited memory. In the proof verification stage, $\mathcal{V}$ and $\mathcal{P}$ exchange a sequence of one or more messages, and afterward $\mathcal{V}$ outputs a value b. $\mathcal{V}$ is allowed to output a special symbol $\perp$ indicating a rejection of $\mathcal{P}$'s claims. Formally, $\mathcal{V}$ constitutes a stream verification protocol if the following two properties are satisfied:

- **Completeness**: There is some prover strategy $\mathcal{P}$ such that, for all streams σ, the probability that $\mathcal{V}$ outputs $F(\sigma)$ after interacting with $\mathcal{P}$ is at least $2/3$.
- **Soundness**: For all streams σ and all prover strategies $\mathcal{P}$, the probability that $\mathcal{V}$ outputs a value not in $\{F(x), \perp\}$ after interacting with $\mathcal{P}$ is at most $\epsilon \leq 1/3$.

Here, the probabilities are taken over $\mathcal{V}$'s internal randomness. The constants $2/3$ and $1/3$ are not essential and are chosen by convention. The parameter ϵ is referred to as the *soundness error* of the protocol.

Costs

There are five primary costs in any stream verification protocol: (1) $\mathcal{V}$'s space usage, (2) the total communication cost, (3) $\mathcal{V}$'s runtime, (4) $\mathcal{P}$'s runtime, and (5) the number of messages exchanged.

Differences Between Models

There are three primary differences between the various models of stream verification that have been put forth in the literature. The first is whether the soundness condition is required to hold against all cheating provers (such protocols are called *information-theoretically* or *statistically* sound), or only against cheating provers that run in polynomial time (such protocols are called *computationally* sound). The second is the amount and format of the interaction allowed between $\mathcal{P}$ and $\mathcal{V}$. The third is the temporal relationship between the stream observation and proof verification stage – in particular, several models permit $\mathcal{P}$ and $\mathcal{V}$ to exchange messages before and during the stream observation stage and sometimes permit the prover's messages to depend on parts of the data stream that $\mathcal{V}$ has not yet seen. In general, more permissive models allow a larger class of problems to be solved efficiently, but may yield protocols that are less realistic.

Summary of Models

The *annotated data streaming* (ADS) model [3] is noninteractive: $\mathcal{P}$ is permitted to send a single message to $\mathcal{V}$, with no communication allowed in the reverse direction. Technically, this model permits the contents of $\mathcal{P}$'s message to be interleaved with the stream, in which case each bit of $\mathcal{P}$'s message may be viewed as an "annotation" associated with a particular stream update. However, for most ADS protocols that have been developed, $\mathcal{P}$'s message can be sent after the stream observation phase. There are two kinds of ADS protocols: *prescient* protocols, in which the annotation sent at any given time can depend on parts of the data stream that $\mathcal{V}$ has not yet seen, and *online* protocols, which disallow this kind of dependence.

Streaming interactive proofs (SIPs) extend the ADS model to allow the prover and verifier to exchange many messages [6]. The *Arthur–Merlin streaming model* [10] is equivalent to a restricted class of SIPs, in which $\mathcal{V}$ is only allowed to send a single message to $\mathcal{P}$ (which must consist entirely of random coin tosses, in analogy with the classical complexity class AM), before receiving $\mathcal{P}$'s reply. The *streaming delegation model* [5] corresponds to SIPs that only satisfy computational, rather than information-theoretic, soundness.

Key Results

Obtaining exact answers even for basic problems in the standard data streaming model is impossible using $o(n)$ space. In contrast, stream verification protocols with $o(n)$ space and communication costs have been developed for (exactly solving) a wide variety of problems. Many of these protocols have adapted powerful algebraic techniques originally developed in the classical literature on interactive proofs, particularly the *sum-check* protocol of Lund et al. [14]. All of the protocols described here apply even to streams in the strict turnstile update model, where universe items can be deleted as well as inserted.

Annotated Data Streams

Chakrabarti et al. [3] showed that prescient ADS protocols can be exponentially more powerful than online ones for some problems. For example,

there is a prescient ADS protocol with logarithmic space and communication costs for computing the median of a sequence of numbers: $\mathcal{P}$ sends $\mathcal{V}$ the claimed median τ at the start of the stream, and while observing the stream, $\mathcal{V}$ checks that $|\{j : a_j < \tau\}| \leq m/2$, and $|\{j : a_j > \tau\}| \leq m/2$, which can be done using an $O(\log m)$-bit counter. Meanwhile, [3] proved that any online protocol for MEDIAN with communication cost h and space cost v requires $h \cdot v = \Omega(n)$ and gave an online ADS protocol achieving these communication–space trade-offs up to logarithmic factors.

Chakrabarti et al. [3] also gave online ADS protocols achieving identical trade-offs between space and communication costs for problems including FREQUENCY MOMENTS and FREQUENT ITEMS and used a lower bound due to Klauck [11] on the *Merlin–Arthur communication complexity* of the SET-DISJOINTNESS function to show that these trade-offs are optimal for these problems even among prescient protocols. Subsequent work gave similarly optimal online ADS protocols for several more problems, including maximum matching and counting triangles in graphs and matrix-vector multiplication [8, 18]. Chakrabarti et al. [2] gave optimized protocols for streams whose length m is much smaller than the universe size n.

Streaming Interactive Proofs

Cormode, Thaler, and Yi [6] showed that several general protocols from the classical literature on interactive proofs can be simulated in the SIP model. In particular, this includes a powerful, general-purpose protocol due to Goldwasser, Kalai, and Rothblum [9] (henceforth, the GKR protocol). Given any problem computed by an arithmetic or Boolean circuit of polynomial size and polylogarithmic depth, the GKR protocol requires only polylogarithmic space and communication while using polylogarithmic rounds of verifier–prover interaction. This yields SIPs for exactly solving many basic streaming problems with polylogarithmic space and communication costs, including FREQUENCY MOMENTS, FREQUENT ITEMS, and GRAPH CONNECTIVITY. Optimized protocols for specific problems, including FREQUENCY MOMENTS (see the detailed example below), were also presented.

Chakrabarti et al. [4] give *constant-round* SIPs with logarithmic space and communication costs for many problems, including INDEX, RANGE-COUNTING, and NEAREST-NEIGHBOR SEARCH. Gur and Raz [10] gave an Arthur–Merlin streaming protocol for the DISTINCT ELEMENTS problem with communication cost $\tilde{O}(h)$ space cost $\tilde{O}(v)$ for any h, v satisfying $h \cdot v \geq n$. Klauck and Prakash [13] extended this protocol to give an SIP for Distinct Elements with polylogarithmic space and communication costs and logarithmically many rounds of prover–verifier interaction.

Computationally Sound Protocols

Computationally sound protocols may achieve properties that are unattainable in the information-theoretic setting. For example, they typically achieve reusability, allowing the verifier to use the same randomness to answer many queries. In contrast, most SIPs only support "one-shot" queries, because they require the verifier to reveal secret randomness to the prover over the course of the protocol. Chung et al. [5] combined the GKR protocol with fully homomorphic encryption (FHE) to give reusable two-message protocols with polylogarithmic space and communication costs for any problem in the complexity class NC. They also gave reusable four-message protocols with polylogarithimic space and communication costs for any problem in the complexity class P. Papamanthou et al. [15] gave improved protocols for a class of low-complexity queries including point queries and range search: these protocols avoid the use of FHE and allow the prover to answer such queries in polylogarithmic time. (In contrast, protocols based on the GKR protocol [5, 6] require the prover to spend time quasilinear in the size of the data stream after receiving a query, even if the answer itself can be computed in sublinear time.)

Implementations

Implementations of the GKR protocol were provided in [7, 17]. Cormode, Mitzenmacher, and Thaler [7] also provided optimized

implementations of several ADS protocols from [3, 8]. Thaler et al. [19] provided parallelized implementations using Graphics Processing Units.

Detailed Example

The sum-check protocol can be directly applied to give an SIP for the kth frequency moment problem with $\log n$ rounds of prover–verifier iteration and $O(\log^2(n))$ space and communication costs. The sum-check protocol is described in Fig. 1.

Properties and Costs of the Sum-Check Protocol

The sum-check protocol satisfies perfect completeness and has soundness error $\epsilon \leq \deg(g)/|\mathbb{F}|$, where $\deg(g)$ denotes the total degree of g [14]. There is one round of prover–verifier interaction in the sum-check protocol for each of the v variables of g, and the total communication is $O(\deg(g))$ field elements.

Note that as described in Fig. 1, the sum-check protocol assumes that the verifier has oracle access to g. However, this will not be the case in applications, as g will ultimately be a polynomial that depends on the input data stream.

The SIP for Frequency Moments

In the kth frequency moment problem, the goal is to output $\sum_{i \in \mathcal{U}} f_i^k$, where f_i is the number of times item i appears in the data stream σ. For a vector $\mathbf{i} = (i_1, \ldots, i_{\log n}) \in \{0,1\}^{\log n}$, let $\chi_{\mathbf{i}}(x_1, \ldots, x_{\log n}) = \prod_{k=1}^{\log n} \chi_{i_k}(x_k)$, where $\chi_0(x_k) = 1 - x_k$ and $\chi_1(x_k) = x_k$. $\chi_{\mathbf{i}}$ is the unique multilinear polynomial that maps $\mathbf{i} \in \{0,1\}^{\log n}$ to 1 and all other values in $\{0,1\}^{\log n}$ to 0, and it is referred to as the *multilinear extension* of $\mathbf{i}$.

For each $i \in \mathcal{U}$, associate i with a vector $\mathbf{i} \in \{0,1\}^{\log n}$ in the natural way, and let $\mathbb{F}$ be a finite field with $n^k \leq |\mathbb{F}| \leq 4 \cdot n^k$. Define the polynomial $\hat{f}: \mathbb{F}^{\log n} \to \mathbb{F}$ via

Input: $\mathcal{V}$ is given oracle access to a v-variate polynomial g over finite field $\mathbb{F}$ and an $H \in \mathbb{F}$.
Goal: Determine whether $H = \sum_{(x_1,\ldots,x_v) \in \{0,1\}^v} g(x_1,\ldots,x_v)$.

- In the first round, $\mathcal{P}$ computes the univariate polynomial

$$g_1(X_1) := \sum_{x_2,\ldots,x_v \in \{0,1\}^{v-1}} g(X_1, x_2, \ldots, x_v),$$

  and sends g_1 to $\mathcal{V}$. $\mathcal{V}$ checks that g_1 is a univariate polynomial of degree at most $\deg_1(g)$, and that $H = g_1(0) + g_1(1)$, rejecting if not.
- $\mathcal{V}$ chooses a random element $r_1 \in \mathbb{F}$, and sends r_1 to $\mathcal{P}$.
- In the jth round, for $1 < j < v$, $\mathcal{P}$ sends to $\mathcal{V}$ the univariate polynomial

$$g_j(X_j) = \sum_{(x_{j+1},\ldots,x_v) \in \{0,1\}^{v-j}} g(r_1, \ldots, r_{j-1}, X_j, x_{j+1}, \ldots, x_v).$$

  $\mathcal{V}$ checks that g_j is a univariate polynomial of degree at most $\deg_j(g)$, and that $g_{j-1}(r_{j-1}) = g_j(0) + g_j(1)$, rejecting if not.
- $\mathcal{V}$ chooses a random element $r_j \in \mathbb{F}$, and sends r_j to $\mathcal{P}$.
- In round v, $\mathcal{P}$ sends to $\mathcal{V}$ the univariate polynomial

$$g_v(X_v) = g(r_1, \ldots, r_{v-1}, X_v).$$

  $\mathcal{V}$ checks that g_v is a univariate polynomial of degree at most $\deg_v(g)$, rejecting if not.
- $\mathcal{V}$ chooses a random element $r_v \in \mathbb{F}$ and evaluates $g(r_1, \ldots, r_v)$ with a single oracle query to g. $\mathcal{V}$ checks that $g_v(r_v) = g(r_1, \ldots, r_v)$, rejecting if not.
- If $\mathcal{V}$ has not yet rejected, $\mathcal{V}$ halts and accepts.

Data Stream Verification, Fig. 1 Description of the sum-check protocol. $\deg_i(g)$ denotes the degree of g in the ith variable

$$\hat{f} = \sum_{\mathbf{i} \in \{0,1\}^{\log n}} f_i \cdot \chi_{\mathbf{i}}. \qquad (1)$$

Note that $\hat{f}$ is the unique multilinear polynomial satisfying the property that $\hat{f}(\mathbf{i}) = f_i$ for all $\mathbf{i} \in \{0,1\}^{\log n}$.

The kth frequency moment of σ is equal to

$$\sum_{\mathbf{i} \in \{0,1\}^{\log n}} f_i^k = \sum_{\mathbf{i} \in \{0,1\}^{\log n}} (\hat{f}^k)(\mathbf{i}) \,.$$

Hence, in order to compute the kth frequency moment of σ, it suffices to apply the sum-check protocol to the polynomial $g = \hat{f}^k$. This requires $\log n$ rounds of prover–verifier interaction, and since the total degree of $\hat{f}^k$ is $k \cdot \log n$, the total communication cost is $O(k \log n)$ field elements, which require $O(k^2 \log^2 n)$ total bits to specify.

At the end of the sum-check protocol, $\mathcal{V}$ must compute

$$g(r_1, \ldots, r_{\log n}) = (\hat{f}^k)(r_1, \ldots, r_{\log n})$$

for randomly chosen $(r_1, \ldots, r_{\log n}) \in \mathbb{F}^{\log n}$. It suffices for $\mathcal{V}$ to evaluate $z := \hat{f}(r_1, \ldots, r_{\log n})$, since $(\hat{f}^k)(r_1, \ldots, r_{\log n}) = z^k$. The following lemma establishes that $\mathcal{V}$ can evaluate z with a single pass over σ, while storing $O(\log n)$ field elements.

Lemma 1 *$\mathcal{V}$ can compute $z = \hat{f}(r_1, \ldots, r_{\log n})$ with a single streaming pass over σ, while storing $O(\log n)$ field elements.*

Proof Given any stream update $a_j \in \mathcal{U}$, let $\mathbf{a_j}$ denote the binary vector associated with a_j. It follows from Eq. (1) that $\hat{f}(r_1, \ldots, r_{\log n}) = \sum_{j=1}^{m} \chi_{\mathbf{a_j}}(r_1, \ldots, r_{\log n})$. Thus, $\mathcal{V}$ can compute $\hat{f}(r_1, \ldots, r_{\log n})$ incrementally from the raw stream by initializing $\hat{f}(r_1, \ldots, r_{\log n}) \leftarrow 0$ and processing each update a_j via

$$\hat{f}(r_1, \ldots, r_{\log n}) \leftarrow \hat{f}(r_1, \ldots, r_{\log n}) + \chi_{\mathbf{a_j}}(r_1, \ldots, r_{\log n}).$$

$\mathcal{V}$ only needs to store $(r_1, \ldots, r_{\log n})$ and $\hat{f}(r_1, \ldots, r_{\log n})$, which is $O(\log n)$ field elements in total.

Open Problems

- For several functions $F: \{0,1\}^n \to \{0,1\}$, it is known that any online ADS protocol for F with communication cost h and space cost v requires $h \cdot v = \Omega(n)$. This lower bound is tight in many cases, such as for the INDEX function [3]. However, it is open to exhibit a function that cannot be computed by any online ADS protocol with communication and space costs both bounded above by h, for some $h = \omega(n^{1/2})$.
- Two-message online SIP protocols with logarithmic space and communication costs are known for several functions, including the INDEX function [4]. It is also known that *existing techniques* cannot yield 2- or 3-message online SIPs of polylogarithmic cost for the MEDIAN or FREQUENCY MOMENT problems. However, it is open to exhibit a function $F: \{0,1\}^n \to \{0,1\}$ that cannot be computed by any online two-message SIP with communication and space costs both bounded above by h, for some $h = \omega(\log n)$.

Cross-References

▸ Communication Complexity

Recommended Reading

1. Aaronson S, Wigderson A (2009) Algebrization: a new barrier in complexity theory. ACM Trans Comput Theory 1(1):2:1–2:54
2. Chakrabarti A, Cormode G, Goyal N, Thaler J (2014) Annotations for sparse data streams. In: Chekuri C (ed) SODA, Portland. SIAM, Philadelphia, pp 687–706
3. Chakrabarti A, Cormode G, McGregor A, Thaler J (2014) Annotations in data streams. ACM Trans Algorithms 11(1):7. Preliminary version appeared in ICALP, 2009
4. Chakrabarti A, Cormode G, McGregor A, Thaler J, Venkatasubramanian S (2015) Verifiable stream computation and Arthur-Merlin communication. In: 30th conference on computational complexity (CCC), Portland, 17–19 June 2015, pp. 217–243
5. Chung K-M, Kalai YT, Liu F-H, Raz R (2011) Memory delegation. In: Rogaway P (ed) CRYPTO, Santa Barbara. Volume 6841 of Lecture notes in computer science. Springer, Heidelberg/New York, pp 151–168

6. Cormode G, Thaler J, Yi K (2011) Verifying computations with streaming interactive proofs. PVLDB 5(1):25–36
7. Cormode G, Mitzenmacher M, Thaler J (2012) Practical verified computation with streaming interactive proofs. In: Goldwasser S (ed) ITCS, Cambridge, MA. ACM, pp 90–112
8. Cormode G, Mitzenmacher M, Thaler J (2013) Streaming graph computations with a helpful advisor. Algorithmica 65(2):409–442
9. Goldwasser S, Kalai YT, Rothblum GN (2008) Delegating computation: interactive proofs for muggles. In: Proceedings of the 40th annual ACM symposium on theory of computing (STOC'08), Victoria. ACM, New York, pp 113–122
10. Gur T, Raz R (2013) Arthur-Merlin streaming complexity. In: Proceedings of the 40th international colloquium on automata, languages and programming: Part I (ICALP'13). Springer, Berlin/Heidelberg
11. Klauck H (2003) Rectangle size bounds and threshold covers in communication complexity. In: IEEE conference on computational complexity, Aarhus, pp 118–134
12. Klauck H, Prakash V (2013) Streaming computations with a loquacious prover. In: Kleinberg RD (ed) ITCS, Berkeley. ACM, pp 305–320
13. Klauck H, Prakash V (2014) An improved interactive streaming algorithm for the distinct elements problem. In: Esparza J, Fraigniaud P, Husfeldt T, Koutsoupias E (eds) ICALP (1), Copenhagen. Volume 8572 of Lecture notes in computer science. Springer, Heidelberg, pp 919–930
14. Lund C, Fortnow L, Karloff H, Nisan N (1992) Algebraic methods for interactive proof systems. J ACM 39:859–868
15. Papamanthou C, Shi E, Tamassia R, Yi K (2013) Streaming authenticated data structures. In: Johansson T, Nguyen PQ (eds) EUROCRYPT, Athens. Volume 7881 of Lecture notes in computer science. Springer, Heidelberg/New York, pp 353–370
16. Schröder D, Schröder H (2012) Verifiable data streaming. In: Yu T, Danezis G, Gligor VD (eds) ACM conference on computer and communications security, Raleigh. ACM, New York, pp 953–964
17. Thaler J (2013) Time-optimal interactive proofs for circuit evaluation. In: Proceedings of the 33rd annual conference on advances in cryptology (CRYPTO'13), Santa Barbara. Springer, Berlin/Heidelberg
18. Thaler J (2014) Semi-streaming algorithms for annotated graph streams. Electron Colloq Comput Complex 21:90
19. Thaler J, Roberts M, Mitzenmacher M, Pfister H (2012) Verifying computations with massively parallel interactive proofs. In: HotCloud, Boston
20. Vu V, Setty S, Blumberg AJ, Walfish M (2013) A hybrid architecture for interactive verifiable computation. In: IEEE symposium on security and privacy, San Francisco

3D Conforming Delaunay Triangulation

Siu-Wing Cheng
Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China

Keywords

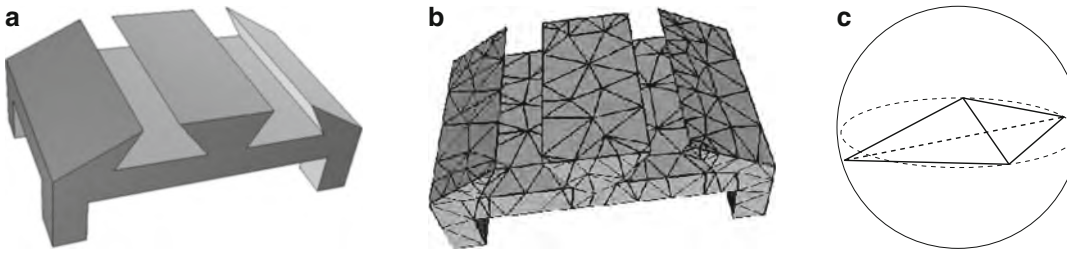
Delaunay refinement; Protecting ball; Radius-edge ratio; Weighted Delaunay triangulation

Years and Authors of Summarized Original Work

1998; Shewchuk
2004; Cohen-Steiner, de Verdière, Yvinec
2006; Cheng, Poon
2012; Cheng, Dey, Shewchuk

Problem Definition

A three-dimensional domain with piecewise linear boundary elements can be represented as a *piecewise linear complex* (PLC) of *linear cells* – vertices, edges, polygons, and polyhedra – that satisfy the following properties [4]. First, no vertex lies in the interior of an edge and every two edges are interior-disjoint. Second, the boundary of a polygon or polyhedra are union of cells in the PLC. Third, if two cells f and g intersect, the intersection is a union of cells in the PLC with dimensions lower than f or g. A triangulation of an input PLC is *conforming* if every edge and polygon appear as a union of segments and triangles in the triangulation. Additional Steiner vertices are often necessary. The 3D conforming Delaunay triangulation problem is to construct a triangulation of an input PLC that is both conforming and Delaunay. Figure 1a, b shows an input PLC and its conforming Delaunay triangulation. In many applications, it is often desired that the triangulation is not unnecessarily dense and the resulting tetrahedra are of *bounded*



3D Conforming Delaunay Triangulation, Fig. 1 A PLC and its conforming Delaunay triangulation in (**a**) and (**b**). A sliver with negligible volume and edge lengths similar to its circumradius in (**c**).

aspect ratio. Another popular shape measure is the *radius-edge ratio*, which is the ratio of the circumradius of the tetrahedron to its shortest edge length. Tetrahedra with bounded radius-edge ratio may still have negligible volume, and they are known as *slivers*. Figure 1c shows a sliver.

Key Results

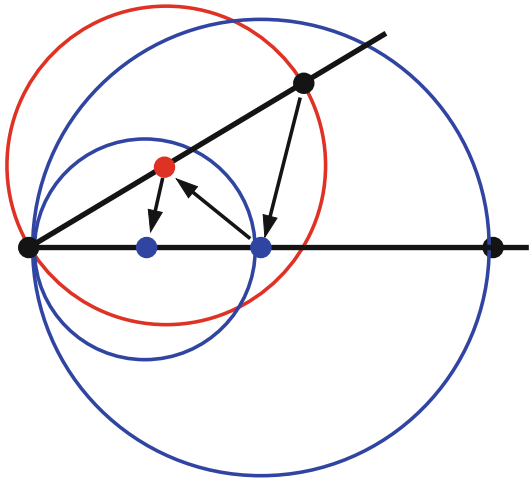
Since the Delaunayhood of edges and triangles are guaranteed by the emptiness of their circumspheres, one would imagine that a conforming Delaunay triangulation can be obtained by sprinkling Steiner vertices on the input edges and polygons. Indeed, Murphy, Mount, and Gable [6] showed a way to do this, but the resulting triangulation is very dense, and no shape guarantee is offered.

Shewchuk [9] gave the first algorithm that offers shape guarantee for PLCs in which two adjoining elements do not make an acute angle. (The exact requirement is more general and is called the projection condition [4, 9].) The algorithm is a generalization of Ruppert's Delaunay refinement algorithm in the plane [8]. An initial Delaunay triangulation is formed using the input vertices, and then Steiner vertices are added incrementally. Boundary conformity takes precedence. Therefore, whenever a segment on an edge or a triangle on a polygon has a nonempty diametric ball (the ball enclosed by the smallest circumsphere), that segment or triangle is split by inserting the center of its diametric ball. A Delaunay tetrahedron with radius-edge ratio larger than a prescribed constant $\rho > 2$ is split by inserting its circumcenter. However, if this circumcenter lies inside the diametric ball of a segment or a triangle, then the insertion is aborted and that segment or triangle is split instead. Similarly, the insertion of a triangle's diametric ball center is aborted if it lies in the diametric ball of a segment, and that segment is split instead. The following theorem states the main result.

Theorem 1 ([4, 9]) *Let ρ be a constant greater than 2. Let P be a PLC with no acute input angle. A conforming Delaunay triangulation of P can be obtained by Delaunay refinement and all tetrahedra obtained have radius-edge ratio at most ρ.*

In the presence of acute angles, the splitting of segments and triangles may lead to an infinite loop as illustrated in Fig. 2. Notice that the Steiner vertices inserted are approaching the input vertex in Fig. 2. In the plane, Ruppert [8] proposed a fix: place some protecting circles centered at the input vertices, disallow the insertion of Steiner vertices inside these protecting circles, and triangulate the inside of these protecting circles using a separate mechanism. A key change is that if a Steiner vertex to be inserted is too close to an arc on a protecting circle, then the insertion of the Steiner vertex is aborted and the circular arc is split by inserting its midpoint. This is analogous to the splitting of segments.

Cohen-Steiner, de Verdière, and Yvinec [5] generalized this idea partly to three dimensions. They proposed to place protecting balls centered at the input vertices as well as at some appropriate points in the interior of input edges. These protecting balls cover all input vertices and edges. The intersection between a protecting ball



3D Conforming Delaunay Triangulation, Fig. 2 The midpoint of the segment with the largest diametric ball triggers the splitting of the segment with the second largest diametric ball, which in turn triggers the splitting of the segment with the smallest diametric ball. This may go on indefinitely

boundary and an input polygon is analogous to a protecting circle in 2D. Therefore, when we want to insert Steiner vertices in a polygon f to recover the Delaunay triangles on f, if such a Steiner vertex v is too close to an arc α at the intersection of f and some protecting ball boundary, the insertion of v is aborted and α is split instead. The portions of polygons inside the protecting balls are triangulated using a separate mechanism. If a tetrahedron τ has large radius-edge ratio but its circumcenter lies inside some protecting ball, then τ is just kept in the triangulation. As a result, no shape guarantee is offered.

Theorem 2 ([5]) *There is a Delaunay refinement algorithm that constructs a conforming Delaunay triangulation of any input PLC.*

Cheng and Poon [2] extended Delaunay refinement by observing that segments, circular arcs, triangles, and spherical triangles can all be handled in a uniform way.

Let $\mathcal{B}$ be the union of protecting balls with centers at the input vertices and interior of input edges. Let B be a protecting ball. Let ∂ denote the boundary operator. For every input polygon f, the Steiner vertices on $f \cap B \cap \partial\mathcal{B}$ divide $f \cap B \cap \partial\mathcal{B}$ into circular arcs. For every protecting B, the projection of the convex hull of the Steiner vertices on B onto ∂B divides ∂B into some spherical triangles. The diametric ball of a segment or triangle can be viewed as the circumscribing ball whose boundary intersects the affine hull of the segment or triangle at right angle. Analogously, the "diametric ball" of a circular arc in ∂B or a spherical triangle with vertices in ∂B can be defined as the circumscribing ball whose boundary intersects B at right angle. If a Steiner vertex to be inserted lies inside this "diametric ball," the insertion is aborted and the circular arc or spherical triangle is split instead. A circular arc is split by inserting its midpoint. A spherical triangle is split by inserting the intersection point between ∂B and the line segment joining the centers of B and the "diametric ball." The last ingredient is that for every pair of protecting balls whose centers are adjacent on an input edge, their boundaries should intersect at right angle. That is, the protecting ball B' adjacent to B serves as the "diametric ball" of the spherical triangles that have vertices on the circle $\partial B \cap \partial B'$.

Theorem 3 ([2]) *There exist a constant $\rho > 2$ and a Delaunay refinement algorithm that constructs a conforming Delaunay triangulation of any input PLC such that all resulting tetrahedra have radius-edge ratio at most ρ.*

In fact, all tetrahedra inside the union of protecting balls have aspect ratios that depend only on the smallest angle in the input PLC. Subsequently, different simplifications and algorithms with less expensive primitives have been proposed [3, 7].

Placing Steiner vertices on the protecting balls and constructing the convex hull of the Steiner vertices on a protecting ball are fairly expensive. Even checking whether a Steiner vertex to be inserted lies inside any protecting ball is a burden. These expensive computations can be bypassed by switching to the *weighted Delaunay triangulation* – a more general variant of Delaunay triangulation.

Let B_x and B_y be two balls with centers x and y and radii $r_x \geq 0$ and $r_y \geq 0$. The *power distance* between B_x and B_y is defined to be

$$\pi(B_x, B_y) = d(x, y)^2 - r_x^2 - r_y^2.$$

This definition allows B_x or B_y to degenerate to a single point. As in the Euclidean case, the bisector between B_x and B_y is also a plane perpendicular to the line through x and y; however, the bisector plane may not pass through the midpoint of xy. Using the power distance, one can define a weighted version of the Voronoi diagram called the *power diagram*. The dual of the power diagram is known as the *weighted Delaunay triangulation*. For each segment, triangle or tetrahedron σ in the triangulation, there is a point z at equal and minimum power distances D from the vertices of σ. This point z is known as the *orthocenter* of σ. The ball centered at z with radius $\sqrt{D}$ is called the *orthoball* of σ, which is at zero power distances from the vertices of σ.

The key idea is to use a weighted Delaunay triangulation after placing the protecting balls. The Delaunay refinement strategy is then modified to insert orthocenters instead of centers of diametric balls. If the protecting balls are not too large, every triangle or tetrahedron σ in the initial weighted Delaunay triangulation involve a pair of nonoverlapping protecting ball, which must be a positive power distance apart. It follows that the orthocenter of σ lies outside all protecting balls. As the refinement progresses, an edge, triangle, or tetrahedron σ may involve Steiner vertices which can be viewed as balls of zero radii. Such a Steiner vertex must be at positive power distances from the other vertices of σ, so the orthocenter of σ also lies outside all protecting balls. In summary, the indefinite insertions of Steiner vertices at decreasing distances from the input vertices and edges as shown in Fig. 2 cannot happen. The following theorem summarizes the result.

Theorem 4 ([4]) *Let P be a PLC. Let ρ be any constant at least 2. There is an algorithm that constructs a conforming weighted Delaunay triangulation of P in which no tetrahedron has an orthoradius-edge ratio greater than ρ. Therefore, tetrahedra with no weighted vertices have circumradius-edge ratio at most ρ.*

There are known methods for eliminating slivers from a conforming weighted Delaunay triangulation of a PLC [1, 4].

Cross-References

- ▸ Voronoi Diagrams and Delaunay Triangulations
- ▸ Meshing Piecewise Smooth Complexes

Recommended Reading

1. Cheng S-W, Dey TK (2003) Quality meshing with weighted Delaunay refinement. SIAM J Comput 33(1):69–93
2. Cheng S-W, Poon S-H (2006) Three-dimensional Delaunay mesh generation. Discret Comput Geom 36(3):419–456; conference version in Proceedings of the ACM-SIAM symposium on discrete algorithms (2003)
3. Cheng S-W, Dey TK, Ramos EA, Ray T (2005) Quality meshing for polyhedra with small angles. Int J Comput Geom Appl 15(4):421–461; conference version in Proceedings of the annual symposium on computational geometry (2004)
4. Cheng S-W, Dey TK, Shewchuk JR (2012) Delaunay mesh generation. CRC, Boca Raton
5. Cohen-Steiner D, de Verdière ÉC, Yvinec M (2004) Conforming Delaunay triangulation in 3D. Comput Geom Theory Appl 28(2–3):217–233
6. Murphy M, Mount DM, Gable CW (2001) A point-placement strategy for conforming Delaunay tetrahedralization. Int J Comput Geom Appl 11(6):669–682
7. Pav SE, Walkington NJ (2004) Robust three-dimensional Delaunay refinement. In: Proceedings of the international meshing roundtable, Williamsburg, pp 145–156
8. Ruppert J (1995) A Delaunay refinement algorithm for quality 2-dimensional mesh generation. J. Algorithms 18(3):548–585
9. Shewchuk JR (1998) Tetrahedral mesh generation by Delaunay refinement. In: Proceedings of the annual symposium on computational geometry, Minneapolis, pp 86–95

Decoding Reed–Solomon Codes

Venkatesan Guruswami
Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA

Keywords

Decoding; Error correction

Years and Authors of Summarized Original Work

1999; Guruswami, Sudan

Problem Definition

In order to ensure the integrity of data in the presence of errors, an *error-correcting code* is used to *encode* data into a redundant form (called a *codeword*). It is natural to view both the original data (or *message*) as well as the associated codeword as strings over a finite alphabet. Therefore, an error-correcting code C is defined by an injective encoding map $E: \Sigma^k \rightarrow \Sigma^n$, where k is called the *message length*, and n the *block length*. The codeword, being a redundant form of the message, will be longer than the message. The *rate* of an error-correcting code is defined as the ratio k/n of the length of the message to the length of the codeword. The rate is a quantity in the interval $(0, 1]$, and is a measure of the redundancy introduced by the code. Let $R(C)$ denote the rate of a code C.

The redundancy built into a codeword enables detection and hopefully also correction of any errors introduced, since only a small fraction of all possible strings will be legitimate codewords. Ideally, the codewords encoding different messages should be "far-off" from each other, so that one can recover the original codeword even when it is distorted by moderate levels of noise. A natural measure of distance between strings is the Hamming distance. The Hamming distance between strings $x, y \in \Sigma^*$ of the same length, denoted $\mathsf{dist}(x, y)$, is defined to be the number of positions i for which $x_i \neq y_i$.

The *minimum distance*, or simply *distance*, of an error-correcting code C, denoted $d(C)$, is defined to be the smallest Hamming distance between the encodings of two distinct messages. The *relative distance* of a code C of block length n, denoted $\delta(C)$, is the ratio between its distance and n. Note that arbitrary corruption of any $\lfloor (d(C) - 1)/2 \rfloor$ of locations of a codeword of C cannot take it closer (in Hamming distance) to any other codeword of C. Thus in principle (i.e., efficiency considerations apart) error patterns of at most $\lfloor (d(C) - 1)/2 \rfloor$ errors can be corrected. This task is called *unique decoding* or *decoding up to half-the-distance*. Of course, it is also possible, and will often be the case, that error patterns with more than $d(C)/2$ errors can also be corrected by decoding the string to the closest codeword in Hamming distance. The latter task is called *Nearest-Codeword decoding* or *Maximum Likelihood Decoding (MLD)*.

One of the fundamental trade-offs in the theory of error-correcting codes, and in fact one could say all of combinatorics, is the one between rate $R(C)$ and distance $d(C)$ of a code. Naturally, as one increases the rate and thus number of codewords in a code, some two codewords must come closer together thereby lowering the distance. More qualitatively, this represents the tension between the redundancy of a code and its error-resilience. To correct more errors requires greater redundancy, and thus lower rate.

A code defined by encoding map $E : \Sigma^k \rightarrow \Sigma^n$ with minimum distance d is said to be an (n, k, d) code. Since there are $|\Sigma|^k$ codewords and only $|\Sigma^{k-1}|$ possible projections onto the first $k = 1$ coordinates, some two codewords must agree on the first $k - 1$ positions, implying that the distance d of the code must obey $d \leq n - k + 1$ (this is called the *Singleton bound*). Quite surprisingly, over large alphabets Σ there are well-known codes called Reed–Solomon codes which meet this bound exactly and have the optimal distance $d = n - k + 1$ for any given rate k/n. (In contrast, for small alphabets, such as $\Sigma = \{0, 1\}$, the optimal trade-off between rate and relative distance for an asymptotic family of codes is unknown and is a major open question in combinatorics.)

This article will describe the best known algorithmic results for error-correction of Reed–Solomon codes. These are of central theoretical and practical interest given the above-mentioned optimal trade-off achieved by Reed–Solomon codes, and their ubiquitous use in our everyday lives ranging from compact disc players to deep-space communication.

Reed–Solomon Codes

Definition 1 A *Reed–Solomon code* (or RS code), $\mathsf{RS}_{\mathbb{F},S}[n,k]$, is parametrized by integers n,k satisfying $1 \le k \le n$, a finite field $\mathbb{F}$ of size at least n, and a tuple $S = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ of n *distinct* elements from $\mathbb{F}$. The code is described as a subset of $\mathbb{F}^n$ as:

$$\mathsf{RS}_{\mathbb{F},S}[n,k] = \{(p(\alpha_1), p(\alpha_2), \ldots, p(\alpha_n)) \mid p(X) \in \mathbb{F}[X] \text{ is a polynomial of degree} \le k-1\}.$$

In other words, the message is viewed as a polynomial, and it is encoded by evaluating the polynomial at n distinct field elements $\alpha_1, \ldots, \alpha_n$. The resulting code is linear of dimension k, and its minimum distance equals $n - k + 1$, which matches the Singleton bound.

The distance property of RS codes follows from the fact that the evaluations of two distinct polynomials of degree less than k can agree on at most $k-1$ field elements. Note that in the absence of errors, given a codeword $\mathbf{y} \in \mathbb{F}^n$, one can recover its corresponding message by polynomial interpolation on any k out of the n codeword positions. In fact, this also gives an *erasure decoding* algorithm when all but the information-theoretically bare minimum necessary k symbols are erased from the codeword (but the receiver *knows* which symbols have been erased and the correct values of the rest of the symbols). The RS decoding problem, therefore, amounts to a noisy polynomial interpolation problem when some of the evaluation values are incorrect.

The holy grail in decoding RS codes would be to find the polynomial $p(X)$ whose RS encoding is closest in Hamming distance to a noisy string $\mathbf{y} \in \mathbb{F}^n$. One could then decode $\mathbf{y}$ to this message $p(X)$ as the maximum likelihood choice. No efficient algorithm for such nearest-codeword decoding is known for RS codes (or for that matter any family of "good" or non-trivial codes), and it is believed that the problem is NP-hard. Guruswami and Vardy [6] proved the problem to NP-hard over exponentially large fields, but this is a weak negative result since normally one considers Reed–Solomon codes over fields of size at most $O(n)$.

Given the intractability of nearest-codeword decoding in its extreme generality, lot of attention has been devoted to the *bounded distance decoding* problem, where one assumes that the string $\mathbf{y} \in \mathbb{F}^n$ to be decoded has at most e errors, and the goal is to find the Reed–Solomon codeword(s) within Hamming distance e from $\mathbf{y}$.

When $e < (n-k)/2$, this corresponds to decoding up to half the distance. This is a classical problem for which a polynomial time algorithm was first given by Peterson [8]. (It is notable that this even before the notion of polynomial time was put forth as the metric of theoretical efficiency.) The focus of this article is on a *list decoding* algorithm for Reed–Solomon codes due to Guruswami and Sudan [5] that decode beyond half the minimum distance. The formal problem and the key results are stated next.

Key Results

In this section, the main result of focus concerning decoding Reed–Solomon codes is stated. Given the target of decoding errors beyond half-the-minimum distance, one needs to deal with inputs where there may be more than one codeword within the radius e specified in the bounded distance decoding problem. This is achieved by a relaxation of decoding called *list decoding* where the decoder outputs a list of all codewords (or the corresponding messages) within Hamming distance e from the received word. If one wishes, one can choose the closest codeword among the list as the "most likely" answer, but there are many applications of Reed–Solomon decoding, for example to decoding concatenated codes and several applications in complexity theory and cryptography, where having the entire list of codewords adds to the power of the decoding primitive. The main result of Guruswami and Sudan [5], building upon the work of Sudan [9], is the following:

Theorem 1 ([5]) *Let $C = \mathsf{RS}_{\mathbb{F},S}[n,k]$ be a Reed–Solomon code over a field $\mathbb{F}$ of size $q \ge n$ with $S = (\alpha_1, \alpha_2, \ldots, \alpha_n)$. There is a deterministic algorithm running in time polynomial in q that on input $y \in \mathbb{F}_q^n$ outputs*

a list of all polynomials $p(X) \in \mathbb{F}[X]$ *of degree less than* k *for which* $p(\alpha_i) \neq y_i$ *for less than* $n - \sqrt{(k-1)n}$ *positions* $i \in \{1, 2, \ldots, n\}$. *Further, at most* $O(n^2)$ *polynomials will be output by the algorithm in the worst-case.*

Alternatively, one can correct a RS code of block length n and rate $R = k/n$ up to $n - \sqrt{(k-1)}$ errors, or equivalently a fraction $1 - \sqrt{R}$ of errors.

The Reed–Solomon decoding algorithm is based on the solution to the following more general polynomial reconstruction problem which seems like a natural algebraic question in itself. (The problem is more general than RS decoding since the α_i's need not be distinct.)

Problem 1 (Polynomial Reconstruction)
Input: Integers $k, t \leq n$ and n distinct pairs $\{(\alpha_i, y_i)\}_{i=1}^{n}$ where $\alpha_i, y_i \in \mathbb{F}$.
Output: A list of all polynomials $p(X) \in \mathbb{F}[X]$ of degree less than k which satisfy $p(\alpha_i) = y_i$ for at least t values of $i \in [n]$.

Theorem 2 *The polynomial reconstruction problem can be solved in time polynomial in* $n, |\mathbb{F}|$, *provided* $t > \sqrt{(k-1)n}$.

The reader is referred to the original papers [5, 9], or a recent survey [1], for details on the above algorithm. A quick, high level peek into the main ideas is given below. The first step in the algorithm consists of an interpolation step where a nonzero bivariate polynomial $Q(X,Y)$ is "fit" through the n pairs (α_i, y_i), so that $Q(\alpha_i, y_i) = 0$ for every i. The key is to do this with relatively low degree; in particular one can find such a $Q(X,Y)$ with so-called $(1, k-1)$-weighted degree at most $D \approx \sqrt{2(k-1)n}$. This degree budget on Q implies that for any polynomial $p(X)$ of degree less than k, $Q(X, p(X))$ will have degree at most D. Now whenever $p(\alpha_i) = y_i$, $Q(\alpha_i, p(\alpha)i)) = Q(\alpha_i, y_i) = 0$. Therefore, if a polynomial $p(X)$ satisfies $p(\alpha_i) = y_i$ for at least t values of i, then $Q(X, p(X))$ has at least t roots. On the other hand the polynomial $Q(X, p(X))$ has degree at most D. Therefore, if $t > D$, one must have $Q(X, p(X)) = 0$, or in other words $Y - p(X)$ is a factor of $Q(X,Y)$. The second step of the algorithm factorized the polynomial $Q(X,Y)$, and all polynomials $p(X)$ that must be output will be found as factors $Y - p(X)$ of $Q(X,Y)$.

Note that since $D \approx \sqrt{2(k-1)n}$ this gives an algorithm for polynomial reconstruction provided the agreement parameter t satisfies $t > \sqrt{2(k-1)n}$ [9]. To get an algorithm for $t > \sqrt{(k-1)n}$, and thus decode beyond half the minimum distance $(n-k)/2$ for all parameter choices for k, n, Guruswami and Sudan [5] use the crucial idea of allowing "multiple roots" in the interpolation step. Specifically, the polynomial Q is required to have $r \geq 1$ roots at each pair (α_i, y_i) for some integer multiplicity parameter r (the notion needs to be formalized properly, see [5] for details). This necessitates an increase in the $(1, k-1)$-weighted degree of a factor of about $r/\sqrt{2}$, but the gain is that one gets a factor r more roots for the polynomial $Q(X, p(X))$. These facts together lead to an algorithm that works as long as $t > \sqrt{(k-1)n}$.

There is an additional significant benefit offered by the multiplicity based decoder. The multiplicities of the interpolation points need not all be equal and they can picked in proportion to the reliability of different received symbols. This gives a powerful way to exploit "soft" information in the decoding stage, leading to impressive coding gains in practice. The reader is referred to the paper by Koetter and Vardy [7] for further details on using multiplicities to encode symbol level reliability information from the channel.

Applications

Reed–Solomon codes have been extensively studied and are widely used in practice. The above decoding algorithm corrects more errors beyond the traditional half the distance limit and therefore directly advances the state of the art on this important algorithmic task. The RS list decoding algorithm has also been the backbone for many further developments in algorithmic coding theory. In particular, using this algorithm in concatenation schemes leads to good binary list-decodable codes. A variant of

RS codes called folded RS codes have been used to achieve the optimal trade-off between error-correction radius and rate [3] (see the companion encyclopedia entry by Rudra on folded RS codes).

The RS list decoding algorithm has also found many surprising applications beyond coding theory. In particular, it plays a key role in several results in cryptography and complexity theory (such as constructions of randomness extractors and pseudorandom generators, hardness amplification, constructions to hardcore predicates, traitor tracing, reductions connecting worst-case hardness to average-case hardness, etc.); more information can be found, for instance, in [10] or Chap. 12 in [2].

Open Problems

The most natural open question is whether one can improve the algorithm further and correct more than a fraction $1 - \sqrt{R}$ of errors for RS codes of rate R. It is important to note that there is a combinatorial limitation to the number of errors one can list decode from. One can only list decode in polynomial time from a fraction ρ of errors if for every received word $\mathbf{y}$ the number of RS codewords within distance $e = \rho n$ of $\mathbf{y}$ is bounded by a polynomial function of the block length n. The largest ρ for which this holds as a function of the rate R is called the list decoding radius $\rho_{\mathsf{LD}} = \rho_{\mathsf{LD}}(R)$ of RS codes. The RS list decoding algorithm discussed here implies that $\rho_{\mathsf{LD}}(R) \geq 1 - \sqrt{R}$, and it is trivial to see than $\rho_{\mathsf{LD}}(R) \leq 1 - R$. Are there RS codes (perhaps based on specially structured evaluation points) for which $\rho_{\mathsf{LD}}(R) > 1 - \sqrt{R}$? Are there RS codes for which the $1 - \sqrt{R}$ radius (the so-called "Johnson bound") is actually tight for list decoding? For the more general polynomial reconstruction problem the $\sqrt{(k-1)n}$ agreement cannot be improved upon [4], but this is not known for RS list decoding.

Improving the NP-hardness result of [6] to hold for RS codes over polynomial sized fields and for smaller decoding radii remains an important challenge.

Cross-References

- Learning Heavy Fourier Coefficients of Boolean Functions
- List Decoding near Capacity: Folded RS Codes
- LP Decoding

Recommended Reading

1. Guruswami V (2007) Algorithmic results in list decoding. In: Foundations and trends in theoretical computer science, vol 2, issue 2. NOW Publishers, Hanover
2. Guruswami V (2004) List decoding of error-correcting codes. Lecture notes in computer science, vol 3282. Springer, Berlin
3. Guruswami V, Rudra A (2008) Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. IEEE Trans Inform Theory 54(1):135–150
4. Guruswami V, Rudra A (2006) Limits to list decoding Reed–Solomon codes. IEEE Trans Inf Theory 52(8):3642–3649
5. Guruswami V, Sudan M (1999) Improved decoding of Reed–Solomon and algebraic-geometric codes. IEEE Trans Inf Theory 45(6):1757–1767
6. Guruswami V, Vardy A (2005) Maximum likelihood decoding of Reed–Solomon codes is NP-hard. IEEE Trans Inf Theory 51(7):2249–2256
7. Koetter R, Vardy A (2003) Algebraic soft-decision decoding of Reed–Solomon codes. IEEE Trans Inf Theory 49(11):2809–2825
8. Peterson WW (1960) Encoding and error-correction procedures for Bose-Chaudhuri codes. IEEE Trans Inf Theory 6:459–470
9. Sudan M (1997) Decoding of Reed–Solomon codes beyond the error-correction bound. J Complex 13(1):180–193
10. Sudan M (2000) List decoding: algorithms and applications. SIGACT News 31(1):16–27

Decremental All-Pairs Shortest Paths

Camil Demetrescu and Giuseppe F. Italiano
Department of Computer and Systems Science, University of Rome, Rome, Italy
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords

Deletions-only dynamic all-pairs shortest paths

Years and Authors of Summarized Original Work

2004; Demetrescu, Italiano

Problem Definition

A dynamic graph algorithm maintains a given property P on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property P quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

This entry addressed the *decremental* version of the all-pairs shortest paths problem (APSP), which consists of maintaining a directed graph with real-valued edge weights under an intermixed sequence of the following operations:

`delete`(u, v): delete edge (u, v) from the graph.
`distance`(x, y): return the distance from vertex x to vertex y.
`path`(x, y): report a shortest path from vertex x to vertex y, if any.

A natural variant of this problem supports a generalized delete operation that removes a vertex and all edges incident to it. The algorithms addressed in this entry can deal with this generalized operation within the same bounds.

History of the Problem

A simple-minded solution to this problem would be to rebuild shortest paths from scratch after each deletion using the best static APSP algorithm so that distance and path queries can be reported in optimal time. The fastest known static APSP algorithm for arbitrary real weights has a running time of $O(mn + n^2 \log\log n)$, where m is the number of edges and n is the number of vertices in the graph [13]. This is $\Omega(n^3)$ in the worst case. Fredman [6] and later Takaoka [19] showed how to break this cubic barrier: the best asymptotic bound is by Takaoka, who showed how to solve APSP in $O(n^3\sqrt{\log\log n/\log n})$ time.

Another simple-minded solution would be to answer queries by running a point-to-point shortest paths computation, without the need to update shortest paths at each deletion. This can be done with Dijkstra's algorithm [3] in $O(m + n\log n)$ time using the Fibonacci heaps of Fredman and Tarjan [5]. With this approach, queries are answered in $O(m + n\log n)$ worst-case time and updates require optimal time.

The dynamic maintenance of shortest paths has a long history, and the first papers date back to 1967 [11, 12, 17]. In 1985 Even and Gazit [4] presented algorithms for maintaining shortest paths on directed graphs with arbitrary real weights. The worst-case bounds of their algorithm for edge deletions were comparable to recomputing APSP from scratch. Also Ramalingam and Reps [15, 16] and Frigioni et al. [7, 8] considered dynamic shortest path algorithms with real weights, but in a different model. Namely, the running time of their algorithm is analyzed in terms of the output change rather than the input size (*output bounded complexity*). Again, in the worst case the running times of output-bounded dynamic algorithms are comparable to recomputing APSP from scratch.

The first decremental algorithm that was provably faster than recomputing from scratch was devised by King for the special case of graphs with integer edge weights less than C: her algorithm can update shortest paths in a graph subject to a sequence of $\Omega(n^2)$ deletions in $O(C \cdot n^2)$ amortized time per deletion [9]. Later, Demetrescu and Italiano showed how to deal with graphs with real non-negative edge weights in $O(n^2 \log n)$ amortized time per deletion [2] in a sequence of $\Omega(m/n)$ operations. Both algorithms work in the more general case where edges are not deleted from the graph, but their weight is increased at each update. Moreover, since they update shortest paths explicitly after each deletion, queries are answered in

optimal time at any time during a sequence of operations.

Key Results

The decremental APSP algorithm by Demetrescu and Italiano hinges upon the notion of locally shortest paths [2].

Definition 1 A path is *locally shortest* in a graph if all of its proper subpaths are shortest paths.

Notice that by the optimal-substructure property, a shortest path is locally shortest. The main idea of the algorithm is to keep information about locally shortest paths in a graph subject to edge deletions. The following theorem derived from [2] bounds the number of changes in the set of locally shortest paths due to an edge deletion:

Theorem 1 *If shortest paths are unique in the graph, then the number of paths that start or stop being shortest at each deletion is* $O(n^2)$ *amortized over* $\Omega(m/n)$ *update operations.*

The result of Theorem 1 is purely combinatorial and assumes that shortest paths are unique in the graph. The latter can be easily achieved using any consistent tie-breaking strategy (see, e.g., [2]). It is possible to design a deletions-only algorithm that pays only $O(\log n)$ time per change in the set of locally shortest paths, using a simple modification of Dijkstra's algorithm [3]. Since by Theorem 1 the amortized number of changes is bounded by $O(n^2)$, this yields the following result:

Theorem 2 *Consider a graph with n vertices and an initial number of m edges subject to a sequence of* $\Omega(m/n)$ *edge deletions. If shortest paths are unique and edge weights are nonnegative, it is possible to support each delete operation in* $O(n^2 \log n)$ *amortized time, each distance query in O(1) worst-case time, and each path query in* $O(\ell)$ *worst-case time, where* ℓ *is the number of vertices in the reported shortest path. The space used is O(mn).*

Applications

Application scenarios of dynamic shortest paths include network optimization [1], document formatting [10], routing in communication systems, robotics, incremental compilation, traffic information systems [18], and dataflow analysis. A comprehensive review of real-world applications of dynamic shortest path problems appears in [14].

URL to Code

An efficient C language implementation of the decremental algorithm addressed in section "Key Results" is available at the URL: http://www.dis.uniroma1.it/~demetres/experim/dsp.

Cross-References

- All Pairs Shortest Paths in Sparse Graphs
- All Pairs Shortest Paths via Matrix Multiplication
- Fully Dynamic All Pairs Shortest Paths

Recommended Reading

1. Ahuja R, Magnanti T, Orlin J (1993) Network flows: theory, algorithms and applications. Prentice Hall, Englewood Cliffs
2. Demetrescu C, Italiano G (2004) A new approach to dynamic all pairs shortest paths. J Assoc Comput Mach 51:968–992
3. Dijkstra E (1959) A note on two problems in connexion with graphs. Numer Math 1:269–271
4. Even S, Gazit H (1985) Updating distances in dynamic graphs. Methods Oper Res 49:371–387
5. Fredman M, Tarjan R (1987) Fibonacci heaps and their use in improved network optimization algorithms. J ACM 34:596–615
6. Fredman ML (1976) New bounds on the complexity of the shortest path problems. SIAM J Comput 5(1):87–89
7. Frigioni D, Marchetti-Spaccamela A, Nanni U (1998) Semi-dynamic algorithms for maintaining single source shortest paths trees. Algorithmica 22: 250–274
8. Frigioni D, Marchetti-Spaccamela A, Nanni U (2000) Fully dynamic algorithms for maintaining shortest paths trees. J Algorithm 34:351–381

9. King V (1999) Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: Proceedings of 40th IEEE symposium on foundations of computer science (FOCS'99). IEEE Computer Society, New York, pp 81–99
10. Knuth D, Plass M (1981) Breaking paragraphs into lines. Softw-Pract Exp 11:1119–1184
11. Loubal P (1967) A network evaluation procedure. Highw Res Rec 205:96–109
12. Murchland J (1967) The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph. Technical report, LBS-TNT-26, London Business School. Transport Network Theory Unit, London
13. Pettie S (2003) A new approach to all-pairs shortest paths on realweighted graphs. Theor Comput Sci 312:47–74, Special issue of selected papers from ICALP (2002)
14. Ramalingam G (1996) Bounded incremental computation. Lect Note Comput Sci 1089
15. Ramalingam G, Reps T (1996) An incremental algorithm for a generalization of the shortest path problem. J Algorithm 21:267–305
16. Ramalingam G, Reps T (1996) On the computational complexity of dynamic graph problems. Theor Comput Sci 158:233–277
17. Rodionov V (1968) The parametric problem of shortest distances. USSR Comput Math Math Phys 8:336–343
18. Schulz F, Wagner D, Weihe K (1999) Dijkstra's algorithm on-line: an empirical case study frompublic railroad transport. In: Proceedings of 3rd workshop on algorithm engineering (WAE'99), London. Notes in computer science, vol 1668, pp 110–123
19. Takaoka T (1992) A new upper bound on the complexity of the all pairs shortest path problem. Inf Proc Lett 43:195–199

Decremental Approximate-APSP in Directed Graphs

Aaron Bernstein
Department of Computer Science, Columbia University, New York, NY, USA

Keywords

Directed graphs; Dynamic graph algorithms; Shortest paths

Years and Authors of Summarized Original Work

2013; Bernstein

Problem Definition

A dynamic graph algorithm maintains information about a graph that is changing over time. Given a property $\mathcal{P}$ of the graph (e.g., minimum spanning tree), the algorithm must support an online sequence of query and update operations, where an update operation changes the underlying graph, while a query operation asks for the state of $\mathcal{P}$ in the current graph. In the typical model studied, each update only affects a single edge. In a *fully dynamic* setting, an update can insert or delete an edge or change the weight of an existing edge; in a *decremental* setting an update can only delete an edge or increase a weight; in an *incremental* setting an update can insert an edge or decrease a weight.

This entry addresses the decremental $(1+\epsilon)$-approximate all-pairs shortest path problem (APSP) in weighted directed graphs. The goal is to maintain a directed graph G with real-valued nonnegative edge weights under an online intermixed sequence of the following operations:

- **delete**(u, v) (update): remove edge (u, v) from G.
- **increase-weight**(u, v, w) (update): increase the weight of edge (u, v) to w.
- **distance**(u, v) (query): return a $(1+\epsilon)$-approximation to the shortest $u - v$ distance in G.
- **path**(u, v) (query): return a $(1+\epsilon)$-approximate shortest path from u to v.

A History of Decremental APSP

The naive approach to the decremental APSP problem is to recompute shortest paths from scratch after every update, allowing queries to be answered in optimal time. Letting n be the number of vertices and m the number of edges, computing APSP requires $O(mn + n^2 \log\log(n))$

time in sparse graphs [11] or slightly less than n^3 in dense graphs (see [13, 14]). Another simple-minded approach would be to not perform any computation during the updates and to simply compute the shortest $u - v$ path from scratch when a query arrived. This would lead to a constant update time and a query time of $O(m + n\log(n))$ using Dijkstra's algorithm with Fibonacci heaps [6].

One can improve significantly upon both the above approaches by reusing information between updates. Decremental shortest path algorithms have a long history, with the current state of the art for the general case of directed graphs with real-valued weights being an algorithm of Demetrescu and Italiano which achieves constant query time and update time $O(n^2\log(n))$ [5]. Later papers improved upon $O(n^2)$ update time in restricted types of graph. In *unweighted* directed graphs, Baswana et al. achieve an amortized update of $O(n^3\log^2(n)/m)$ for exact distances and $\tilde{O}(\epsilon^{-1}n^2/\sqrt{m})$ for $(1+\epsilon)$ approximate distances [1]. (We say that $f(n) = \tilde{O}(g(n))$ if $f(n) = O(g(n)\text{polylog}(n)))$. Keeping the $(1 + \epsilon)$ approximation, Roditty and Zwick further reduced the amortized update time to $\tilde{O}(n/\epsilon)$ [12].

An amortized update time of $\tilde{O}(n)$ forms a natural barrier for decremental APSP because if edges are deleted from the graph one at a time, an $\tilde{O}(n)$ update time allows us to maintain APSP over the entire sequence of deletions in a *total* of $\tilde{O}(mn)$ time; excepting fast matrix multiplication in dense graphs, this $\tilde{O}(mn)$ matches the best known bound for the much simpler problem of computing APSP a single time in the *static* setting. Roddity and Zwick achieve this desired *total* update time of $\tilde{O}(mn)$ only for undirected, unweighted graphs; this entry focuses on a result of Bernstein that achieves the same $\tilde{O}(mn)$ for directed graphs with weights polynomial in n [3].

There have recently been several results on breaking through the $\tilde{O}(n)$ amortized update time barrier in undirected graphs by allowing a larger than $(1 + \epsilon)$ approximation (see [4, 7, 9]), as well in directed graphs for *single-source* shortest paths [8].

Key Results

Bernstein's result shows that in a directed graph with weights polynomial in n, we can maintain $(1 + \epsilon)$-approximate decremental APSP with constant query time and a *total* update time of $\tilde{O}(mn)$ over the entire sequence of deletions and weight increases (see Theorem 2 below). At a high level, Bernstein's result uses the framework from his earlier paper on fully dynamic APSP in undirected graphs [2], but all the details and techniques change significantly in the shift to directed graphs.

From Weighted Distances to Hop Distances

The majority of dynamic APSP algorithms use as a building block an algorithm of King for maintaining a *single-source* shortest path tree under deletions [10]. King's algorithm maintains a shortest path tree up to distance d (assuming integral weights) in the total update time $O(md)$ over all deletions (amortized $O(d)$ per update); hence, $O(mn)$ is the total update time in unweighted graphs where $d \leq n$. This makes it an extremely efficient building block for small distances, but with two main drawbacks: it is inefficient at handling vertices that are far apart and it completely fails in graphs with large weights where d can be very big. Bernstein's algorithm overcomes the second of these problems by showing that if we allow a $(1+\epsilon)$ approximation, then with a simple scaling approach, we can shift the dependency from the weighted distance d between two vertices to the unweighted *hop distance* between them.

Definition 1 The hop distance between two vertices x and y is the number of edges on the shortest $x-y$ path. The $(1 + \epsilon)$-approximate $x-y$ hop distance, denoted $h(x, y)$, is the minimum number of edges among any $(1 + \epsilon)$-approximate $x - y$ path.

Theorem 1 *Given a directed graph G with non-negative real weights and a source s and letting R be the ratio of the heaviest to the lightest nonzero edge weight in the graph, we can for any*

hop distance h decrementally maintain $(1+\epsilon)$*-shortest paths from s to all vertices v for which* $h(s,v) \le h$*. The total update time over the whole sequence of deletions and weight increases is* $O(nh\log(R)/\epsilon)$*, which is* $O(nh)$ *if weights are polynomial in n.*

We refer to the above decremental SSSP algorithm as h-SSSP. In short, Theorem 1 tells us that with a $(1+\epsilon)$ approximation, we can decrementally maintain a shortest path tree in time proportional not to the maximum distance of the tree ($O(nd)$) but to the maximum *hop distance* ($O(nh)$) of the tree. This is a big improvement in weighted graphs where $h \ll d$, but still inadequate as h can be $\Omega(n)$. Bernstein's key idea is that regardless of whether the original graph is weighted or not, we can add weighted edges that reduce hop distances in the graph and hence allow h-SSSP to run extremely efficiently.

Shortcut Edges

The algorithm of Bernstein works by adding many different (weighted) shortcut edges (x,y) to the original graph G, which are defined as edges that do not exist in G itself and have weight $w(x,y)$ satisfying $\delta(x,y) \le w(x,y) \le (1+\epsilon)\delta(x,y)$, where $\delta(x,y)$ is the shortest $x-y$ distance. Note that as the graph changes, $\delta(x,y)$ will increase, and so the algorithm will have to increase $w(x,y)$ for the shortcut edge to remain valid; a shortcut edge (x,y) is not simply computed once, but must be maintained over the whole sequence of edge deletions and weight increases.

It is clear that because the weight of a shortcut edge (x,y) is tethered to $\delta(x,y)$, the shortcut edges do not change shortest distances in the graph. But they do drastically reduce hop distances. In an unweighted graph with $\delta(x,y) = 1{,}000$, a single $x-y$ edge of weight 1,000 (or slightly larger) decreases $h(x,y)$ from 1,000 to 1. Moreover, any path that goes through x and y can also use the (x,y) shortcut edge to reduce its hop distance by 999.

Bernstein's algorithm runs in phases, each of which adds more shortcut edges to successively decrease all hop distances in the graph by a factor of 2. It starts by defining a small set of pairs S_1 for which it maintains approximate shortest paths over the entire sequence of updates; this first step can easily be done in the desired $\tilde{O}(mn\log(R))$ total update time as instead of maintaining *all-pairs* shortest paths, the algorithm only has to maintain $|S_1|$ pairs. Now that the algorithm maintains $\delta(x,y)$ for every pair (x,y) in S_1, it can add shortcut edges (x,y) to the graph. These shortcut edges decrease hop distances in the graph, thus increasing the efficiency of the h-SSSP building block and allowing it to maintain approximate shortest distances within a slightly larger set of pairs S_2 in the same $\tilde{O}(mn\log(R))$ total update time. Since knowing the shortest distance between two vertices allows us to maintain a corresponding shortcut edge, maintaining the larger set of distances S_2 directly leads to a larger set of shortcut edges, which further reduce hop distances in the graph, allowing h-SSSP to efficiently maintain shortest distances for an even larger set of pairs S_3; this in turn leads to more shortcut edges, thus further reducing hop distances and allowing h-SSSP to maintain a larger distance set S_4, and so on. After $\log(n)$ such layers, there are enough shortcut edges to ensure that all hop distances are constant, so by Theorem 1 h-SPPP can decrementally maintain a shortest path tree in the graph in a total update time of only $\tilde{O}(n\log(R)/\epsilon)$; doing this from every vertex yields the desired bound of $\tilde{O}(mn\log(R)/\epsilon)$ for decremental APSP.

Theorem 2 *Let G be a graph with nonnegative real-valued edge weights, n vertices, and m initial edges subject to an arbitrary sequence of* Σ *edge deletions and weight increases. Let R be the ratio of the heaviest weight that ever appears in G to the lightest nonzero weight. It is possible to support the whole sequence of updates in a* total *time* $\tilde{O}(mn\log(R)/\epsilon) + O(\Sigma)$ *while answering queries with a single* $O(1)$ *time lookup.*

Cross-References

- Decremental All-Pairs Shortest Paths
- Dynamic Approximate All-Pairs Shortest Paths: Breaking the $O(mn)$ Barrier and Derandomization
- Dynamic Approximate-APSP
- Trade-Offs for Dynamic Graph Problems

Recommended Reading

1. Baswana S, Hariharan R, Sen S (2007) Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. J Algorithms 62(2):74–92
2. Bernstein A (2009) Fully dynamic approximate all-pairs shortest paths with constant query and close to linear update time. In: Proceedings of the 50th FOCS, Atlanta, pp 50–60
3. Bernstein A (2013) Maintaining shortest paths under deletions in weighted directed graphs: [extended abstract]. In: STOC, Palo Alto, pp 725–734
4. Bernstein A, Roditty L (2011) Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In: Proceedings of the 22nd SODA, San Francisco, pp 1355–1365
5. Demetrescu C, Italiano GF (2004) A new approach to dynamic all pairs shortest paths. J ACM 51(6): 968–992. doi:http://doi.acm.org/10.1145/1039488.1039492
6. Fredman ML, Tarjan RE (1987) Fibonacci heaps and their uses in improved network optimization algorithms. J ACM 34(3):596–615
7. Henzinger M, Krinninger S, Nanongkai D (2013) Dynamic approximate all-pairs shortest paths: breaking the $o(mn)$ barrier and derandomization. In: FOCS 2013, Berkeley, pp 538–547
8. Henzinger M, Krinninger S, Nanongkai D (2014) Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In: STOC, New York, pp 674–683
9. Henzinger M, Krinninger S, Nanongkai D (2014) A subquadratic-time algorithm for decremental single-source shortest paths. In: SODA 2014, Portland
10. King V (1999) Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: FOCS, New York, pp 81–91
11. Pettie S (2004) A new approach to all-pairs shortest paths on real-weighted graphs. Theor Comput Sci 312(1):47–74. doi:10.1016/S0304-3975(03)00402-X, http://dx.doi.org/10.1016/S0304-3975(03)00402-X
12. Roditty L, Zwick U (2012) Dynamic approximate all-pairs shortest paths in undirected graphs. SIAM J Comput 41(3):670–683
13. Takaoka T (1992) A new upper bound on the complexity of the all pairs shortest path problem. Inf Process Lett 43(4):195–199. doi:10.1016/0020-0190(92)90200-F, http://dx.doi.org/10.1016/0020-0190(92)90200-F
14. Williams R (2014) Faster all-pairs shortest paths via circuit complexity. In: STOC, New York, pp 664–673

Degree-Bounded Planar Spanner with Low Weight

Wen-Zhan Song[1], Xiang-Yang Li[2], and Weizhao Wang[3]
[1]School of Engineering and Computer Science, Washington State University, Vancouver, WA, USA
[2]Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA
[3]Google Inc., Irvine, CA, USA

Keywords

Unified energy-efficient unicast and broadcast topology control

Years and Authors of Summarized Original Work

2005; Song, Li, Wang

Problem Definition

An important requirement of wireless ad hoc networks is that they should be self-organizing, and transmission ranges and data paths may need to be dynamically restructured with changing topology. Energy conservation and network performance are probably the most critical issues in wireless ad hoc networks, because wireless devices are usually powered by batteries only and have limited computing capability and memory. Hence, in such a dynamic and resource-limited environment, each wireless node needs to locally select communication neighbors and adjust its transmission power accordingly, such that all nodes together self-form a topology that

is energy efficient for both unicast and broadcast communications.

To support energy-efficient unicast, the topology is preferred to have the following features in the literature:

1. POWER SPANNER: [1, 9, 13, 16, 17] Formally speaking, a subgraph H is called a *power spanner* of a graph G if there is a positive real constant ρ such that for any two nodes, the power consumption of the shortest path in H is at most ρ times of the power consumption of the shortest path in G. Here ρ is called the *power stretch factor* or *spanning ratio*.
2. DEGREE BOUNDED: [1, 9, 11, 13, 16, 17] It is also desirable that the logical node degree in the constructed topology is bounded from above by a small constant. Bounded logical degree structures find applications in Bluetooth wireless networks since a *master* node can have only seven active slaves simultaneously. A structure with small logical node degree will save the cost of updating the routing table when nodes are mobile. A structure with a small degree and using shorter links could improve the overall network throughout [6].
3. PLANAR: [1, 4, 13, 14, 16] A network topology is also preferred to be planar (no two edges crossing each other in the graph) to enable some localized routing algorithms to work correctly and efficiently, such as *Greedy Face Routing* (GFG) [2], *Greedy Perimeter Stateless Routing* (GPSR) [5], *Adaptive Face Routing* (AFR) [7], and *Greedy Other Adaptive Face Routing* (GOAFR) [8]. Notice that with planar network topology as the underlying routing structure, these localized routing protocols guarantee the message delivery without using a routing table: each intermediate node can decide which logical neighboring node to forward the packet to using only local information and the position of the source and the destination.

To support energy-efficient broadcast [15], the locally constructed topology is preferred to be *low-weighted* [10, 12]: the total link length of the final topology is within a constant factor of that of *EMST*. Recently, several localized algorithms [10, 12] have been proposed to construct low-weighted structures, which indeed approximate the energy efficiency of EMST as the network density increases. However, none of them is power efficient for unicast routing.

Before this work, all known topology control algorithms could not support power efficient unicast and broadcast in the same structure. It is indeed challenging to design a unified topology, especially due to the trade off between spanner and low weight property. The main contribution of this algorithm is to address this issue.

Key Results

This algorithm is the *first* localized topology control algorithm for all nodes to maintain a *unified* energy-efficient topology for unicast and broadcast in wireless ad hoc/sensor networks. In one single structure, the following network properties are guaranteed:

1. **Power efficient unicast**: given any two nodes, there is a path connecting them in the structure with total power cost no more than $2\rho + 1$ times the power cost of any path connecting them in the original network. Here $\rho > 1$ is some constant that will be specified later in this algorithm. It assumes that each node u can adjust its power sufficiently to cover its next-hop v on any selected path for unicast.
2. **Power efficient broadcast**: the power consumption for broadcast is within a constant factor of the optimum among all *locally* constructed structures. As proved in [10], to prove this, it equals to prove that the structure is *low-weighted*. Here we called a structure low-weigthed, if its total edge length is within a constant factor of the total length of the Euclidean Minimum Spanning Tree (EMST). For broadcast or generally multicast, it assumes that each node u can adjust its power sufficiently to cover its farthest down-stream node on any selected structure (typically a tree) for multicast.
3. **Bounded logical node degree**: each node has to communicate with at most $k - 1$ logical

Algorithm 1 $S\Theta GG$: Power-Efficient Unicast Topology

1: First, each node self-constructs the Gabriel graph GG locally. The algorithm to construct GG locally is well-known, and a possible implementation may refer to [13]. Initially, all nodes mark themselves WHITE, i.e., *unprocessed*.
2: Once a WHITE node u has the smallest ID among all its WHITE neighbors in $N(u)$, it uses the following strategy to select neighbors:
   1. Node u first sorts all its BLACK neighbors (if available) in $N(u)$ in the distance-increasing order, then sorts all its WHITE neighbors (if available) in $N(u)$ similarly. The sorted results are then restored to $N(u)$, by first writing the sorted list of BLACK neighbors then appending the sorted list of WHITE neighbors.
   2. Node u scans the sorted list $N(u)$ from left to right. In each step, it keeps the current pointed neighbor w in the list, while deletes every *conflicted* node v in the remainder of the list. Here a node v is conflicted with w means that node v is in the θ-dominating region of node w. Here $\theta = 2\pi/k$ ($k \geq 9$) is an adjustable parameter.

   Node u then marks itself BLACK, i.e. *processed*, and notifies each deleted neighboring node v in $N(u)$ by a broadcasting message UPDATEN.
3: Once a node v receives the message UPDATEN from a neighbor u in $N(v)$, it checks whether itself is in the nodes set for deleting: if so, it deletes the sending node u from list $N(v)$, otherwise, marks u as BLACK in $N(v)$.
4: When all nodes are processed, all selected links $\{uv | v \in N(u), \forall v \in GG\}$ form the final network topology, denoted by $S\Theta GG$. Each node can shrink its transmission range as long as it sufficiently reaches its farthest neighbor in the final topology.

neighbors, where $k \geq 9$ is an adjustable parameter.

4. **Bounded average physical node degree**: the expected average physical node degree is at most a small constant. Here the physical degree of a node u in a structure H is defined as the number of nodes inside the disk centered at u with radius $\max_{uv \in H} \|uv\|$.
5. **Planar**: there are no edges crossing each other. This enables several localized routing algorithms, such as [2, 5, 7, 8], to be performed on top of this structure and guarantee the packet delivery without using the routing table.
6. **Neighbors Θ-separated**: the directions between any two logical neighbors of any node are separated by at least an angle θ, which reduces the communication interferences.

It is the *first* known *localized* topology control strategy for all nodes together to maintain such a *single* structure with these desired properties. Previously, only a centralized algorithm was reported in [1]. The first step is Algorithm 1 that can construct a power-efficient topology for unicast, then it extends to the final algorithm (Algorithm 2) that can support power-efficient broadcast at the same time.

Definition 1 (θ-Dominating Region) For each neighbor node v of a node u, the *θ-dominating region* of v is the 2θ-cone emanated from u, with the edge uv as its axis.

Let $N_{\text{UDG}}(u)$ be the set of neighbors of node u in UDG, and let $N(u)$ be the set of neighbors of node u in the final topology, which is initialized as the set of neighbor nodes in GG.

Algorithm 1 constructs a degree-$(k-1)$ planar power spanner.

Lemma 1 *Graph $S\Theta GG$ is connected if the underlying graph GG is connected. Furthermore, given any two nodes u and v, there exists a path $\{u, t_1, \ldots, t_r, v\}$ connecting them such that all edges have length less than $\sqrt{2}\|uv\|$.*

Theorem 1 *The structure $S\Theta GG$ has node degree at most $k-1$ and is planar power spanner with neighbors Θ-separated. Its power stretch factor is at most $\rho = \sqrt{2}^{\beta}/(1-(2\sqrt{2}\sin\frac{\pi}{k})^{\beta})$, where $k \geq 9$ is an adjustable parameter.*

Obviously, the construction is consistent for two endpoints of each edge: if an edge uv is kept by node u, then it is also kept by node v. It is worth mentioning that, the number 3 in criterion $\|xy\| > \max(\|uv\|, 3\|ux\|, 3\|vy\|)$ is carefully selected.

Theorem 2 *The structure $LS\Theta GG$ is a degree-bounded planar spanner. It has a constant power spanning ratio $2\rho + 1$, where ρ is the power spanning ratio of $S\Theta GG$. The node degree is bounded by $k-1$ where $k \geq 9$ is a customizable parameter in $S\Theta GG$.*

Algorithm 2 Construct $LS\Theta GG$: Planar Spanner with Bounded Degree and Low Weight

1: All nodes together construct the graph $S\Theta GG$ in a localized manner, as described in Algorithm 1. Then, each node marks its incident edges in $S\Theta GG$ *unprocessed*.
2: Each node u locally broadcasts its incident edges in $S\Theta GG$ to its one-hop neighbors and listens to its neighbors. Then, each node x can learn the existence of the set of 2-hop links $E_2(x)$, which is defined as follows: $E_2(x) = \{uv \in S\Theta GG \mid u \text{ or } v \in N_{\text{UDG}}(x)\}$. In other words, $E_2(x)$ represents the set of edges in $S\Theta GG$ with at least one endpoint in the transmission range of node x.
3: Once a node x learns that its *unprocessed* incident edge xy has the smallest ID among all *unprocessed* links in $E_2(x)$, it will delete edge xy if there exists an edge $uv \in E_2(x)$ (here both u and v are different from x and y), such that $\|xy\| > \max(\|uv\|, 3\|ux\|, 3\|vy\|)$; otherwise it simply marks edge xy *processed*. Here assume that $uvyx$ is the convex hull of u, v, x and y. Then the link status is broadcasted to all neighbors through a message UPDATESTATUS(XY).
4: Once a node u receives a message UPDATESTATUS(XY), it records the status of link xy at $E_2(u)$.
5: Each node repeats the above two steps until all edges have been *processed*. Let $LS\Theta GG$ be the final structure formed by all remaining edges in $S\Theta GG$.

Theorem 3 *The structure $LS\Theta GG$ is low-weighted.*

Theorem 4 *Assuming that both the ID and the geometry position can be represented by $\log n$ bits each, the total number of messages during constructing the structure $LS\Theta GG$ is in the range of $[5n, 13n]$, where each message has at most $O(\log n)$ bits.*

Compared with previous known low-weighted structures [10, 12], $LS\Theta GG$ not only achieves more desirable properties, but also costs much less messages during construction. To construct $LS\Theta GG$, each node only needs to collect the information $E_2(x)$ which costs at most $6n$ messages for n nodes. The Algorithm 2 can be generally applied to any known degree-bounded planar spanner to make it low-weighted while keeping all its previous properties, except increasing the spanning ratio from ρ to $2\rho + 1$ theoretically.

In addition, the expected average node interference in the structure is bounded by a small constant. This is significant on its own due to the following reasons: it has been taken for granted that "*a network topology with small logical node degree will guarantee a small interference*" and recently Burkhart et al. [3] showed that this is not true generally. This work also shows that, although generally a small logical node degree cannot guarantee a small interference, the expected average interference is indeed small if the logical communication neighbors are chosen carefully.

Theorem 5 *For a set of nodes produced by a Poisson point process with density n, the expected maximum node interferences of EMST, GG, RNG, and Yao are at least $\Theta(\log n)$.*

Theorem 6 *For a set of nodes produced by a Poisson point process with density n, the expected average node interferences of EMST are bounded from above by a constant.*

This result also holds for nodes deployed with uniform random distribution.

Applications

Localized topology control in wireless ad hoc networks are critical mechanisms to maintain network connectivity and provide feedback to communication protocols. The major traffic in networks are unicast communications. There is a compelling need to conserve energy and improve network performance by maintaining an energy-efficient topology in localized ways. This algorithm achieves this by choosing relatively smaller power levels and size of communication neighbors for each node (e.g., reducing interference). Also, broadcasting is often necessary in MANET routing protocols. For example, many unicast routing protocols

such as Dynamic Source Routing (DSR), Ad Hoc On Demand Distance Vector (AODV), Zone Routing Protocol (ZRP), and Location Aided Routing (LAR) use broadcasting or a derivation of it to establish routes. It is highly important to use power-efficient broadcast algorithms for such networks since wireless devices are often powered by batteries only.

Cross-References

- ▸ Applications of Geometric Spanner Networks
- ▸ Geometric Spanners
- ▸ Planar Geometric Spanners
- ▸ Sparse Graph Spanners

Recommended Reading

1. Bose P, Gudmundsson J, Smid M (2002) Constructing plane spanners of bounded degree and low weight. In: Proceedings of European symposium of algorithms, University of Rome, 17–21 Sept 2002
2. Bose P, Morin P, Stojmenovic I, Urrutia J (2001) Routing with guaranteed delivery in ad hoc wireless networks. ACM/Kluwer Wirel Netw 7(6):609–616, 3rd international workshop on discrete algorithms and methods for mobile computing and communications, pp 48–55 (1999)
3. Burkhart M, von Rickenbach P, Wattenhofer R, Zollinger A (2004) Does topology control reduce interference. In: ACM international symposium on mobile Ad-Hoc networking and computing (MobiHoc), Tokyo, 24–26 May 2004
4. Gabriel KR, Sokal RR (1969) A new statistical approach to geographic variation analysis. Syst Zool 18:259–278
5. Karp B, Kung HT (2000) Gpsr: greedy perimeter stateless routing for wireless networks. In: Proceedings of the ACM/IEEE international conference on mobile computing and networking (MobiCom), Boston, 6–11 Aug 2000
6. Kleinrock L, Silvester J (1978) Optimum transmission radii for packet radio networks or why six is a magic number. In: Proceedings of the IEEE national telecommunications conference, Birmingham, pp 431–435, 4–6 Dec 1978
7. Kuhn F, Wattenhofer R, Zollinger A (2002) Asymptotically optimal geometric mobile ad-hoc routing. In: International workshop on discrete algorithms and methods for mobile computing and communications (DIALM), Atlanta, 28 Sept 2002
8. Kuhn F, Wattenhofer R, Zollinger A (2003) Worst-case optimal and average-case efficient geometric ad-hoc routing. In: ACM international symposium on mobile Ad-Hoc networking and computing (MobiHoc), Anapolis, 1–3 June 2003
9. Li L, Halpern JY, Bahl P, Wang Y-M, Wattenhofer R (2001) Analysis of a cone-based distributed topology control algorithms for wireless multi-hop networks. In: PODC: ACM symposium on principle of distributed computing, Newport, 26–29 Aug 2001
10. Li X-Y (2003) Approximate MST for UDG locally. In: COCOON, Big Sky, 25–28 July 2003
11. Li X-Y, Wan P-J, Wang Y, Frieder O (2002) Sparse power efficient topology for wireless networks. In: IEEE Hawaii international conference on system sciences (HICSS), Big Island, 7–10 Jan 2002
12. Li X-Y, Wang Y, Song W-Z, Wan P-J, Frieder O (2004) Localized minimum spanning tree and its applications in wireless ad hoc networks. In: IEEE INFOCOM, Hong Kong, 7–11 Mar 2004
13. Song W-Z, Wang Y, Li X-Y, Frieder O (2004) Localized algorithms for energy efficient topology in wireless ad hoc networks. In: ACM international symposium on mobile Ad-Hoc networking and computing (MobiHoc), Tokyo, 24–26 May 2004
14. Toussaint GT (1980) The relative neighborhood graph of a finite planar set. Pattern Recognit 12(4):261–268
15. Wan P-J, Calinescu G, Li X-Y, Frieder O (2001) Minimum-energy broadcast routing in static ad hoc wireless networks. ACM wireless networks (2002), To appear, Preliminary version appeared in IEEE INFOCOM, Anchorage, 22–26 Apr 2001
16. Wang Y, Li X-Y (2003) Efficient construction of bounded degree and planar spanner for wireless networks. In: ACM DIALMPOMC joint workshop on foundations of mobile computing, San Diego, 19 Sept 2003
17. Yao AC-C (1982) On constructing minimum spanning trees in k-dimensional spaces and related problems. SIAM J Comput 11:721–736

Degree-Bounded Trees

Martin Fürer
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

Keywords

Bounded degree spanning trees; Bounded degree Steiner trees

Years and Authors of Summarized Original Work

1994; Fürer, Raghavachari

Problem Definition

The problem is to construct a spanning tree of small degree for a connected undirected graph $G = (V, E)$. In the Steiner version of the problem, a set of distinguished vertices $D \subseteq V$ is given along with the input graph G. A Steiner tree is a tree in G which spans at least the set D.

As finding a spanning or Steiner tree of the smallest possible degree Δ^* is *NP*-hard, one is interested in approximating this minimization problem. For many such combinatorial optimization problems, the goal is to find an approximation in polynomial time (a constant or larger factor). For the spanning and Steiner tree problems, the iterative polynomial time approximation algorithms of Fürer and Raghavachari [8] (see also [14]) find much better solutions. The degree Δ of the solution tree is at most $\Delta^* + 1$.

There are very few natural *NP*-hard optimization problems for which the optimum can be achieved up to an additive term of 1. One such problem is coloring a planar graph, where coloring with four colors can be done in polynomial time. On the other hand, 3-coloring is *NP*-complete even for planar graphs. An other such problem is edge coloring a graph of degree Δ. While coloring with $\Delta + 1$ colors is always possible in polynomial time, Δ edge coloring is *NP*-complete.

Chvátal [3] has defined the *toughness* $\tau(G)$ of a graph as the minimum ratio $|X|/c(X)$ such that the subgraph of G induced by $V \setminus X$ has $c(X) \geq 2$ connected components. The inequality $1/\tau(G) \leq \Delta^*$ immediately follows. Win [17] has shown that $\Delta^* < \frac{1}{\tau(G)} + 3$; i.e., the inverse of the toughness is actually a good approximation of Δ^*.

A set X, such that the ratio $|X|/c(X)$ is the toughness $\tau(G)$, can be viewed as witnessing the upper bound $|X|/c(X)$ on $\tau(G)$ and therefore the lower bound $c(X)/|X|$ on Δ^*. Strengthening this notion, Fürer and Raghavachari [8] define X to be a witness set for $\Delta^* \geq d$ if d is the smallest integer greater or equal to $(|X| + c(X) - 1)/|X|$. Their algorithm not only outputs a spanning tree, but also a witness set X, proving that its degree is at most $\Delta^* + 1$.

Key Results

The minimum degree spanning tree and Steiner tree problems are easily seen to be *NP*-hard, as they contain the Hamiltonian path problem. Hence, we cannot expect a polynomial time algorithm to find a solution of minimal possible degree Δ^*. The same argument also shows that an approximation by a factor less than $3/2$ is impossible in polynomial time unless $P = NP$.

Initial approximation algorithms obtained solutions of degree $O(\Delta^* \log n)$ [6], where $n = |V|$ is the number of vertices. The optimal result for the spanning tree case has been obtained by Fürer and Raghavachari [7, 8].

Theorem 1 *Let Δ^* be the degree of an unknown minimum degree spanning tree of an input graph $G = (V, E)$. There is a polynomial time approximation algorithm for the minimum degree spanning tree problem that finds a spanning tree of degree at most $\Delta^* + 1$.*

Later this result has been extended to the Steiner tree case [8].

Theorem 2 *Assume a Steiner tree problem is defined by a graph $G = (V, E)$ and an arbitrary subset D of vertices V. Let Δ^* be the degree of an unknown minimum degree Steiner tree of G spanning at least the set D. There is a polynomial time approximation algorithm for the minimum degree Steiner tree problem that finds a Steiner tree of degree at most $\Delta^* + 1$.*

Both approximation algorithms run in time $O(mn \log n\, \alpha(m, n))$, where m is the number of edges and α is the inverse Ackermann function.

Applications

Some possible direct applications are in networks for noncritical broadcasting, where it might be desirable to bound the load per node, and in designing power grids, where the cost of splitting increases with the degree. Another major benefit of a small degree network is limiting the effect of node failure.

Furthermore, the main results on approximating the minimum degree spanning and Steiner tree problems have been the basis for approximating various network design problems, sometimes involving additional parameters.

Klein, Krishnan, Raghavachari and Ravi [11] find 2-connected subgraphs of approximately minimal degree in 2-connected graphs, as well as approximately minimal degree spanning trees (branchings) in directed graphs. Their algorithms run in quasi-polynomial time, and approximate the degree Δ^* by $(1+\epsilon)\Delta^* + O(\log_{1+\epsilon} n)$.

Often the goal is to find a spanning tree that simultaneously has a small degree and a small weight. For a graph having an minimum weight spanning tree (MST) of degree Δ^* and weight w, Fischer [5] finds a spanning tree with degree $O(\Delta^* + \log n)$ and weight w, (i.e., an MST of small weight) in polynomial time.

Könemann and Ravi [12, 13] provide a bi-criteria approximation. For a given $B^* \geq \Delta^*$, let w be the minimum weight of any spanning tree of degree at most B^*. The polynomial time algorithm finds a spanning tree of degree $O(B^* + \log n)$ and weight $O(w)$. In the second paper, the algorithm adapts to the case of a different degree bound on each vertex. Chaudhuri et al. [2] further improved this result to approximate both the degree B^* and the weight w by a constant factor.

In another extension of the minimum degree spanning tree problem, Ravi and Singh [15] have obtained a strict generalization of the $\Delta^* + 1$ spanning tree approximation [8]. Their polynomial time algorithm finds an MST of degree $\Delta^* + k$ for the case of a graph with k distinct weights on the edges.

Recently, there have been some drastic improvements. Again, let w be the minimum cost of a spanning tree of given degree B^*. Goemans [9] obtains a spanning tree of cost w and degree $B^* + 2$. Finally, Singh and Lau [16] decrease the degree to $B^* + 1$ and also handle individual degree bounds Δ_v^* for each vertex v in the same way.

Interesting approximation algorithms are also known for the 2-dimensional Euclidian minimum weight bounded degree spanning tree problem, where the vertices are points in the plane and edge weights are the Euclidian distances. Khuller, Raghavachari, and Young [10] show factor 1.5 and 1.25 approximations for degree bounds 3 and 4 respectively. These bounds have later been improved slightly by Chan [1]. Slightly weaker results are obtained by Fekete et al. [4], using flow-based methods, for the more general case where the weight function just satisfies the triangle inequality.

Open Problems

The time complexity of the minimum degree spanning and Steiner tree algorithms [8] is $O(mn\,\alpha(m,n)\log n)$. Can it be improved to $O(mn)$? In particular, what can be gained by initially selecting a reasonable Steiner tree with some greedy technique instead of starting the iteration with an arbitrary Steiner tree?

Is there an efficient parallel algorithm that can obtain a $\Delta^* + 1$ approximation in polylogarithmic time? Fürer and Raghavachari [6] have obtained such an NC-algorithm, but only with a factor $O(\log n)$ approximation of the degree.

Cross-References

- Fully Dynamic Connectivity
- Graph Connectivity
- Minimum Energy Cost Broadcasting in Wireless Networks
- Minimum Spanning Trees
- Steiner Forest
- Steiner Trees

Recommended Reading

1. Chan TM (2004) Euclidean bounded-degree spanning tree ratios. Discret Comput Geom 32(2):177–194
2. Chaudhuri K, Rao S, Riesenfeld S, Talwar K (2006) A push-relabel algorithm for approximating degree bounded MSTs. In: Proceedings of the 33rd international colloquium on automata, languages and programming (ICALP 2006), Part I. LNCS, vol 4051. Springer, Berlin, pp 191–201
3. Chvátal V (1973) Tough graphs and Hamiltonian circuits. Discret Math 5:215–228
4. Fekete SP, Khuller S, Klemmstein M, Raghavachari B, Young N (1997) A network-flow technique for finding low-weightbounded-degree spanning trees. In: Proceedings of the 5th integer programming and combinatorial optimization conference (IPCO 1996), and J Algorithms 24(2):310–324
5. Fischer T (1993) Optimizing the degree of minimum weight spanning trees. Technical report TR93–1338. Computer Science Department, Cornell University
6. Fürer M, Raghavachari B (1990) An NC approximation algorithm for the minimum-degree spanning tree problem. In: Proceedings of the 28th annual Allerton conference on communication, control and computing, 1990, pp 174–281
7. Fürer M, Raghavachari B (1992) Approximating the minimum degree spanning tree to within one from the optimal degree. In: Proceedings of the third annual ACM-SIAM symposium on discrete algorithms (SODA 1992), pp 317–324
8. Fürer M, Raghavachari B (1994) Approximating the minimum-degree Steiner tree to within one of optimal. J Algorithms 17(3):409–423
9. Goemans MX (2006) Minimum bounded degree spanning trees. In: Proceedings of the 47th annual IEEE symposium on foundations of computer science (FOCS 2006), pp 273–282
10. Khuller S, Raghavachari B, Young N (1996) Low-degree spanning trees of small weight. SIAM J Comput 25(2):355–368
11. Klein PN, Krishnan R, Raghavachari B, Ravi R (2004) Approximation algorithms for finding low-degree subgraphs. Networks 44(3):203–215
12. Könemann J, Ravi R (2002) A matter of degree: improved approximation algorithms for degree-bounded minimum spanning trees. SIAM J Comput 31(6):1783–1793
13. Könemann J, Ravi R (2005) Primal-dual meets local search: approximating MSTs with nonuniform degree bounds. SIAM J Comput 34(3):763–773
14. Raghavachari B (1995) Algorithms for finding low degree structures. In: Hochbaum DS (ed) Approximation algorithms for NP-hard problems. PWS Publishing Company, Boston, pp 266–295
15. Ravi R, Singh M (2006) Delegate and conquer: an LP-based approximation algorithm for minimum degree MSTs. In: Proceedings of the 33rd international colloquiumon automata, languages and programming (ICALP 2006) Part I. LNCS, vol 4051. Springer, Berlin, pp 169–180
16. Singh M, Lau LC (2007) Approximating minimum bounded degree spanning trees to within one of optimal. In: Proceedings of the thirty-ninth annual ACM symposiumon theory of computing (STOC 2007), New York, pp 661–670
17. Win S (1989) On a connection between the existence of k-trees and the toughness of a graph. Graphs Comb 5(1):201–205

Delaunay Triangulation and Randomized Constructions

Olivier Devillers
Inria Nancy – Grand-Est, Villers-lès-Nancy, France

Keywords

Convex hull; Delaunay triangulation; Randomization; Voronoi diagram

Years and Authors of Summarized Original Work

1989; Clarkson, Shor
1993; Seidel
2002; Devillers
2003;Amenta, Choi, Rote

Problem Definition

The Delaunay triangulation and the Voronoi diagram are two classic geometric structures in the field of computational geometry. Their success can perhaps be attributed to two main reasons: Firstly, there exist practical, efficient algorithms to construct them; and secondly, they have an enormous number of useful applications ranging from meshing and 3D-reconstruction to interpolation.

Given a set S of n sites in some space $\mathbb{E}$, we define the Voronoi region$V_S(p)$ of $p \in S$ to be

the set of points in $\mathbb{E}$ whose nearest neighbor in S is p (for some distance δ):

$$V(p) = \left\{x \in \mathbb{E}, \forall q \in S \setminus \{p\} \delta(x,p) < \delta(x,q)\right\}.$$

It is easily seen that these regions form a partition of $\mathbb{E}$ into convex regions which we refer to as *cells*. These concepts may be extended into more exotic spaces such as periodic and hyperbolic spaces or metric spaces using convex distances, though we restrict ourselves to the case where $\mathbb{E}$ is the Euclidean space $\mathbb{E} = \mathbb{R}^d$ and the distance δ is the L_2 norm.

The Voronoi diagram $\mathbb{V}(S)$ may now be defined as the limit between the different Voronoi cells

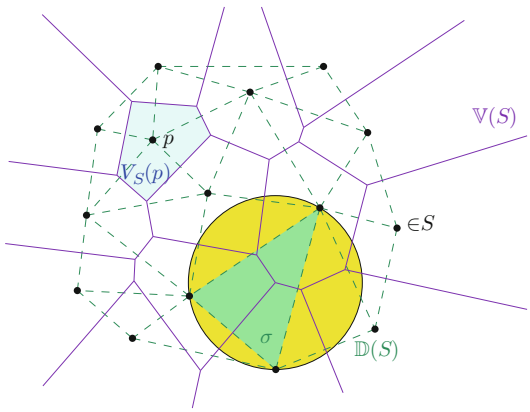
$$\mathbb{V}(S) = \mathbb{E} \setminus \bigcup_{p \in S} V_S(p).$$

The Delaunay triangulation $\mathbb{D}(S)$ is the geometric dual of $\mathbb{V}(S)$. More formally, $\mathbb{D}(S)$ is a simplicial complex defined by

$$\sigma \in \mathbb{D}(S) \Longleftrightarrow \bigcap_{p \in \sigma} \overline{V_S}(p) \neq \emptyset,$$

where $\overline{V_S}(p)$ is the closure of the Voronoi cell $V_S(p)$ (see Fig. 1).

Voronoi diagrams and Delaunay triangulations have received a lot of attention in the literature



Delaunay Triangulation and Randomized Constructions, Fig. 1 The Voronoi diagram of a set S of 15 points and its dual Delaunay triangulation

with several surveys, books, or book chapters (e.g., [4, 14]) and hundreds of papers. In this article, we will focus on randomized construction algorithms for the Delaunay triangulation. Such algorithms use randomness to speed up their running time but do not assume any randomness in the data distribution.

Key Results

Delaunay Properties

Empty Ball Property

One crucial property of the Delaunay triangulation, which is the basis of many algorithms, is the *empty ball property*, which guarantees that a triangle is a Delaunay triangle of S if and only if the interior of its circumball does not contain any point of S.

Size of the Triangulation

In the plane, the combinatorial properties of a triangulation (not necessarily Delaunay) are completely fixed by the Euler relation. In particular, given n vertices, h of which on the convex hull, every triangulation must have $2n-2-h$ triangles and $3n-3-h$ edges. In dimension d, the Dehn-Sommerville relations yield a linear dependence for the number of simplices of all dimensions on the number of simplices of dimensions k for $k \leq \left\lceil \frac{d}{2} \right\rceil$; this gives an $O\left(n^{\left\lceil \frac{d}{2} \right\rceil}\right)$ upper bound for the number of simplices of all dimension. For both Delaunay and more general triangulations, these bounds are tight in the worst case.

These bounds can be tightened given some assumptions on the distribution of the input sites. If the points are uniformly distributed in a compact convex of fixed volume, then the triangulation size (its total number of simplices) is $\Theta(n)$, with a constant exponential in d [9]. In 3D, and for reconstruction purposes, it is convenient to assume that the points lie on a surface. It is known that the Delaunay triangulation of points uniformly distributed on a convex polyhedron has size $\Theta(n)$ (for a constant depending on the polyhedron

complexity). For points uniformly distributed on a (non convex) polyhedron, the triangulation's size is between $\Omega(n)$ and $O(n \log n)$ [12]. If, instead of making a probabilistic assumption, we assume that the points are a "good sampling" of the surface such that every small ball centered on the surface contains between 1 and κ points (where κ is a constant), then the size of the Delaunay triangulation is $\Theta(n)$ for a polyhedron, $O(n \log n)$ for a *generic* smooth surface [3], and $\Omega(n\sqrt{n})$ for a nongeneric surface (e.g., a cylinder). In the case of the cylinder, a uniformly distributed point set has a triangulation of size $\Theta(n \log n)$. In dimension d, a p-dimensional polyhedron whose faces have a "good sampling" has size $O(n^k)$ where $k = \frac{d+1-\left\lceil \frac{d+1}{p+1} \right\rceil}{p}$ [2].

First Algorithms

Many classical techniques in algorithmic and computational geometry have been used to attack the problem of constructing the Delaunay triangulation and the Voronoi Diagram. The gift wrapping and the incremental approaches were introduced in the 1970s [11], followed by some worst case optimal algorithms in 2D, based on divide-and-conquer [13] and sweep line techniques [10]. In higher dimensions, the optimal worst case construction of Delaunay triangulation and convex hulls was solved in the 1990s.

In the remainder of the entry, we will describe some further algorithmic techniques that may be used to construct the Delaunay triangulation.

Randomized Construction

One popular and efficient method, applied to the Delaunay triangulation at the end of the 1980s [5], is Randomized Incremental Construction (RIC). The idea is to exploit the simplicity of an incremental algorithm while avoiding its worst case behavior by simply adjusting the order of insertion of the points.

Conflict Graph

Recalling that $\mathbb{D}(S)$ is the set of triangles with vertices in S whose circumballs are empty, the idea is to maintain for a sequence $\varnothing = S_0 \subset S_1 \subset S_2 \subset \ldots \subset S_n = S$, where $|S_i| = i$, a sequence of triangulations $\mathbb{D}(S_i)$ with associated *conflict graphs*. We define the conflict graph to be a bipartite graph that links a point p of $S \setminus S_i$ to a simplex σ in $\mathbb{D}(S_i)$ if the circumball of σ contains p (p *and* σ *are called in conflict*). The information contained in the conflict graph simplifies the construction of $\mathbb{D}(S_{i+1})$ from $\mathbb{D}(S_i)$ since it gives directly the simplices in $\mathbb{D}(S_i) \setminus \mathbb{D}(S_{i+1})$.

The key point comes from an analysis based on random sampling [6]; let's assume that S_i is a random sample of size i of S. We say that a simplex has *width* j if it has j points in conflict in S. In which case a Delaunay simplex is a simplex of width 0. Denote by Δ_j the number of simplices of width j and let $\Delta_{\leq k} = \sum_{j \leq k} \Delta_j$. We first bound $\Delta_{\leq k}$ using the following remark: a simplex of width j is a Delaunay simplex of a random sample R of size $\frac{n}{k}$ of S with probability $p_k = \frac{1}{k^{d+1}} \left(1 - \frac{1}{k}\right)^j$ (vertices of σ must be chosen in R and points in conflict must not). Notice that for $j \in [2 \ldots k]$, we have $\left(1 - \frac{1}{k}\right)^j \geq \left(1 - \frac{1}{k}\right)^k \geq \frac{1}{4}$ since $\left(1 - \frac{1}{x}\right)^x$ is an increasing function of value $\frac{1}{4}$ for $x = 2$. For $k \geq 2$, we have (using $\mathbb{P}$ to denote the probability measure)

$$|\mathbb{D}(R)| = O\left(\frac{n}{k}^{\lceil \frac{d}{2} \rceil}\right) = \sum_{\sigma \in S^{d+1}} \mathbb{P}(\sigma \in \mathbb{D}(R)) = \sum_{j \leq n} \Delta_j p_j \geq \frac{\Delta_0}{k^{d+1}} + \frac{\Delta_1 \frac{1}{2}}{k^{d+1}}$$

$$+ \sum_{j=2}^{k} \frac{\Delta_j \frac{1}{4}}{k^{d+1}} \geq \frac{\Delta_{\leq k}}{4k^{d+1}}, \Delta_{\leq k} \leq O\left(\frac{n}{k}^{\lceil \frac{d}{2} \rceil} k^{d+1}\right) = O\left(n^{\lceil \frac{d}{2} \rceil} k^{\lceil \frac{d+1}{2} \rceil}\right).$$

We can now analyze the incremental construction of $\mathbb{D}(S)$. The probability that a triangle of width j appears during the construction is

$$p'_j = \frac{\binom{d+1}{j+d+1}}{(j+d+1)!}$$

(the number of permutations that look at the vertices of σ before the points in conflict divided by the total number of permutations). Then the cost of the algorithm is given by the total number of conflicts occurring during the construction:

$$\sum_{\sigma \in S^{d+1}} \text{width}\,(\sigma)\mathbb{P}(\sigma \text{ appears}) = \sum_j \Delta_j \cdot j \cdot p'_j = \sum_j (\Delta_{\le j} - \Delta_{\le j-1}) j \cdot p'_j$$

$$= \sum_j \Delta_{\le j}(j \cdot p'_j - (j+1)p'_{j+1}) \le \sum_j n^{\lceil \frac{d}{2} \rceil} j^{\lceil \frac{d+1}{2} \rceil} O\left(\frac{1}{j^{d+2}}\right) \le O\left(n^{\lceil \frac{d}{2} \rceil} \sum_j j^{-\lceil \frac{d}{2} \rceil}\right)$$

which gives $O(n \log n)$ for $d = 2$ and $O\left(n^{\lceil \frac{d}{2} \rceil}\right)$ for higher d.

Backward Analysis

A simpler way of analyzing RIC is backward analysis [15], and we will sketch it in 2D. The idea is quite simple and consists in asking: *what is the cost of the last step?* The answer is that the cost of modifying the triangulation during last insertion is clearly proportional to the *degree* (number of simplices incident) of the last point inserted into the triangulation. Since the last point is a random point, its expected degree is

$$\mathbb{E}\text{xp}(\text{degree(last point)}) = \mathbb{E}\text{xp}(\text{degree(random point)}) = \frac{1}{n}\sum_{p \in S} \text{degree}(p) \le \frac{6n}{n} = 6,$$

and summing over all insertion steps gives a linear cost for updating the triangulation. It remains to count the cost of updating the conflict graph. We remark that there is a conflict between the last point p_n and a triangle created by the insertion of the jth point p_j if and only if the edge $p_j p_n$ exists in $\mathbb{D}(S_j \cup \{p_n\})$. Since p_j and p_n are both random points of $S_j \cup \{p_n\}$, it happens with probability $O\left(\frac{1}{j}\right)$, the expected number of conflicts for p_n is thus $O\left(\sum_j \frac{1}{j}\right) = O(\log n)$, and the total number of conflicts is $O\left(\sum_k \log k\right) = O(n \log n)$.

Delaunay Hierarchy

The conflict graph approach assumes complete knowledge of S to initialize the conflict graph. Using a lazy approach and postponing the conflict determination, it is possible to obtain online algorithms [5].

Among the online schemes to construct the Delaunay triangulation, the Delaunay hierarchy [7] gives good results both in theory and in practice. The Delaunay hierarchy constructs a sequence of random samples $S = S_0 \supset S_1 \supset \cdots \supset S_h$ such that $\mathbb{P}(p \in S_i \mid p \in S_{i-1}) = \alpha$. Then the Delaunay triangulations of $\mathbb{D}(S_i), 1 \le i \le h$ are maintained under point insertions. Pointers from a vertex of $\mathbb{D}(S_i)$ to the vertices at the same position in $\mathbb{D}(S_{i-1})$ (if $i < h$) and in $\mathbb{D}(S_{i+1})$ (if it actually belongs to S_{i+1}) are also computed.

When a new point p needs to be inserted, it is located by walking in $\mathbb{D}(S_h)$ (using neighborhood relations) to reach the closest vertex w_h of p in $\mathbb{D}(S_h)$. Then the hierarchy is descended, walking in $\mathbb{D}(S_i)$ from w_{i+1} to find w_i the closest neighbor in sample S_i. Using these neighbors, it is easy to insert p in $\mathbb{D}(S_h)$ and in the triangulation of other samples that the random process assigns to p.

In 2D, the expected cost of the walk at any level is $O(\alpha^{-1})$ and the expected value for h is

$O\left(\frac{\log n}{-\log \alpha}\right)$. Thus the theoretical complexity of the algorithm is $O(n \log n)$. The value of α can be optimized depending on the input distribution: for random points $\alpha = \frac{1}{30}$ gives good timings and a very low memory requirement in addition to the one needed for $\mathbb{D}(S)$.

A Less Randomized Construction

Constructing the Delaunay triangulation by inserting the points in a random order presents a drawback with respect to memory management. Since the inserted point is random in S, there is very little chance that the triangles needed are present in the cache memory. So, an idea is to sort the points using a space-filling curve (see Fig. 2-left) to ensure locality of the insertions. Unfortunately, when inserting the points in such an order, the randomized complexity results no longer apply and the number of created and destroyed triangles during the construction may explode on certain data sets.

A smart solution has been proposed: it is possible to use an insertion order random enough to apply randomized complexity results and allow some locality to benefit from cache memory. BRIO (Biased Randomized Insertion Order) [1] proposes to partition S in a set of random samples $S = \bigcup_{0 \le i \le h} S_i$ such that $|S_i| = \alpha |S_{i+1}|$ for $\alpha \le 1$, a small constant (e.g., $\alpha = \frac{1}{4}$), and to insert the samples by increasing size, each sample being sorted using a spatial filling curve (see Fig. 2-right).

In the random setting, we have seen that the probability for a triangle of width j to appear in the conflict graph algorithm was $\frac{1 \cdot 2 \cdot 3}{(j+1)(j+2)(j+3)} = \Theta(j^{-3})$. Using BRIO, this probability is a bit less intricate to compute, but it can be bounded in terms of α and it can be shown that it is still $\Theta(j^{-3})$ and thus randomized complexity results still apply.

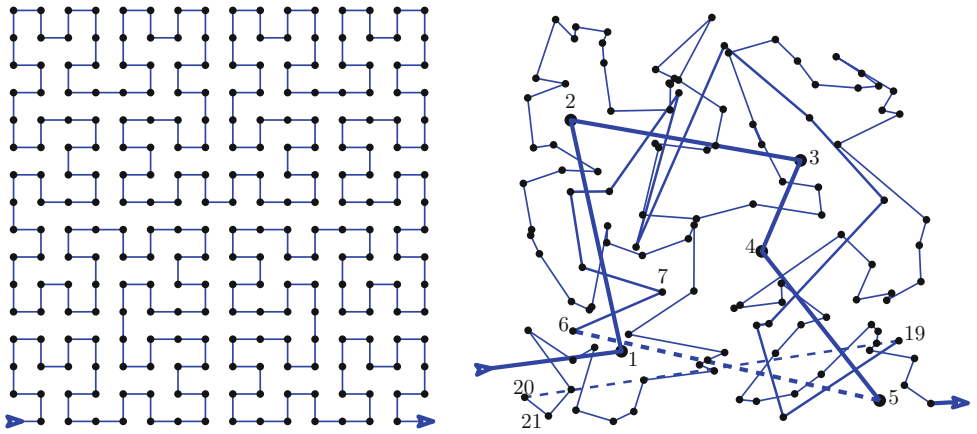
Experimental Results

On a 16 GB, 2.3 GHz desktop CGAL currently computes the Delaunay triangulation of up to 200M points in 2D and 50M points in 3D [8].

Static timings are almost constant with respect to the total number of points and are about 1 µs per point in 2D and 8 µs per point in 3D. In the dynamic setting, one million points are processed in 6s in 2D and 25s in 3D.

URLs to Code and Data Sets

CGAL, among a big collection of computational geometry algorithms, provides implementations for Delaunay triangulations in 2D, 3D, and general dimension. It computes the Delaunay triangulation in 2D and 3D using the Delaunay hierarchy in a dynamic setting and using BRIO for static computation (http://cgal.org).



Delaunay Triangulation and Randomized Constructions, Fig. 2 *Right*: the Hilbert space-filling curve. *Left*: random points sorted with BRIO

Cross-References

- Triangulation Data Structures
- Voronoi Diagrams and Delaunay Triangulations

Recommended Reading

1. Amenta N, Choi S, Rote G (2003) Incremental constructions con BRIO. In: Proceedings of the 19th annual symposium on computational geometry, San Diego, pp 211–219. doi:10.1145/777792.777824, http://page.inf.fu-berlin.de/~rote/Papers/pdf/Incremental+constructions+con+BRIO.pdf
2. Amenta N, Attali D, Devillers O (2012) A tight bound for the Delaunay triangulation of points on a polyhedron. Discret Comput Geom 48:19–38. doi:10.1007/s00454-012-9415-7, http://hal.inria.fr/hal-00784900
3. Attali D, Boissonnat JD, Lieutier A (2003) Complexity of the Delaunay triangulation of points on surfaces: the smooth case. In: Proceedings of the 19th annual symposium on computational geometry, San Diego, pp 201–210. doi:10.1145/777792.777823, http://dl.acm.org/citation.cfm?id=777823
4. Aurenhammer F, Klein R (2000) Voronoi diagrams. In: Sack JR, Urrutia J (eds) Handbook of computational geometry. Elsevier/North-Holland, Amsterdam, pp 201–290. ftp://ftp.cis.upenn.edu/pub/cis610/public_html/ak-vd-00.ps
5. Boissonnat JD, Teillaud M (1986) A hierarchical representation of objects: the Delaunay tree. In: Proceedings of the 2nd annual symposium computational geometry, Yorktown Heights, pp 260–268. http://dl.acm.org/citation.cfm?id=10543
6. Clarkson KL, Shor PW (1989) Applications of random sampling in computational geometry, II. Discret Comput Geom 4:387–421. doi:10.1007/BF02187740, http://www.springerlink.com/content/b9n24vr730825p71/
7. Devillers O (2002) The Delaunay hierarchy. Int J Found Comput Sci 13:163–180. doi:10.1142/S0129054102001035, http://hal.inria.fr/inria-00166711
8. Devillers O (2012) Delaunay triangulations, theory vs practice. In: Abstracts 28th European workshop on computational geometry, Assisi, pp 1–4. http://hal.inria.fr/hal-00850561, invited talk
9. Dwyer R (1993) The expected number of k-faces of a Voronoi diagram. Int J Comput Math 26(5):13–21. doi:10.1016/0898-1221(93)90068-7, http://www.sciencedirect.com/science/article/pii/0898122193900687
10. Fortune SJ (1987) A sweepline algorithm for Voronoi diagrams. Algorithmica 2:153–174. doi:10.1007/BF01840357, http://www.springerlink.com/content/n88186tl165168rw/
11. Frederick CO, Wong YC, Edge FW (1970) Two-dimensional automatic mesh generation for structural analysis. Internat J Numer Methods Eng 2:133–144. doi:10.1002/nme.1620020112/abstract
12. Golin MJ, Na HS (2002) The probabilistic complexity of the Voronoi diagram of points on a polyhedron. In: Proceedings of the 18th annual symposium on computational geometry. Barcelona http://www.cse.ust.hk/~golin/pubs/SCG_02.pdf
13. Lee DT (1978) Proximity and reachability in the plane. PhD thesis, Coordinated Science Lab., University of Illinois, Urbana. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA069764
14. Okabe A, Boots B, Sugihara K (1992) Spatial tessellations: concepts and applications of Voronoi diagrams. Wiley, Chichester
15. Seidel R (1993) Backwards analysis of randomized geometric algorithms. In: Pach J (ed) New trends in discrete and computational geometry, algorithms and combinatorics, vol 10. Springer, pp 37–68. http://ftp.icsi.berkeley.edu/ftp/pub/techreports/1992/tr-92-014.ps.gz

Derandomization of *k*-SAT Algorithm

Dominik Scheder
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
Institute for Computer Science, Shanghai Jiaotong University, Shanghai, China

Keywords

Algorithms for satisfiability; Derandomization; Local search

Years and Authors of Summarized Original Work

2011; Moser, Scheder

Problem Definition

Satisfiability is *the* central NP-complete problem. Given a Boolean formula in conjunctive normal form, for example, $(x \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{z}) \wedge \ldots$,

decide whether there is a satisfying assignment. An important subclass is k-SAT, where the input is restricted to k-CNF formulas: CNF formulas in which every clause has at most k literals. In 1999, Uwe Schöning [6] gave an extremely simple randomized algorithm for k-SAT of running time

$$\left(\frac{2(k-1)}{k}\right)^n \text{poly}(n).$$

In particular this solves 3-SAT in time $O^*(1.334^n)$, 4-SAT in $O^*(1.5^n)$ for 4-SAT, and so on (we use O^* to suppress polynomial factors in n). Several authors have attempted to derandomize Schöning's algorithm, albeit at the cost of a greater running time: an algorithm of Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan, and Schöning [2] runs in time $O^*((2k/(k+1))^n)$, which for $k = 3$ is $O^*(1.5^n)$. For $k = 3$ Brueggemann and Kern [1] achieve $O(1.473^n)$; Scheder [5] achieves $O(1.465^n)$; Kutzkov and Scheder [3] reduced this to $O(1.439^n)$. All improvements suffer from two drawbacks: they fall short of achieving the running time of Schöning's randomized algorithm; most of them are tailored to $k = 3$; finally, they are all fairly complicated.

Key Results

We describe a simple deterministic algorithm due to Moser and Scheder [4] with a running time that matches that of Schöning's up to subexponential overhead. That is, we prove the following theorem:

Theorem 1 *There is a deterministic algorithm deciding satisfiability of k-CNF formulas over n variables in time $\left(\frac{2(k-1)}{k}\right)^{n+o(n)}$.*

Notation

For a CNF formula F we denote by $\text{vbl}(F)$ the set of variables appearing in F. Usually $n = |\text{vbl}(F)|$ denotes the number of variables in a formula. By $\{0,1\}^{\text{vbl}(F)}$ (or $\{0,1\}^n$), we denote the set of all truth assignments to these variables. We make frequent use of the notation $F^{[u \mapsto b]}$, which is the CNF formula created from F by replacing every occurrence of the literal u by b and of $\bar{u}$ by $1-b$. For a literal u and a truth assignment α, we denote by $\alpha[u = b]$ the truth assignment that sets u to b and agrees with α otherwise.

A Promise Version of k-SAT

The heart of the proof will be an algorithm solving a promise version of k-SAT:

Theorem 2 *Let F be a k-CNF formula over n variables, $\alpha \in \{0,1\}^n$ a (not necessarily satisfying) assignment, and $r \in \mathbb{N}_0$. There is a deterministic algorithm* `sb-fast` *with the following properties:*

1. *If F is unsatisfiable,* `sb-fast` (F, r) *returns* `unsatisfiable`.
2. *If F has a satisfying assignment α^* with $d_H(\alpha, \alpha^*) \leq r$, then* `sb-fast`$(F, \alpha, r)$ *returns* `satisfiable`.
3. *Otherwise (i.e., if F is satisfiable but all satisfying assignments of F are too far from α), then* `sb-fast`(F, α, r) *might return* `unsatisfiable` *or* `satisfiable`.

Furthermore, `sb-fast` *runs in time* $(k-1)^{r+o(r)}\text{poly}(n)$.

The "inner random walk" in Schöning's algorithm has all properties stated in Theorem 2, except that it is randomized (with a small error probability). Combining Theorem 2 with the covering code machinery of Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan, and Schöning [2] yields our main result, Theorem 1.

Theorem 3 *Suppose there is an algorithm A which satisfies properties 1–4 of Theorem 2 and runs in time $c^r\text{poly}(n)$. Then there is an algorithm B solving k-SAT in time $\left(\frac{2c}{c+1}\right)^{n+o(n)}$. Furthermore, B is deterministic if A is.*

Plugging in $c = k-1$ yields Theorem 2.

Preliminaries: A Slower Algorithm

Dantsin et al. [2] give a deterministic algorithm, henceforth called `sb-slow`, satisfying Point 1–4 of Theorem 2 with a running time of $k^r \mathrm{poly}(n)$. We will start by explaining and analyzing this algorithm because our algorithm `sb-fast` uses it as a subroutine in case the input formula F is "well behaved."

> `Algorithm sb-slow`(F, α, r). F is a k-CNF formula over n variables, $\alpha \in \{0,1\}^{\mathrm{vbl}(F)}$ a truth assignment, and $r \in \mathbb{N}_0$.
>
> 1. If α satisfies F, return `satisfiable`.
> 2. Else if $r = 0$, return `unsatisfiable`.
> 3. Else:
>   (a) Pick some clause $C = (u_1 \vee \cdots \vee u_\ell)$ unsatisfied by α. Note that $\ell \le k$ holds in any case, but $\ell < k$ is possible.
>   (b) Set $F_i := F^{[u_i=1]}$.
>   (c) Call `sb-slow`$(F_i, \alpha, r-1)$ for all $1 \le i \le \ell$.
>   (d) If some of these ℓ recursive calls returns `satisfiable`, return `satisfiable`, otherwise return `unsatisfiable`.

It is obvious that `sb-slow` runs in time $k^r \mathrm{poly}(n)$. For correctness, note that `sb-slow` returns `unsatisfiable` if F is unsatisfiable. If F has a satisfying assignment $\alpha^* \neq \alpha$, let $(u_1 \vee \ldots \vee u_k)$ be the clause picked in step (3a). Now α^* satisfies some literal u_i in that clause, and thus the formula $F_i := F^{[u_i=1]}$ is satisfiable, as well. Since neither u_i nor $\bar{u}_i$ appears in F_i, we see that $\alpha^*[u_i = 0]$ satisfies F_i. Since $d_H(\alpha^*[u_i = 0], \alpha) = d_H(\alpha^*, \alpha) - 1 \le r - 1$, the call `sb-slow`$(F_i, \alpha, r-1)$ will return `satisfiable`.

Speeding Up the Algorithm

Let $k \in \mathbb{N}_0$ be fixed from now on.

Definition 1 Let F be a k-CNF formula and $\alpha \in \{0,1\}^{\mathrm{vbl}(F)}$. We say that F is good (with respect to α) if α satisfies all k-clauses of F (it might still violate smaller clauses).

Observe that if F is good, then $F^{[u=1]}$ is good for every literal u. If F is good with respect to α, then `sb-slow`(F, α, r) picks a clause of size at most $k-1$ in step (3a) and causes at most $k-1$ recursive calls, each again with a good formula:

Lemma 1 *Suppose F is a k-CNF that is good with respect to α. Then* `sb-slow` (F, α, r) *runs in time* $(k-1)^r \mathrm{poly}(n)$.

Great! The only thing that is left is to do something smart for formulas that are not good, i.e., have some unsatisfied clauses of size k. We will now describe how `sb-fast` proceeds and give a precise pseudocode description later. The algorithm `sb-fast` greedily finds a maximal set of pairwise disjoint k-clauses that are unsatisfied by α: $C_1, \ldots, C_m$, all over disjoint sets of variables. This can be done in polynomial time. Let β be an assignment to these variables.

Proposition 1 $F^{[\beta]}$ *is good with respect to α.*

This is easy to see: consider a k-clause C in F that is not satisfied by α. Then C might or might not be among $C_1, \ldots, C_m$, but by maximality it shares at least one variable with some i. This variable disappears in $F^{[\beta]}$, and C is either satisfied or shrinks to something smaller than k.

Fix $t = \lfloor \log_k \log_2 n \rfloor$. If $m \le t$ `sb-fast` can simply iterate over all $2^{km} \le 2^{kt} \le (\log_2 n)^k$ different assignments β to the variables in $C_1, \ldots, C_m$: each $F^{[\beta]}$ is good, and thus `sb-fast` $(F^{[\beta]}, \alpha, r)$ runs in time $(k-1)^r \mathrm{poly}(n)$. Also, if F is unsatisfiable then all $F^{[\beta]}$ are. If F has a satisfying assignment α^* with $d_H(\alpha^*, \alpha) \le r$, then at least one $F^{[\beta]}$ does, too. So this step is correct. We are left with the case that $m \ge t$. In this case we define $G := C_1, \ldots, C_t$.

k-ary Covering Codes

The set $[k]^t$ is endowed with a Hamming distance, just as the Hamming cube is: for

$w, w' \in [k]^t$, we define $d_H(w, w') := \left|\left\{i \in [t] \mid w_i \neq w'_i\right\}\right|$. The k-ary Hamming ball around w of radius s is the set $B_s^{(k)}(w) := \left\{w' \in [k]^t \mid d_H(w, w') \leq s\right\}$. The number of elements in such a ball is independent of w. We define and observe

$$\mathrm{vol}^{(k)}(t, s) := \left|B_s^{(k)}(w)\right| = \sum_{i=0}^{s} \binom{t}{i} (k-1)^i.$$

If $\mathcal{C} \subseteq [k]^t$ and $\cup_{w \in \mathcal{C}} B_s^{(k)}(w) = [k]^t$, we call $\mathcal{C}$ a k-ary code of length t and covering radius s. Using the probabilistic method it is easy to show the following result:

Lemma 2 *Let $t, k \in \mathbb{N}$, and $s \in \mathbb{N}_0$. There exists a k-ary code of length t and covering radius s with at most*

$$\left\lceil \frac{t \ln(k)\, k^t}{\mathrm{vol}^{(k)}(t, s)} \right\rceil$$

elements.

Observe that $k^t \leq \log_2 n$ and thus there are at most $2^{\log_2 n} = n$ subsets $\mathcal{C} \subseteq [k]^t$. We iterate through all of them and find a smallest code of covering radius s.

Consider $G = (C_1, \ldots, C_t)$, our maximal set of pairwise disjoint k-clauses not satisfied by α. Any satisfying assignment α^* of F must satisfy at least one literal in each C_i. Since they are pairwise disjoint, this implies $d_H(\alpha, \alpha^*) \geq t$. There are exactly k^t assignments that satisfy G and have distance exactly t from α. Each such assignment can be represented by a $w \in [k]^t$ in the obvious way. To be more precise, for $w \in [k]^t$ we define $\alpha[G, w]$ to be the assignment which we obtain from α by flipping the w_i^{th} literal in C_i, for $1 \leq i \leq t$. If G is understood from the context, we write $\alpha[w]$ instead of $\alpha[G, w]$.

Example 1 Consider $G = ((x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3))$, $\alpha = (0, \ldots, 0)$ and $t = 3$. Let $w = (2, 3, 3)$. Then $\alpha[w]$ is the assignment that sets y_1, z_2, and z_3 to 1 and all other variables to 0.

Proposition 2 *We observe the following facts about $\alpha[w]$:*

1. *$d_H(\alpha, \alpha[w]) = t$ for every $w \in [k]^t$.*
2. *If α^* satisfies F, then for some $w^* \in [k]^t$ we have $d_H(\alpha[w^*], \alpha^*) = d_H(\alpha, \alpha^*) - t$.*
3. *Let $w, w' \in [k]^t$. Then $d_H(\alpha[w], \alpha[w']) = 2 d_H(w, w')$.*

Lemma 3 *Let t and G be defined as above, and let $\mathcal{C} \subseteq [k]^t$ be a k-ary code of covering radius s. If α^* is a satisfying assignment of F, then there is some $w \in \mathcal{C}$ such that $d_H(\alpha[w^*], \alpha^*) \leq d_H(\alpha, \alpha^*) - t + 2s$.*

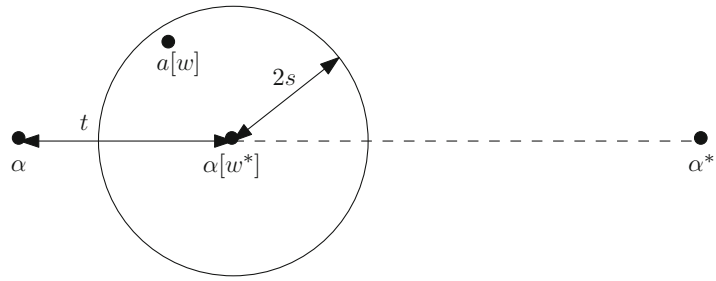
In particular, if $B_r(\alpha)$ contains a satisfying assignment, then there is some $w \in \mathcal{C}$ such that $B_{r-t+2s}(\alpha[w])$ contains it, too.

Proof (of Lemma 3) By Proposition 2, there is some $w^* \in [k]^t$ such that $d_H(\alpha[w^*], \alpha^*) = d_H(\alpha, \alpha^*) - t \leq r - t$. Since $\mathcal{C}$ has covering radius s, there is some $w \in \mathcal{C}$ such that $d_H(ww^*) \leq s$, and by Observation 2, $d_H(\alpha[w], \alpha[w^*]) \leq 2s$. The lemma now follows from the triangle inequality. The proof is illustrated in Fig. 1.

We now state `sb-fast` formally. We compute an optimal k-ary code $\mathcal{C}$ of length t and covering radius $s = t/k$ that is fixed throughout the run of the algorithm.

`Algorithm sb-fast`(F, α, r).

1. If α satisfies F, return `satisfiable`.
2. Else if $r = 0$, return `unsatisfiable`.
3. Else let G be maximal set of pairwise disjoint k-clauses of F unsatisfied by α.
4. If $|G| \geq t := \log_k \log_2 n$: Call `sb-slow`$(F^{[\beta]}, \alpha, r)$ for every $\beta \in \{0,1\}^{\mathrm{vbl}(G)}$ and return `satisfiable` iff at least one call returns `satisfiable`.
5. Else, if $|G| \geq t$ set $G = (C_1, \ldots, C_t)$ and call `sb-fast`$(F, \alpha[G, w], r - t + 2t/k)$ for every $w \in \mathcal{C}$ and return `satisfiable` iff at least one call returns `satisfiable`.



Derandomization of *k*-SAT Algorithm, Fig. 1 The distance from α^* to $\alpha[w]$ is at most the distance from α^* to $\alpha[w^*]$ plus $2s$

Correctness of `sb-fast` follows from the above discussion: If there is some $\alpha^* \in B_r(\alpha)$ that satisfies F, then for at least one $w \in \mathcal{C}$ it holds that $d_H(\alpha[w], \alpha^*) \leq r - t + 2t/k$, thus the corresponding recursive call to `sb-fast` will be successful.

What about the running time? If $|G| \leq t$, then every call to `sb-slow` $(F^{[\beta]}, \alpha, r)$ runs in time $O((k-l)^r \text{poly}(n))$. Otherwise, the procedure `sb-fast` calls itself recursively for each $w \in \mathcal{C}$. Every level further into the recursion, the parameter r decreases by $t - 2t/k$. The running time is therefore $|\mathcal{C}|^{r/(t-2t/k)}\,\text{poly}(n)$. To evaluate this we have to estimate the size of $\mathcal{C}$. Recall that $s = t/k$.

$$
\begin{aligned}
|\mathcal{C}| &\leq \frac{k^t \text{poly}(t)}{\text{vol}^{(k)}(t,s)} \leq \frac{k^t \text{poly}(t)}{\binom{t}{s}(k-1)^s} \\
&\leq \frac{k^t \text{poly}(t)}{\left(\frac{t}{s}\right)^s \left(\frac{t}{t-s}\right)^{t-s}(k-1)^s} \\
&= \frac{k^t \text{poly}(t)}{k^s \left(\frac{k}{k-1}\right)^{t-s}(k-1)^s} \\
&= (k-1)^{t-2s}\,\text{poly}(t).
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
|\mathcal{C}|^{r/(t-2t/k)} &\leq \left((k-1)^{t-2s}\,\text{poly}(t)\right)^{r/(t-2s)} \\
&= (k-1)^r\,\text{poly}(t)^{r/(t-2s)}.
\end{aligned}
$$

Since t is a growing function in n, the term $\text{poly}(t)^{1/(t-2s)}$ converges to 1 as n grows, and the running time is at most $(k-1)^{r+o(r)\text{poly}(n)}$. This completes the proof of Theorem 2.

Cross-References

- Backtracking Based k-SAT Algorithms
- Exact Algorithms and Strong Exponential Time Hypothesis
- Exact Algorithms and Time/Space Tradeoffs
- Exact Algorithms for General CNF SAT
- Exact Algorithms for k SAT Based on Local Search
- Exact Algorithms for Maximum Two-Satisfiability
- Exact Graph Coloring Using Inclusion-Exclusion
- Random Planted 3-SAT
- Thresholds of Random k-SAT
- Unique k-SAT and General k-SAT

Recommended Reading

1. Brueggemann T, Kern W (2004) An improved deterministic local search algorithm for 3-SAT. Theor Comput Sci 329(1–3):303–313
2. Dantsin E, Goerdt A, Hirsch EA, Kannan R, Kleinberg J, Papadimitriou C, Raghavan O, Schöning U (2002) A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search. Theor Comput Sci 289:69–83
3. Kutzkov K, Scheder D (2010) Using CSP to improve deterministic 3-SAT. CoRR abs/1007.1166
4. Moser RA, Scheder D (2011) A full derandomization of Schöning's k-SAT algorithm. In: Fortnow L, Vadhan SP (eds) Proceedings of the 43rd ACM symposium on theory of computing, STOC 2011, San Jose, 6–8 June 2011. ACM, pp 245–252
5. Scheder D (2008) Guided search and a faster deterministic algorithm for 3-SAT. In: Proceedings of the 8th Latin American symposium on theoretical informatics (LATIN'08), Búzios. Lecture notes in computer science, vol 4957, pp 60–71
6. Schöning U (1999) A probabilistic algorithm for k-SAT and constraint satisfaction problems. In: FOCS '99: proceedings of the 40th annual symposium on foundations of computer science, Washington, DC. IEEE Computer Society, p 410

Deterministic Broadcasting in Radio Networks

Leszek Gąsieniec
University of Liverpool, Liverpool, UK

Keywords

Dissemination of information; One-to-all communication; Wireless networks

Years and Authors of Summarized Original Work

2000; Chrobak, Gąsieniec, Rytter

Problem Definition

One of the most fundamental communication problems in wired as well as wireless networks is **broadcasting**, where one distinguished source node has a message that needs to be sent to all other nodes in the network.

The radio network abstraction captures the features of distributed communication networks with multi-access channels, with minimal assumptions on the channel model and processors' knowledge. Directed edges model unidirectional links, including situations in which one of two adjacent transmitters is more powerful than the other. In particular, there is no feedback mechanism (see, for example, [13]). In some applications, collisions may be difficult to distinguish from the noise that is normally present on the channel, justifying the need for protocols that do not depend on the reliability of the collision detection mechanism (see [9,10]). Some network configurations are subject to frequent changes. In other networks, topologies could be unstable or dynamic; for example, when mobile users are present. In such situations, algorithms that do not assume any specific topology are more desirable.

More formally a radio network is a directed graph where by n we denote the number of nodes in this graph. If there is an edge from u to v, then we say that v is an *out-neighbor* of u and u is an *in-neighbor* of v. Each node is assigned a unique identifier from the set $\{1, 2, \ldots, n\}$. In the broadcast problem, one node, for example, node 1, is distinguished as the *source node*. Initially, the nodes do not possess any other information. In particular, they do not know the network topology.

The time is divided into discrete time steps. All nodes start simultaneously, have access to a common clock, and work synchronously. A broadcasting algorithm is a protocol that for each identifier *id*, given all past messages received by *id*, specifies, for each time step t, whether *id* will transmit a message at time t, and if so, it also specifies the message. A message M transmitted at time t from a node u is sent instantly to all its out-neighbors. An out-neighbor v of u receives M at time step t only if no collision occurred, that is, if the other in-neighbors of v do not transmit at time t at all. Further, collisions cannot be distinguished from background noise. If v does not receive any message at time t, it knows that either none of its in-neighbors transmitted at time t or that at least two did, but it does not know which of these two events occurred. The *running time* of a broadcasting algorithm is the smallest t such that for any network topology, and any assignment of identifiers to the nodes, all nodes receive the source message no later than at step t.

All efficient radio broadcasting algorithms are based on the following purely combinatorial concept of selectors.

Selectors Consider subsets of $\{1, \ldots, n\}$. We say that a set S *hits* a set X iff $|S \cap X| = 1$, and that S *avoids* Y iff $S \cap Y = \emptyset$. A family $\mathcal{S}$ of sets is a *w-selector* if it satisfies the following property:

(*) For any two disjoint sets X, Y with $w/2 \leq |X| \leq w$, $|Y| \leq w$, there is a set in $\mathcal{S}$ which hits X and avoids Y.

A complete layered network is a graph consisting of layers $L_0, \ldots, L_{m-1}$, in which each node in layer L_i is directly connected to every node in layer L_{i+1}, for all $i = 0, \ldots, m-1$. The layer L_0 contains only the source node s.

Key Results

Theorem 1 ([5]) *For all positive integers w and n, s.t., $w \leq n$, there exists a w-selector $\bar{S}$ with $O(w \log n)$ sets.*

Theorem 2 ([5]) *There exists a deterministic $O(n \log^2 n)$-time algorithm for broadcasting in radio networks with arbitrary topology.*

Theorem 3 ([5]) *There exists a deterministic $O(n \log n)$-time algorithm for broadcasting in complete layered radio networks.*

Applications

Prior to this work, Bruschi and Del Pinto showed in [1] that radio broadcasting requires time $\Omega(n \log D)$ in the worst case. In [4], Chlebus et al. presented a broadcasting algorithm with time complexity $O(n^{11/6})$ – the first subquadratic upper bound. This upper bound was later improved to $O(n^{5/3} \log^3 n)$ by De Marco and Pelc [8] and by Chlebus et al. [3] to $O(n^{3/2})$ by application of finite geometries.

Recently, Kowalski and Pelc in [12] proposed a faster $O(n \log n \log D)$ – time radio broadcasting algorithm, where D is the eccentricity of the network. Later, Czumaj and Rytter showed in [6] how to reduce this bound to $O(n \log^2 D)$. The results presented in [5] (see Theorems 1–3, as well as further improvements in [6, 12]) are existential (non-constructive). The proofs are based on the probabilistic method. A discussion on efficient explicit construction of selectors was initiated by Indyk in [11] and then continued by Chlebus and Kowalski in [2].

More careful analysis and further discussion on selectors in the context of *combinatorial group testing* can be found in [7], where DeBonis et al. proved that the size of selectors is $\Theta(w \log \frac{n}{w})$.

Open Problems

The exact complexity of radio broadcasting remains an open problem, although the gap between the lower and upper bounds $\Omega(n \log D)$ and $O(n \log^2 D)$ is now only a factor of $\log D$. Another promising direction for further studies is improvement of efficient explicit construction of selectors.

Recommended Reading

1. Bruschi D, Del Pinto M (1997) Lower bounds for the broadcast problem in mobile radio networks. Distrib Comput 10(3):129–135
2. Chlebus BS, Kowalski DR (2005) Almost optimal explicit selectors. In: Proceedings of 15th international symposium on fundamentals of computation theory, Lübeck, pp 270–280
3. Chlebus M, Gąsieniec L, Östlin A, Robson JM (2000) Deterministic broadcasting in radio networks. In: Proceedings of 27th international colloquium on automata, languages and programming, Geneva. LNCS, vol 1853, pp 717–728
4. Chlebus BS, Gąsieniec L, Gibbons AM, Pelc A, Rytter W (2002) Deterministic broadcasting in unknown radio networks. Distrib Comput 15(1): 27–38
5. Chrobak M, Gąsieniec L, Rytter W (2002) Fast broadcasting and gossiping in radio networks,. In: Proceedings of 41st annual symposium on foundations of computer science, Redondo Beach, 2000, pp 575–581; Full version in J Algorithms 43(2):177–189 (2002)
6. Czumaj A, Rytter W (2006) Broadcasting algorithms in radio networks with unknown topology. J Algorithms 60(2):115–143
7. De Bonis A, Gąsieniec L, Vaccaro U (2005) Optimal two-stage algorithms for group testing problems. SIAM J Comput 34(5):1253–1270
8. De Marco G, Pelc A (2001) Faster broadcasting in unknown radio networks. Inf Process Lett 79(2): 53–56
9. Ephremides A, Hajek B (1998) Information theory and communication networks: an unconsummated union. IEEE Trans Inf Theory 44: 2416–2434
10. Gallager R (1985) A perspective on multiaccess communications. IEEE Trans Inf Theory 31: 124–142
11. Indyk P (2002) Explicit constructions of selectors and related combinatorial structures, with applications. In: Proceedings of 13th annual ACM-SIAM symposium on discrete algorithms, San Francisco, pp 697–704
12. Kowalski DR, Pelc A (2005) Broadcasting in undirected ad hoc radio networks. Distrib Comput 18(1):43–57
13. Massey JL, Mathys P (1985) The collision channel without feedback. IEEE Trans Inf Theory 31:192–204

Deterministic Searching on the Line

Spyros Angelopoulos
Sorbonne Universités, L'Université Pierre et Marie Curie (UPMC), Université Paris 06, Paris, France

Keywords

Online Searching; Pure strategies; Searching for a point on the infinite line; Searching in one dimension

Years and Authors of Summarized Original Work

1993; Baeza-Yates, Culberson, Rawlins
2001; Jaillet, Stafford

Problem Definition

In the *Linear Search Problem* (LSP), we seek efficient strategies for locating an immobile target on the infinite line. More formally, the search environment consists of the infinite (i.e., unbounded) line, with a point O designated as a specific start point. A mobile searcher is initially located at O, whereas the target may be hidden at any point on the line. The searcher's strategy S defines the movement of the searcher on the line; on the other hand, the hider's strategy H is defined as the precise placement of the target on the line, and we denote by $|H|$ the distance of the target from the start point. Given strategies S, H, the *cost* of locating the target, denoted by $c(S, H)$ is the total distance traversed by the searcher at the first time the target is located. The *normalized cost* of the strategies is defined as the quantity $\overline{c}(S, H) = \frac{c(S,H)}{|H|}$.

The objective of the linear search problem is to determine a strategy S for the searcher that minimizes the worst-case normalized cost, namely, the quantity $\sup_H \overline{c}(S, H)$; the latter is often referred to as the *competitive ratio* of the strategy S, due to similarities of this setup with the competitive analysis of online algorithms. In game-theoretic terms, the problem can be described as a zero-sum game between the searcher and the hider in which we seek the minimax strategy of the game.

Extensions

A natural extension of the linear search problem is the *m-ray search* problem, also known as the *star search* problem. Here, the search environment consists of m infinite rays, with the start point O being their common intersection point. Clearly, the linear search problem is precisely the 2-ray search problem.

Constraints

It must be noted that if $|H|$ is arbitrarily small, no strategy of constant competitive ratio exists. Hence, a frequent and natural assumption in the field is to assume that $|H| \geq 1$, i.e., that the target is hidden at least at some minimum allowed distance from the start point. A different assumption that can be made in order to circumvent this complication is that the search strategy must incorporate an infinite sequence of infinitesimal steps (i.e., depths of exploration). In this entry we assume the former, namely, that $|H| \geq 1$.

In addition, we assume only deterministic (i.e., pure) strategies for both the searcher and the hider. We note that a substantial amount of previous work has addressed mixed strategies under given probability distributions on the placement of the target. We refer the reader to the textbook of Alpern and Gal [1].

Key Results

We consider two variants of the problem. In the first variant, the searcher lacks any information concerning the hidden target. In the second variant, the searcher knows that the target is within distance $h = |H|$ from the start point O.

Note that for the linear search problem, the searcher's strategy is completely determined by the sequence of search depths $\{x_i\}_{i \geq 1}$, where x_i denotes the total distance from the start point

in which the line is searched during the i-th exploration.

Searching with No Information

It has long been known that a *doubling* strategy, namely, the strategy $\{2^i\}_{i\geq 1}$ attains an optimal competitive ratio equal to 9 for this variant. The result is due to Beck and Newman [4] and rediscovered by Baeza-Yates et al. [2].

Calculating an upper bound on the competitive ratio is easy: if the target is at distance h from O, the doubling strategy will discover it at traversal $2k+1$, where $2^{2k-1} < h$ and $2^{2k+1} \geq h$. The total distance traversed by the searcher is equal to $2\sum_{i=1}^{2k} 2i + d = 4(2^{2k}-1)+h$. The competitive ratio is maximized when $h \to \infty$ and converges (from below) to 9.

An elegant approach for proving the tightness of this bound is based on lower bounds on certain functionals over positive sequences [8]. Let $\{x_i\}_{i\geq 1}$ be an optimal search strategy, then it is easy to see that its competitive ratio is at least equal to $\sup_k 1 + 2\frac{\sum_{i=1}^{k+1} x_i}{x_k}$. In addition, it can be readily seen that an optimal search strategy $\{x_i\}_{i\geq 1}$ must be *monotone*, i.e., $x_i \geq x_j$ for $i > j$. Given the above, Gal shows that there exists $a > 1$ such that

$$\sup_k 1 + 2\frac{\sum_{i=1}^{k+1} x_i}{x_k} \geq \sup_a \lim_{k\to\infty} 1 + 2\frac{\sum_{i=1}^{k+1} a^i}{a^k},$$

which in turn is at least equal to 9, for all $a > 1$.

Informally, the above argument shows that *geometric* strategies of the form $\{a^i\}_{i\geq 1}$ comprise the space of optimal strategies, and by choosing $a = 2$ one obtains the best strategy.

Searching with an Upper Bound on the Target Distance

In the setting in which the searcher has an upper bound h on the distance of the target from the start point, it is possible to obtain improved competitive ratios. Jaillet and Stafford [9] approach this problem by solving the following "dual" problem: given a target competitive ratio r, and the upper bound h, what is the largest "extent" (i.e., the furthest one can go in both directions) that can be searched while guaranteeing a competitive ratio at most r? A solution to this problem implies a solution to the (primal) search problem: it suffices to find the smallest r such that $e(r) \geq h$, where $e(h)$ is the best-possible extent.

The dual problem of finding $e(r)$ is addressed by means of a series of linear-program formulations. The solution to this series of linear programs defines a search strategy in which the search depths $\{x_i\}_{i\geq 1}$ are determined by an appropriate linear recurrence relation. As a last step, $e(r)$ is obtained as a particular element of the sequence that is generated by the linear recurrence in question. It should be noted that although the strategy is optimal, this technique does not yield a closed-form expression of the optimal competitive ratio (given h).

A similar approach leads to a solution for m-ray searching with an upper bound on the target distance. The crucial difficulty here, as opposed to the case $m = 2$, is in showing that an optimal strategy can be found in the class of *cyclic* strategies: these are strategies in which the searcher always visits the rays in some fixed round-robin order. This seemingly intuitive property is surprisingly hard to be established formally. To bypass this obstacle, one must first show that the property holds when the search depths form a nondecreasing sequence. Then one can argue that, once a searcher is at the start point, it will always choose to explore the ray that has been explored the least up to the current point. As noted in [9], this "least-extended-so-far discipline is the link between non-decreasing depths and the cyclic property that is sought." Once the optimality of cyclic strategies is established, a similar approach can be applied as in the case $m = 2$; namely, the search depths are determined by a (more complicated) linear recurrence.

Applications

The problem has obvious applications in the context of robotic navigation in an unknown environment. Strategies based on doubling are used in searching more complicated environments, e.g., a graph [11]. The linear search problem and its

generalization have connections with the design of black-box strategies for obtaining *interruptible* algorithms. The latter class consists of algorithms with the property that they return efficient solutions even if interrupted during their execution. Such algorithms are very desirable in the context of real-time and anytime applications in artificial intelligence [5].

Cross-References

▸ Randomized Searching on Rays or the Line

Recommended Reading

1. Alpern S, Gal S (2003) The theory of search games and rendezvous. Kluwer Academic, Boston
2. Baeza-Yates R, Culberson J, Rawlins G (1993) Searching in the plane. Inf Comput 106(2): 234–252
3. Beck A (1964) On the linear search problem. Naval Res Logist 2:221–228
4. Beck A, Newman DJ (1970) Yet more on the linear search problem. Isr J Math 8:419–429
5. Bernstein DS, Finkelstein L, Zilberstein S (2003) Contract algorithms and robots on rays: unifying two scheduling problems. In: Proceedings of the 18th international joint conference on artificial intelligence (IJCAI), Acapulco, pp 1211–1217
6. Bose P, De Carufel J, Durocher S (2013) Revisiting the problem of searching on a line. In: Proceedings of the 21st European symposium on algorithms (ESA), Sophia Antipolis, pp 205–216
7. Gal S (1972) A general search game. Isr J Math 12:34–45
8. Gal S (1974) Minimax solutions for linear search problems. SIAM J Appl Math 27:17–30
9. Jaillet P, Stafford M (2001) Online searching. Oper Res 49:501–515
10. Kirkpatrick DG (2009) Hyperbolic dovetailing. In: Proceedings of the 17th annual European symposium on algorithms (ESA), Copenhagen, pp 616–627
11. Koutsoupias E, Papadimitriou C, Yannakakis M (1996) Searching a fixed graph. In: Proceedings of the 23rd international colloquium on automata, languages, and programming (ICALP), Paderborn, pp 280–289
12. López-Ortiz A, Schuierer S (2001) The ultimate strategy to search on m rays. Theor Comput Sci 261(2):267–295
13. Schuierer S (2001) Lower bounds in online geometric searching. Comput Geom Theory Appl 18(1): 37–53

Dictionary Matching

Moshe Lewenstein
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Keywords

Approximate dictionary matching; Approximate text indexing

Years and Authors of Summarized Original Work

2004; Cole, Gottlieb, Lewenstein

Problem Definition

Indexing and *dictionary matching* are generalized models of pattern matching. These models have attained importance with the explosive growth of multimedia, digital libraries, and the Internet.

1. *Text Indexing:* In text indexing one desires to preprocess a text t, of length n, and to answer where subsequent queries p, of length m, appear in the text t.
2. *Dictionary Matching:* In dictionary matching one is given a dictionary D of strings $p_1, \ldots, p_d$ to be preprocessed. Subsequent queries provide a query string t, of length n, and ask for each location in t at which patterns of the dictionary appear.

Key Results

Text Indexing

The *indexing* problem assumes a large text that is to be preprocessed in a way that will allow the following efficient future queries. Given a query pattern, one wants to find all text locations that match the pattern in time proportional to the *pattern length and to the number of occurrences*.

To solve the indexing problem, Weiner [14] invented the *suffix tree* data structure (originally called a *position tree*), which can be constructed in linear time, and subsequent queries of length m are answered in time $O(m \log |\Sigma| + tocc)$, where $tocc$ is the number of pattern occurrences in the text.

Weiner's suffix tree in effect solved the indexing problem for exact matching of fixed texts. The construction was simplified by the algorithms of McCreight and, later, Chen and Seiferas. Ukkonen presented an online construction of the suffix tree. Farach presented a linear time construction for large alphabets (specifically, when the alphabet is $\{1, \ldots, n^c\}$, where n is the text size and c is some fixed constant). All results, besides the latter, work by handling one suffix at a time. The latter algorithm uses a divide-and-conquer approach, dividing the suffixes to be sorted to even-position suffixes and odd-position suffixes. See the entry on ▶ Suffix Tree Construction for full details. The standard query time for finding a pattern p in a suffix tree is $O(m \log |\Sigma|)$. By slightly adjusting the suffix tree, one can obtain a query time of $O(m + \log n)$; see [12].

Another popular data structure for indexing is suffix arrays. Suffix arrays were introduced by Manber and Myers. Others proposed linear time constructions for linearly bounded alphabets. All three extend the divide and conquer approach presented by Farach. The construction in [11] is especially elegant and significantly simplifies the divide-and-conquer approach, by dividing the suffix set into three groups instead of two. See the entry on ▶ Suffix Array Construction for full details. The query time for suffix arrays is $O(m + \log n)$ achievable by embedding additional lcp (longest common prefix) information into the data structure. See [11] for reference to other solutions. *Suffix Trays* were introduced in [6] as a merge between suffix trees and suffix arrays. The construction time of suffix trays is the same as for suffix trees and suffix arrays. The query time is $O(m + \log |\Sigma|)$.

Solutions for the indexing problem in *dynamic* texts, where insertions and deletions (of single characters or entire substrings) are allowed, appear in several Papers; see [2] and references therein.

Dictionary Matching

Dictionary matching is, in some sense, the "inverse" of text indexing. The large body to be preprocessed is a set of patterns, called the *dictionary*. The queries are texts whose length is typically significantly smaller than the dictionary size. It is desired to find all (exact) occurrences of dictionary patterns in the text in time proportional to the *text length and to the number of occurrences*.

Aho and Corasick [1] suggested an automaton-based algorithm that preprocesses the dictionary in time $O(d)$ and answers a query in time $O(n + docc)$, where $docc$ is the number of occurrences of patterns within the text. Another approach to solving this problem is to use a generalized suffix tree. A *generalized suffix tree* is a suffix tree for a collection of strings. Dictionary matching is done for the dictionary of patterns. Specifically, a suffix tree is created for the generalized string $p_1\$_1 p_2\$_2 \ldots \$_d p_d\$_d$, where the $\$_i$'s are not in the alphabet. A randomized solution using a fingerprint scheme was proposed in [3]. In [7] a parallel work-optimal algorithm for dictionary matching was presented. Ferragina and Luccio [8] considered the problem in the external memory model and suggested a solution based upon the String Btree data structure along with the notion of a certificate for dictionary matching. Two-dimensional dictionary matching is another fascinating topic which appears as a separate entry. See also the entry on ▶ Multidimensional String Matching.

Dynamic Dictionary Matching

Here one allows insertion and deletion of patterns from the dictionary D. The first solution to the problem was a suffix tree-based method for solving the dynamic dictionary matching problem. Idury and Schäffer [10] showed that the failure function (function mapping from one longest matching prefix to the next longest matching prefix; see [1]) approach and basic scanning loop of the Aho-Corasick algorithm can be adapted to dynamic dictionary matching for improved initial

dictionary preprocessing time. They also showed that faster search time can be achieved at the expense of slower dictionary update time.

A further improvement was later achieved by reducing the problem to maintaining a sequence of well-balanced parentheses under certain operations. In [13] an optimal method was achieved based on a labeling paradigm, where labels are given to, sometimes overlapping, substrings of different lengths. The running times are $O(|D|)$ preprocessing time, $O(m)$ update time, and $O(n + docc)$ time for search. See [13] for other references.

Text Indexing and Dictionary Matching with Errors

In most real-life systems, there is a need to allow errors. With the maturity of the solutions for *exact* indexing and *exact* dictionary matching, the quest for *approximate* solutions began. Two of the classical measures for approximating closeness of strings, Hamming distance and Edit distance, were the first natural measures to be considered.

Approximate Text Indexing

For approximate text indexing, given a distance k, one preprocesses a specified text t. The goal is to find all locations ℓ of t within distance k of the query p, i.e., for the Hamming distance all locations ℓ such that the length m substring of t beginning at that location can be made equal to p with at most k character substitutions. (An analogous statement applies for the edit distance.) For $k = 1$ [4] one can preprocess in time $O(n \log^2 n)$ and answer subsequent queries p in time $O(m\sqrt{\log n} \log\log n + occ)$. For small $k \geq 2$, the following naive solutions can be achieved. The first possible solution is to traverse a suffix tree checking all possible configurations of k, or less, mismatches in the pattern. However, while the preprocessing needed to build a suffix tree is cheap, the search is expensive, namely, $O(m^{k+1}|\Sigma|^k + occ)$. Another possible solution, for the Hamming distance measure only, leads to data structures of size approximately $O(n^{k+1})$ embedding all mismatch possibilities into the tree. This can be slightly improved by using the data structures for $k = 1$, which reduce the size to approximately $O(n^k)$.

Approximate Dictionary Matching

The goal is to preprocess the dictionary along with a threshold parameter k in order to support the following subsequent queries: Given a query text, seek all pairs of patterns (from the dictionary) and text locations which match within distance k. Here once again there are several algorithms for the case where $k = 1$ [4, 9]. The best solution for this problem has query time $O(m \log\log n + occ)$; the data structure uses space $O(n \log n)$ and can be built in time $O(n \log n)$.

The solutions for $k = 1$ in both problems (Approximate Text Indexing and Approximate Dictionary Matching) are based on the following, elegant idea, presented in Indexing terminology. Say a pattern p matches a text t at location i with one error at location j of p (and at location $i + j - 1$ of t). Obviously, the $j - 1$-length prefix of p matches the aligned substring of t and so does the $m - j - 1$ length suffix. If t and p are reversed, then the $j - 1$-th length prefix of p becomes a $j - 1$-th length suffix of p^R (that is p reverse). Notice that there is a match with, at most one error, if (1) the suffix of p starting at location $j + 1$ matches the (prefix of the) suffix of t starting at location $i + j$ and (2) the suffix of p^R starting at location $m - j + 1$ (the reverse of the $j - 1$-th length prefix of p) matches the (prefix of the) suffix of t^R starting at location $m - i - j + 3$. So, the problem now becomes a search for locations j which satisfy the above. To do so, the abovementioned solutions, naturally, use two suffix trees, one for the text and one for its reverse (with additional data structure tricks to answer the query fast). In dictionary matching the suffix trees are defined on the dictionary. The problem is that this solution does not carry over for $k \geq 2$. See the introduction of [5] for a full list of references.

Text Indexing and Dictionary Matching Within (Small) Distance *k*

Cole et al. [5] proposed a new method that yields a unified solution for approximate text indexing,

approximate dictionary matching, and other related problems. However, since the solution is somewhat involved, it will be simpler to explain the ideas on the following problem. The desire is to index a text t to allow fast searching for all occurrences of a pattern containing, at most, k don't cares (don't cares are special characters which match all characters).

Once again, there are two possible, relatively straightforward, solutions to be elaborated. The first is to use a suffix tree, which is cheap to preprocess, but causes the search to be expensive, namely, $O(m|\Sigma|^k + occ)$ (if considering k mismatches this would increase to $O(m^{k+1}|\Sigma|^k + occ)$. To be more specific, imagine traversing a path in a suffix tree. Consider the point where a don't care is reached. If in the middle of an edge the only text suffixes (representing substrings) that can match the pattern with this don't care must also go through this edge, so simply continue traversing. However, if at a node, then all the paths leaving this node must be explored. This explains the mentioned time bound.

The second solution is to create a tree that contains all strings that are at Hamming distance k from a suffix. This allows fast search but leads to trees of size exponential in k, namely, $O(n^{k+1})$ size trees. To elaborate, the tree, called a *k-error trie*, is constructed as follows. First, consider the case for one don't care, i.e., a 1-*error trie*, and then extend it. At any node v a don't care may need to be evaluated. Therefore, create a special subtree branching off this node that represents a don't care at this node. To understand this subtree, note that the subtree (of the suffix tree) rooted at v is actually a compressed trie of (some of the) suffixes of the text. Denote the collection of suffixes S_v. The first character of all these suffixes has to be removed (or, perhaps better imagined as a replacement with a don't care character). Each will be a new suffix of the text. Denote the new collection as S'_v. Now, create a new compressed trie of suffixes for S'_v, calling this new subtree an *error tree*. Do so for every v. The suffix tree along with its error trees is a 1-*error trie*. Turning to queries in the 1-*error trie*, when traversing the 1-*error trie*, do so with the suffix tree up till the don't care at node v. Move into the error tree at node v and continue the traversal of the pattern.

To create a 2-*error trie*, simply take each error tree and construct an error tree for each node within. A $(k+1)$-*error trie* is created recursively from a k-*error trie*. Clearly the 1-*error trie* is of size $O(n^2)$, since any node u in the original suffix tree will appear in all the new subtrees of the 1-*error trie* created for each of the nodes v which are ancestors of u. Likewise, the k-*error trie* is of size $O(n^{k+1})$.

The method introduced in Cole et al. [5] uses the idea of the error trees to form a new data structure, which is called a k-*errata trie*. The k-*errata trie* will be much smaller than $O(n^{k+1})$. However, it comes at the cost of a somewhat slower search time. To understand the k-*errata tries*, it is useful to first consider the 1-*errata tries* and to extend. The 1-*errata trie* is constructed as follows. The suffix tree is first decomposed with a centroid path decomposition (which is a decomposition of the nodes into paths, where all nodes along a path have their subtree sizes within a range 2^r and 2^{r+1}, for some integer r). Then, as before, error trees are created for each node v of the suffix tree with the following difference. Namely, consider the subtree, T_v, at node v and consider the edge (v, x) going from v to child x on the centroid path. T_v can be partitioned into two subtrees, $T_x \cup (v, x)$, and T'_v all the rest of T_v. An error tree is created for the suffixes in T'_v. The 1-*errata trie* is the suffix tree with all of its error trees. Likewise, a $(k+1)$-*errata trie* is created recursively from a k-*errata trie*. The contents of a k-*errata trie* should be viewed as a collection of error trees, k levels deep, where error trees at each level are constructed on the error trees of the previous level (at level 0 there is the original suffix tree). The following lemma helps in obtaining a bound on the size of the k-*errata trie*.

Lemma 1 *Let C be a centroid decomposition of a tree T. Let u be an arbitrary node of T and π be the path from the root to u. There are at most* $\log n$ *nodes v on π for which v and v's parent on π are on different centroid paths.*

The implication is that every node u in the original suffix tree will only appear in $\log n$ error trees of the 1-*errata trie* because each ancestor v of u is on the path π from the root to u and only $\log n$ such nodes are on different centroid paths than their children (on π). Hence, u appears in only $\log^k n$ error trees in the k-*errata trie*. Therefore, the size of the k-*errata trie* is $O(n \log^k n)$. Creating the k-*errata trie*s in $O(n \log^{k+1} n)$ can be done. To answer queries on a k-*errata trie*, given the pattern with (at most) k don't cares, the 0th level of the k-*errata trie*, i.e., the suffix tree, needs to be traversed. This is to be done until the first don't care, at location j, in the pattern is reached. If at node v in the 0th level of the k-*errata trie*, enter the (1st level) error tree hanging off of v and traverse this error tree from location $j + 2$ of the pattern (until the next don't care is met). However, the error tree hanging off of node v does not contain the subtree hanging off of v that is along the centroid path. Hence, continue traversing the pattern in the 0th level of the k-*errata trie*, starting along the edge on the centroid path leaving v (until the next don't care is met). The search is done recursively for k don't cares and, hence, yields an $O(2^k m)$ time search.

Recall that a solution for indexing text that supports queries of a pattern with k don't cares has been described. Unfortunately, when indexing to support k mismatch queries, not to mention k edit operation queries, the traversal down a k-*errata trie* can be very time consuming as frequent branching is required since an error may occur at any location of the pattern. To circumvent this problem, search many error trees in parallel. In order to do so, the error trees have to be grouped together. This needs to be done carefully; see [5] for the full details. Moreover, edit distance needs even more careful handling. The time and space of the algorithms achieved in [5] are as follows:

Approximate Text Indexing: The data structure for mismatches uses space $O(n \log^k n)$, takes time $O(n \log^{k+1} n)$ to build, and answers queries in time $O((\log^k n) \log \log n + m + occ)$. For edit distance, the query time becomes $O((\log^k n) \log \log n + m + 3^k \cdot occ)$. It must be pointed out that this result is mostly effective for constant k.

Approximate Dictionary Matching: For k mismatches the data structure uses space $O(n + d \log^k d)$, is built in time $O(n + d \log^{k+1} d)$, and has a query time of $O((m + \log^k d) \cdot \log \log n + occ)$. The bounds for edit distance are modified as in the indexing problem.

Applications

Approximate Indexing has a wide array of applications in signal processing, computational biology, and text retrieval, among others. Approximate Dictionary Matching is important in digital libraries and text retrieval systems.

Cross-References

- Indexed Approximate String Matching
- Indexed Two-Dimensional String Matching
- Multidimensional String Matching
- Multiple String Matching
- Suffix Tree Construction
- Suffix Trees and Arrays

Recommended Reading

1. Aho AV, Corasick MJ (1975) Efficient string matching. Commun ACM 18(6):333–340
2. Alstrup S, Brodal GS, Rauhe T (2000) Pattern matching in dynamic texts. In: Proceedings of the symposium on discrete algorithms (SODA), San Francisco, pp 819–828
3. Amir A, Farach M, Matias Y (1992) Efficient randomized dictionary matching algorithms. In: Proceedings of the symposium on combinatorial pattern matching (CPM), Tucson, pp 259–272
4. Amir A, Keselman D, Landau GM, Lewenstein N, Lewenstein M, Rodeh M (1999) Indexing and dictionary matching with one error. In: Proceedings of the workshop on algorithms and data structures (WADS), Vancouver, pp 181–192
5. Cole R, Gottlieb L, Lewenstein M (2004) Dictionary matching and indexing with errors and don't cares. In: Proceedings of the symposium on theory of computing (STOC), Chicago, pp 91–100

6. Cole R, Kopelowitz T, Lewenstein M (2006) Suffix trays and suffix trists: structures for faster text indexing. In: Proceedings of the international colloquium on automata, languages and programming (ICALP), Venice, pp 358–369
7. Farach M, Muthukrishnan S (1995) Optimal parallel dictionary matching and compression. In: Symposium on parallel algorithms and architecture (SPAA), Santa Barbara, pp 244–253
8. Ferragina P, Luccio F (1998) Dynamic dictionary matching in external memory. Inf Comput 146(2): 85–99
9. Ferragina P, Muthukrishnan S, de Berg M (1999) Multi-method dispatching: a geometric approach with applications to string matching. In: Proceedings of the symposium on the theory of computing (STOC), Atlanta, pp 483–491
10. Idury RM, Schäffer AA (1992) Dynamic dictionary matching with failure functions. In: Proceedings of the 3rd annual symposium on combinatorial pattern matching, Tucson, pp 273–284
11. Karkkainen J, Sanders P, Burkhardt S (2006) Linear work suffix array construction. J ACM 53(6):918–936
12. Mehlhorn K (1979) Dynamic binary search. SIAM J Comput 8(2):175–198
13. Sahinalp SC, Vishkin U (1996) Efficient approximate and dynamic matching of patterns using a labeling paradigm. In: Proceedings of the foundations of computer science (FOCS), Burlington, pp 320–328
14. Weiner P (1973) Linear pattern matching algorithm. In: Proceedings of the symposium on switching and automata theory, Iowa City, pp 1–11

Dictionary-Based Data Compression

Travis Gagie[1,2] and Giovanni Manzini[1,3]
[1]Department of Computer Science, University of Eastern Piedmont, Alessandria, Italy
[2]Department of Computer Science, University of Helsinki, Helsinki, Finland
[3]Department of Science and Technological Innovation, University of Piemonte Orientale, Alessandria, Italy

Keywords

LZ compression; Lempel compression; Parsing-based compression; Ziv

Years and Authors of Summarized Original Work

1977; Ziv, Lempel

Problem Definition

The problem of lossless data compression is the problem of compactly representing data in a format that admits the faithful recovery of the original information. Lossless data compression is achieved by taking advantage of the redundancy which is often present in the data generated by either humans or machines.

Dictionary-based data compression has been "the solution" to the problem of lossless data compression for nearly 15 years. This technique originated in two theoretical papers of Ziv and Lempel [15, 16] and gained popularity in the "1980s" with the introduction of the Unix tool `compress` (1986) and of the gif image format (1987). Although today there are alternative solutions to the problem of lossless data compression (e.g., Burrows-Wheeler compression and Prediction by Partial Matching), dictionary-based compression is still widely used in everyday applications: consider for example the zip utility and its variants, the modem compression standards V.42bis and V.44, and the transparent compression of pdf documents. The main reason for the success of dictionary-based compression is its unique combination of compression power and compression/decompression speed. The reader should refer to [13] for a review of several dictionary-based compression algorithms and of their main features.

Key Results

Let T be a string drawn from an alphabet Σ. Dictionary-based compression algorithms work by parsing the input into a sequence of substrings (also called words) $T_1, T_2, \ldots, T_d$ and by encoding a compact representation of these substrings. The parsing is usually done incrementally and

on-line with the following iterative procedure. Assume the encoder has already parsed the substrings $T_1, T_2, \ldots, T_{i-1}$. To proceed, the encoder maintains a dictionary of potential candidates for the next word T_i and associates a unique codeword with each of them. Then, it looks at the incoming data, selects one of the candidates, and emits the corresponding codeword. Different algorithms use different strategies for establishing which words are in the dictionary and for choosing the next word T_i. A larger dictionary implies a greater flexibility for the choice of the next word, but also longer codewords. Note that for efficiency reasons the dictionary is usually not built explicitly: the whole process is carried out implicitly using appropriate data structures.

Dictionary-based algorithms are usually classified into two families whose respective ancestors are two parsing strategies, both proposed by Ziv and Lempel and today universally known as LZ78 [16] and LZ77 [15].

The LZ78 Algorithm

Assume the encoder has already parsed the words $T_1, T_2, \ldots, T_{i-1}$, that is, $T = T_1 T_2 \cdots T_{i-1} \hat{T}_i$ for some text suffix $\hat{T}_i$. The LZ78 dictionary is defined as the set of strings obtained by adding a single character to one of the words $T_1, \ldots, T_{i-1}$ or to the empty word. The next word T_i is defined as the longest prefix of $\hat{T}_i$ which is a dictionary word. For example, for $T = aabbaaabaabaabba$ the LZ78 parsing is: a, ab, b, aa, aba, $abaa$, bb, a. It is easy to see that all words in the parsing are distinct, with the possible exception of the last one (in the example the word a). Let T_0 denote the empty word. If $T_i = T_j \alpha$, with $0 \le j < i$ and $\alpha \in \Sigma$, the codeword emitted by LZ78 for T_i will be the pair (j, α). Thus, if LZ78 parses the string T into t words, its output will be bounded by $t \log t + t \log |\Sigma| + \Theta(t)$ bits.

The LZ77 Algorithm

Assume the encoder has already parsed the words $T_1, T_2, \ldots, T_{i-1}$, that is, $T = T_1 T_2 \cdots T_{i-1} \hat{T}_i$ for some text suffix $\hat{T}_i$. The LZ77 dictionary is defined as the set of strings of the form $w\alpha$ where $\alpha \in \Sigma$ and w is a substring of T starting in the already parsed portion of T. The next word T_i is defined as the longest prefix of $\hat{T}_i$ which is a dictionary word. For example, for $T = aabbaaabaabaabba$ the LZ77 parsing is: a, ab, ba, $aaba$, $abaabb$, a. Note that, in some sense, $T_5 = abaabb$ is defined in terms of itself: it is a copy of the dictionary word $w\alpha$ with w starting at the second a of T_4 and extending into T_5! It is easy to see that all words in the parsing are distinct, with the possible exception of the last one (in the example the word a), and that the number of words in the LZ77 parsing is smaller than in the LZ78 parsing. If $T_i = w\alpha$ with $\alpha \in \Sigma$, the codeword for T_i is the triplet (s_i, ℓ_i, α) where s_i is the distance from the start of T_i to the last occurrence of w in $T_1 T_2 \cdots T_{i-1}$, and $\ell_i = |w|$.

Entropy Bounds

The performance of dictionary-based compressors has been extensively investigated since their introduction. In [15] it is shown that LZ77 is optimal for a certain family of sources, and in [16] it is shown that LZ78 achieves asymptotically the best compression ratio attainable by a finite-state compressor. This implies that, when the input string is generated by an ergodic source, the compression ratio achieved by LZ78 approaches the entropy of the source. More recent work has established similar results for other Ziv–Lempel compressors and has investigated the rate of convergence of the compression ratio to the entropy of the source (see [14] and references therein).

It is possible to prove compression bounds without probabilistic assumptions on the input, using the notion of *empirical entropy*. For any string T, the order k empirical entropy $H_k(T)$ is the maximum compression one can achieve using a uniquely decodable code in which the codeword for each character may depend on the k characters immediately preceding it [6]. The following lemma is a useful tool for establishing upper bounds on the compression ratio of dictionary-based algorithms which hold pointwise on every string T.

Lemma 1 ([6, Lemma 2.3]) *Let $T = T_1 T_2 \cdots T_d$ be a parsing of T such that each word T_i appears at most M times. Then, for any $k \geq 0$*

$$d \log d \leq |T| H_k(T) + d \log(|T|/d) + d \log M + \Theta(kd + d),$$

where $H_k(T)$ whereis the k-th order empirical entropy of T. □

Consider, for example, the algorithm LZ78. It parses the input T into t distinct words (ignoring the last word in the parsing) and produces an output bounded by $t \log t + t \log |\Sigma| + \Theta(t)$ bits. Using Lemma 1 and the fact that $t = O(|T|/\log T)$, one can prove that LZ78′s output is at most $|T| H_k(T) + o(|T|)$ bits. Note that the bound holds for any $k \geq 0$: this means that LZ78 is essentially "as powerful" as any compressor that encodes the next character on the basis of a finite context.

Algorithmic Issues

One of the reasons for the popularity of dictionary-based compressors is that they admit linear-time, space-efficient implementations. These implementations sometimes require non-trivial data structures: the reader is referred to [12] and references therein for further reading on this topic.

Greedy vs. Non-Greedy Parsing

Both LZ78 and LZ77 use a greedy parsing strategy in the sense that, at each step, they select the longest prefix of the unparsed portion which is in the dictionary. It is easy to see that for LZ77 the greedy strategy yields an optimal parsing; that is, a parsing with the minimum number of words. Conversely, greedy parsing is not optimal for LZ78: for any sufficiently large integer m there exists a string that can be parsed to $O(m)$ words and that the greedy strategy parses in $\Omega(m^{3/2})$ words. In [9] the authors describe an efficient algorithm for computing an optimal parsing for the LZ78 dictionary and, indeed, for any dictionary with the prefix-completeness property (a dictionary is prefix-complete if any prefix of a dictionary word is also in the dictionary). Interestingly, the algorithm in [9] is a one-step lookahead greedy algorithm: rather than choosing the longest possible prefix of the unparsed portion of the text, it chooses the prefix that results in the longest advancement in the *next* iteration.

Applications

The natural application field of dictionary-based compressors is lossless data compression (see, for example [13]). However, because of their deep mathematical properties, the Ziv–Lempel parsing rules have also found applications in other algorithmic domains.

Prefetching

Krishnan and Vitter [7] considered the problem of prefetching pages from disk into memory to anticipate users' requests. They combined LZ78 with a pre-existing prefetcher P_1 that is asymptotically at least as good as the best memoryless prefetcher, to obtain a new algorithm P that is asymptotically at least as good as the best finite-state prefetcher. LZ78′s dictionary can be viewed as a trie: parsing a string means starting at the root, descending one level for each character in the parsed string and, finally, adding a new leaf. Algorithm P runs LZ78 on the string of page requests as it receives them, and keeps a copy of the simple prefetcher P_1 for each node in the trie; at each step, P prefetches the page requested by the copy of P_1 associated with the node LZ78 is currently visiting.

String Alignment

Crochemore, Landau and Ziv-Ukelson [4] applied LZ78 to the problem of sequence alignment, i.e., finding the cheapest sequence of character insertions, deletions and substitutions that transforms one string T into another T' (the cost of an operation may depend on the character or characters involved). Assume, for simplicity, that $|T| = |T'| = n$. In 1980 Masek and Paterson proposed an $O(n^2/\log n)$-time algorithm with the restriction that the costs be rational; Crochemore et al.'s algorithm allows real-valued costs, has the same asymptotic cost

in the worst case, and is asymptotically faster for compressible texts.

The idea behind both algorithms is to break into blocks the matrix $A[1 \dots n, 1 \dots n]$ used by the obvious $O(n^2)$-time dynamic programming algorithm. Masek and Paterson break it into uniform-sized blocks, whereas Crochemore et al. break it according to the LZ78 parsing of T and T'. The rationale is that, by the nature of LZ78 parsing, whenever they come to solve a block $A[i \dots i', j \dots j']$, they can solve it in $O(i' - i + j' - j)$ time because they have already solved blocks identical to $A[i \dots i' - 1, j \dots j']$ and $A[i \dots i', j \dots j' - 1]$ [8]. Lifshits, Mozes, Weimann and Ziv-Ukelson [8] recently used a similar approach to speed up the decoding and training of hidden Markov models.

Compressed Full-Text Indexing

Given a text T, the problem of compressed full-text indexing is defined as the task of building an index for T that takes space proportional to the entropy of T and that supports the efficient retrieval of the occurrences of any pattern P in T. In [10] Navarro proposed a compressed full-text index based on the LZ78 dictionary. The basic idea is to keep two copies of the dictionary as tries: one storing the dictionary words, the other storing their reversal. The rationale behind this scheme is the following. Since any non-empty prefix of a dictionary word is also in the dictionary, if the sought pattern P occurs within a dictionary word, then P is a suffix of some word and easy to find in the second dictionary. If P overlaps two words, then some prefix of P is a suffix of the first word–and easy to find in the second dictionary–and the remainder of P is a prefix of the second word–and easy to find in the first dictionary. The case when P overlaps three or more words is a generalization of the case with two words. Recently, Arroyuelo et al. [1] improved the original data structure in [10]. For any text T, the improved index uses $(2 + \epsilon)|T|H_k(T) + o(|T| \log |\Sigma|)$ bits of space, where $H_k(T)$ is the k-th order empirical entropy of T, and reports all occ occurrences of P in T in $O(|P|^2 \log |P| + (|P| + occ) \log |T|)$ time.

Independently of [10], in [5] the LZ78 parsing was used together with the Burrows-Wheeler compression algorithm to design the first full-text index that uses $o(|T| \log |T|)$ bits of space and reports the occ occurrences of P in T in $O(|P| + occ)$ time. If $T = T_1 T_2 \cdots T_d$ is the LZ78 parsing of T, in [5] the authors consider the string $T_\$ = T_1 \$ T_2 \$ \cdots \$ T_d \$$ where \$ is a new character not belonging to Σ. The string $T_\$$ is then compressed using the Burrows-Wheeler transform. The \$'s play the role of anchor points: their positions in $T_\$$ are stored explicitly so that, to determine the position in T of any occurrence of P, it suffices to determine the position with respect to any of the \$'s. The properties of the LZ78 parsing ensure that the overhead of introducing the \$'s is small, but at the same time the way they are distributed within $T_\$$ guarantees the efficient location of the pattern occurrences.

Related to the problem of compressed full-text indexing is the compressed matching problem in which text and pattern are given together (so the former cannot be preprocessed). Here the task consists in performing string matching in a compressed text without decompressing it. For dictionary-based compressors this problem was first raised in 1994 by A. Amir, G. Benson, and M. Farach, and has received considerable attention since then. The reader is referred to [11] for a recent review of the many theoretical and practical results obtained on this topic.

Substring Compression Problems

Substring compression problems involve preprocessing T to be able to efficiently answer queries about compressing substrings: e.g., how compressible is a given substring s in T? what is s's compressed representation? or, what is the least compressible substring of a given length ℓ? These are important problems in bioinformatics because the compressibility of a DNA sequence may give hints as to its function, and because some clustering algorithms use compressibility to measure similarity. The solutions to these problems are often trivial for simple compressors, such as Huffman coding or run-length encoding, but they are open for more powerful algorithms, such as dictionary-based compressors, BWT

compressors, and PPM compressors. Recently, Cormode and Muthukrishnan [3] gave some preliminary solutions for LZ77. For any string s, let $C(s)$ denote the number of words in the LZ77-parsing of s, and let LZ77(s) denote the LZ77-compressed representation of s. In [3] the authors show that, with $O(|T|\ \text{polylog}(|T|))$ time preprocessing, for any substring s of T they can: a) compute LZ77(s) in $O(C(s)\log|T|\log\log|T|)$ time, b) compute an approximation of $C(s)$ within a factor $O(\log|T|\log^*|T|)$ in $O(1)$ time, c) find a substring of length ℓ that is close to being the least compressible in $O(|T|\ell/\log\ell)$ time. These bounds also apply to general versions of these problems, in which queries specify another substring t in T as context and ask about compressing substrings when LZ77 starts with a dictionary already containing the words in the LZ77 parsing of t.

Grammar Generation

Charikar et al. [2] considered LZ78 as an approximation algorithm for the NP-hard problem of finding the smallest context-free grammar that generates only the string T. The LZ78 parsing of T can be viewed as a context-free grammar in which for each dictionary word $T_i = T_j\alpha$ there is a production $X_i \to X_j\alpha$. For example, for $T = aabbaaabaabaabba$ the LZ78 parsing is: a, ab, b, aa, aba, $abaa$, bb, a, and the corresponding grammar is: $S \to X_1 \ldots X_7X_1, X_1 \to a, X_2 \to X_1b, X_3 \to b, X_4 \to X_1a, X_5 \to X_2a, X_6 \to X_5a, X_7 \to X_3b$. Charikar et al. showed LZ78's approximation ratio is in $O((|T|/\log|T|)^{2/3}) \cap \Omega(|T|^{2/3}\log|T|)$; i.e., the grammar it produces has size at most $f(|T|)\cdot m^*$, where $f(|T|)$ is a function in this intersection and m^* is the size of the smallest grammar. They also showed m^* is at least the number of words output by LZ77 on T, and used LZ77 as the basis of a new algorithm with approximation ratio $O(\log(|T|/m^*))$.

URL to Code

The source code of the gzip tool (based on LZ77) is available at the page http://www.gzip.org/. An LZ77-based compression library zlib is available from http://www.zlib.net/. A more recent, and more efficient, dictionary-based compressor is LZMA (Lempel–Ziv Markov chain Algorithm), whose source code is available from http://www.7-zip.org/sdk.html.

Cross-References

- Arithmetic Coding for Data Compression
- Boosting Textual Compression
- Burrows-Wheeler Transform
- Compressed Text Indexing

Recommended Reading

1. Arroyuelo D, Navarro G, Sadakane K (2006) Reducing the space requirement of LZ-index. In: Proceedings of 17th combinatorial pattern matching conference (CPM). LNCS, vol 4009. Springer, pp 318–329
2. Charikar M, Lehman E, Liu D, Panigraphy R, Prabhakaran M, Sahai A, Shelat A (2005) The smallest grammar problem. IEEE Trans Inf Theory 51:2554–2576
3. Cormode G, Muthukrishnan S (2005) Substring compression problems. In: Proceedings of. 16th ACM-SIAM symposium on discrete algorithms (SODA '05), pp 321–330
4. Crochemore M, Landau G, Ziv-Ukelson M (2003) A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM J Comput 32:1654–1673
5. Ferragina P, Manzini G (2005) Indexing compressed text. J ACM 52:552–581
6. Kosaraju R, Manzini G (1999) Compression of low entropy strings with Lempel–Ziv algorithms. SIAM J Comput 29:893–911
7. Krishnan P, Vitter J (1998) Optimal prediction for prefetching in the worst case. SIAM J Comput 27:1617–1636
8. Lifshits Y, Mozes S, Weimann O, Ziv-Ukelson M (2007) Speeding up HMMdecoding and training by exploiting sequence repetitions. Springer, 2007
9. Matias Y, Sahinalp C (1999) On the optimality of parsing in dynamic dictionary based data compression. In: Proceedings 10th annual ACM-SIAM symposium on discrete algorithms (SODA'99), pp 943–944
10. Navarro G (2004) Indexing text using the Ziv–Lempel trie. J Discret Algorithm 2:87–114
11. Navarro G, Tarhio J (2005) LZgrep: a Boyer-Moore string matching tool for Ziv–Lempel compressed text. Softw Pract Exp 35:1107–1130
12. Sahinalp C, Rajpoot N (2003) Dictionary-based data compression: an algorithmic perspective. In: Sayood

K (ed) Lossless compression handbook. Academic Press, pp 153–167
13. Salomon D (2007) Data compression: the complete reference, 4th edn. Springer, London
14. Savari S (1997) Redundancy of the Lempel–Ziv incremental parsing rule. IEEE Trans Inf Theory 43:9–21
15. Ziv J, Lempel A (1977) A universal algorithm for sequential data compression. IEEE Trans Inf Theory 23:337–343
16. Ziv J, Lempel A (1978) Compression of individual sequences via variable-length coding. IEEE Trans Inf Theory 24:530–536

Differentially Private Analysis of Graphs

Sofya Raskhodnikova and Adam Smith
Computer Science and Engineering Department, Pennsylvania State University, University Park, State College, PA, USA

Keywords

Degree distribution; Graphs; Privacy; Subgraph counts

Years and Authors of Summarized Original Work

2013; Blum, Blocki, Datta, Sheffet
2013; Chen, Zhou
2013; Kasiviswanatan, Nissim, Raskhodnikova, Smith
2015; Borgs, Chayes, Smith
2015; Raskhodnikova, Smith

Problem Definition

Many datasets can be represented by graphs, where nodes correspond to individuals and edges capture relationships between them. On one hand, such datasets contain potentially sensitive information about individuals; on the other hand, there are significant public benefits from allowing access to *aggregate* information about the data. Thus, analysts working with such graphs are faced with two conflicting goals: protecting privacy of individuals and publishing accurate aggregate statistics. This article describes algorithms for releasing accurate graph statistics while preserving a rigorous notion of privacy, called *differential privacy*.

Differential privacy was introduced by Dwork et al. [6]. It puts a restriction on the algorithm that processes sensitive data and publishes the output. Intuitively, differential privacy requires that, for every individual, the output distribution of the algorithm is roughly the same whether or not this individual's data is present in the dataset. Next, we give a formal definition of differential privacy, specialized to datasets represented by graphs.

Two graphs are called *neighbors* if one can be obtained from the other by removing a node and its adjacent edges. Given a parameter $\epsilon > 0$, an algorithm A is *ϵ-node differentially private* if for all neighbor graphs G and G' and for all sets S of possible outputs produced by A:

$$\Pr[A(G) \subseteq S] \leq e^{\epsilon} \cdot \Pr[A(G) \subseteq S].$$

This variant of differential privacy is called *node-differential privacy* because neighbor graphs are defined with respect to node removals. Analogously, we can define *edge differential privacy* by letting graphs be neighbors if they differ in exactly one edge. Intuitively, edge differential privacy protects edges (which represent connections between people), while node-differential privacy protects nodes together with their adjacent edges (i.e., all information pertaining to individuals). Node-differential privacy is a stronger privacy definition, but it is much harder to attain because it requires the output distribution of the algorithm to hide much larger differences in the input graph.

We would like to design differentially private algorithms (preferably, node-differentially private) that compute accurate graph statistics on a large family of realistic graphs. Typically, graphs that contain sensitive information, such as friendships, sexual relationships, and communication patterns, are sparse. Some examples of graph statistics we would like to compute

on these graphs are the number of edges, small subgraph counts, and the degree distribution.

Most work on the topic considers an analyst who wants to evaluate a real-valued function f on the private input graph G (e.g., the number of triangles or the number of connected components in G). The goal is to release as good an approximation as possible to the true value $f(G)$. Differentially private algorithms must be randomized, so we try to minimize the expectation of the random variable $\text{error}_A(G) = |A(G) - f(G)|$. We will also discuss work on algorithms that release higher-dimensional summaries (i.e., output a real vector).

Bibliographical Notes

Edge privacy was first studied by Nissim et al. [16], and the distinction between node and edge privacy was laid out by Hay et al. [9]. Edge differentially private algorithms for a variety of tasks have been widely investigated. Examples include subgraph counts, degree distributions, and parameters of generative statistical models. Gehrke et al. [7] investigated a notion whose strength lies between edge and node privacy: node privacy for bounded-degree graphs. (The focus of their work is a generalization of differential privacy, called *zero-knowledge privacy*.)

Until recently, no node-differentially private algorithms (where privacy guarantees hold with respect to all graphs) were known that compute accurate graph statistics on realistic (namely, sparse) graphs. The first such algorithms were designed independently by Blocki et al. [3], Kasiviswanathan et al. [11], and Chen and Zhou [5]. Those algorithms look at releasing one real-valued statistic at a time. Two more recent works focus on higher-dimensional node-private releases: Raskhodnikova and Smith [17] and Borgs et al. [4].

This encyclopedia entry focuses on node-differentially private algorithms, since these offer the strongest privacy guarantees. Progress, however, continues on edge-private algorithms; see Lin and Kifer [13], Karwa and Slavkovic [10], Lu and Miklau [14], and Zhang et al. [18] for recent results.

Key Results

The main difficulty in the design of node-private algorithms is that techniques based on *local sensitivity* of a function (which are the basis of the best edge-private algorithms) yield node-private algorithms whose error on "typical" inputs swamps the statistic that one wants to release. The local sensitivity of a function f is a discrete analogue of the derivative of f – it measures how much the value of f can change when the input graph is replaced with its neighbor. On sparse graphs, the local sensitivity can be larger than the value of the function. Any method whose error is proportional to the local sensitivity will have large relative error.

Focus on a "Preferred Subset"

To get around the challenge of high local sensitivity, two works [3, 11] independently designed algorithms that are given a set S of "nice" graphs that hopefully contains G (e.g., graphs with an upper bound on the maximum degree). These algorithms are private on *all* graphs and return an accurate answer *on graphs in* S. What makes this approach work is that S is selected so that the sensitivity of f is small when restricted to inputs in S.

Let $\mathbb{G}$ denote the set of all labeled, undirected graphs. We will call $S \subseteq \mathbb{G}$ the "preferred" subset. Define the *Lipschitz constant* (also called the *restricted sensitivity*) of f on S to be

$$\Delta_f(S) = \sup_{G,G' \in S} \frac{\|f(G') - f(G)\|_1}{d_{\text{node}}(G, G')},$$

where d_{node} is the node distance between two graphs – the number of vertex insertions and deletions needed to go from G to G'. Blocki et al. [3] and Kasiviswanathan et al. [11] give methods for adding noise proportional to the Lipschitz constant of f on S.

Theorem 1 ([3, 11]) *For every $S \subseteq \mathbb{G}$, function $f : S \to \mathbb{R}$, and $\epsilon > 0$, there exists an algorithm A_S that is ϵ-differentially private (for all inputs) and such that, for all $G \in S$,*

$$\mathbb{E}|A_S(G) - f(G)| = O(\Delta_S(f)/\epsilon^2).$$

Moreover, for $S = \mathbb{G}_{\mathbb{D}}$ (the set of D-bounded graphs), the running time of A is the running time for one evaluation of f plus a fixed polynomial in the size of G.

The same works [3, 11] also give generic reductions showing that given any algorithm that is ϵ-differentially private when restricted to graphs in S, one can design an algorithm A that has similar behavior on graphs in S but is ϵ'-differentially private for *all* inputs, for ϵ' not too much larger than ϵ.

"Down" Sensitivity

Rather than focusing on a single "nice" subset, some works [5, 17] sought to add noise proportional to a quantity related to, but usually much smaller than, the local sensitivity.

Define the *down sensitivity* (called *empirical global sensitivity* when first defined by Chen and Zhou [5]) of f at a graph G to be the Lipschitz constant of f *when restricted to the set of induced subgraphs of G*. Specifically, we write $G \preceq H$ to denote that G is an induced subgraph of H (i.e., G can be obtained by deleting a set of vertices from H) and define the down sensitivity to be

$$DS_f(G) = \max_{H, H' \text{neighbors}, H \preceq H' \preceq G} |f(G') - f(G)|.$$

By carefully (and privately) selecting the "preferred" subset based on the input, one can add noise essentially proportional to the down sensitivity.

Theorem 2 ([17]) *For every monotone function $f : \mathbb{G} \to \mathbb{R}$ and $\epsilon > 0$, there is an algorithm A_f that is ϵ-differentially private and such that, for all $G \in \mathbb{G}$,*

$$\mathbb{E}|A_f(G) - f(G)| = \frac{DS_f(G) + 1}{\epsilon} \cdot O(\log\log \max_{G'} DS_f(G')).$$

Moreover, A_f can be made efficient when f is a generalized linear query (a class that includes counting occurrences of a fixed subgraph).

The down sensitivity is low for many commonly studied statistics in graphs that satisfy α-*decay*, a condition on the degree distribution that is satisfied by known generative models (including those that generate "scale-free"). (See [11] for a definition of α-decay.)

Lipschitz Extensions and Higher-Dimensional Releases

The main technical tool in the down-sensitivity-based results [5, 17] is the construction of *efficient* (i.e., polynomial time computable) *Lipschitz extensions* of the function f from subsets S of graphs to the space of all graphs. Kasiviswanathan et al. [11] and Chen and Zhou [5] give efficient Lipschitz extensions of several useful functions (including graph counts) that return a single real value. Raskhodnikova and Smith [17] give efficient Lipschitz extensions of higher-dimensional functions, namely, the degree distribution and adjacency matrix of a graph.

Borgs et al. [4] use the Lipschitz extension technique together with the *exponential mechanism* to provide the first node-differentially private algorithms for fitting high-dimensional statistical models to a given graph (specifically, they consider *stochastic block models* and generalizations thereof).

Applications

The algorithms discussed above address a real problem: datasets containing sensitive information about relationships among a collection of individuals are often valuable sources of information, but publishing useful summaries about such data without leaking individual information is difficult. Even when the graphs are "anonymized" by removing all obviously identifying information, such as names, addresses, birthdays, and zip codes, they present a privacy risk. For example, [1, 15] give de-anonymization attacks based only on unlabeled links. Node-differentially private

algorithms offer a principled method for releasing information about a network while providing rigorous privacy guarantees (though some authors argue that even stronger notions may be needed [7, 12]).

Open Problems

Gupta et al. [8] and Blocki et al. [2] give edge differentially private algorithms for releasing a data structure that approximates the sizes of all cuts in the input graph in the following sense: for any cut, with high probability, the estimated cut size is accurate (the first reference gives weaker approximation guarantees with a stronger quantifier order: with high probability, all cut sizes are accurate). It is open whether a node-differentially private algorithm can obtain similar results.

For datasets that do not contain information about relationships, but only contain personal attributes that come from a relatively small set, differentially private algorithms can output a large number of statistics at once (see ▸ Query Release via Online Learning and ▸ Geometric Approaches to Answering Queries cross-referenced below). It is open how to do achieve similar results for graph statistics, even with edge differential privacy.

Finally, all algorithms we discussed release numerical graph statistics. The subject of differentially private synthetic graphs is largely unexplored. See [10, 13] for initial results.

Cross-References

- ▸ Beyond Worst Case Sensitivity in Private Data Analysis
- ▸ Geometric Approaches to Answering Queries
- ▸ Private Spectral Analysis
- ▸ Query Release via Online Learning

Acknowledgments The authors were supported in part by NSF award IIS-1447700, Boston University's Hariri Institute for Computing and Center for Reliable Information Systems and Cyber Security, and, while visiting the Harvard Center for Research on Computation & Society, by a Simons Investigator grant to Salil Vadhan.

Recommended Reading

1. Backstrom L, Dwork C, Kleinberg J (2007) Wherefore art thou r3579x? Anonymized social networks, hidden patterns, and structural steganography. In: Proceedings of the 16th international World Wide Web conference, Banff, pp 181–190
2. Blocki J, Blum A, Datta A, Sheffet O (2012) The Johnson-Lindenstrauss transform itself preserves differential privacy. In: 53rd annual IEEE symposium on foundations of computer science, FOCS 2012, New Brunswick, 20–23 Oct 2012. IEEE Computer Society, pp 410–419. doi:10.1109/FOCS.2012.67, http://dx.doi.org/10.1109/FOCS.2012.67
3. Blocki J, Blum A, Datta A, Sheffet O (2013) Differentially private data analysis of social networks via restricted sensitivity. In: Innovations in theoretical computer science (ITCS), Berkeley, pp 87–96
4. Borgs C, Chayes JT, Smith A (2015) Private graphon estimation for sparse graphs. arXiv:150606162 [mathST]
5. Chen S, Zhou S (2013) Recursive mechanism: towards node differential privacy and unrestricted joins. In: ACM SIGMOD international conference on management of data, New York, pp 653–664
6. Dwork C, McSherry F, Nissim K, Smith A (2006) Calibrating noise to sensitivity in private data analysis. In: Halevi S, Rabin T (eds) TCC, New York, vol 3876, pp 265–284
7. Gehrke J, Lui E, Pass R (2011) Towards privacy for social networks: a zero-knowledge based definition of privacy. In: Ishai Y (ed) TCC, Providence. Lecture notes in computer science, vol 6597. Springer, pp 432–449
8. Gupta A, Roth A, Ullman J (2012) Iterative constructions and private data release. In: TCC, Taormina
9. Hay M, Li C, Miklau G, Jensen D (2009) Accurate estimation of the degree distribution of private networks. In: International conference on data mining (ICDM), Miami, pp 169–178
10. Karwa V, Slavkovic A (2014) Inference using noisy degrees: differentially private ß-model and synthetic graphs. statME arXiv:1205.4697v3 [stat.ME]
11. Kasiviswanathan SP, Nissim K, Raskhodnikova S, Smith A (2013) Analyzing graphs with node-differential privacy. In: Theory of cryptography conference (TCC), Tokyo, pp 457–476
12. Kifer D, Machanavajjhala A (2011) No free lunch in data privacy. In: Sellis TK, Miller RJ, Kementsietsidis A, Velegrakis Y (eds) SIGMOD conference. ACM, Athens, Greece, pp 193–204
13. Lin BR, Kifer D (2013) Information preservation in statistical privacy and Bayesian estimation of unattributed histograms. In: ACM SIGMOD international conference on management of data, New York City, pp 677–688
14. Lu W, Miklau G (2014) Exponential random graph estimation under differential privacy. In: 20th ACM

SIGKDD international conference on knowledge discovery and data mining, New York City, pp 921–930
15. Narayanan A, Shmatikov V (2009) De-anonymizing social networks. In: IEEE symposium on security and privacy, Oakland, pp 173–187
16. Nissim K, Raskhodnikova S, Smith A (2007) Smooth sensitivity and sampling in private data analysis. In: Symposium on theory of computing (STOC), San Diego, pp 75–84, full paper on authors' web sites
17. Raskhodnikova S, Smith A (2015) High-dimensional Lipschitz extensions and node-private analysis of network data. arXiv:150407912
18. Zhang J, Cormode G, Procopiuc CM, Srivastava D, Xiao X (2015) Private release of graph statistics using ladder functions. In: ACM SIGMOD international conference on management of data, Melbourne, pp 731–745

Dilation of Geometric Networks

Rolf Klein
Institute for Computer Science, University of Bonn, Bonn, Germany

Keywords

Detour; Spanning ratio; Stretch factor

Years and Authors of Summarized Original Work

2005; Ebbers-Baumann, Grüne, Karpinski, Klein, Knauer, Lingas

Problem Definition

Notations

Let $G = (V, E)$ be a plane geometric network, whose vertex set V is a finite set of point sites in $\mathbb{R}^2$, connected by an edge set E of non-crossing straight line segments with endpoints in V. For two points $p \neq q \in V$, let $\xi_G(p, q)$ denote a shortest path from p to q in G. Then

$$\sigma(p, q) := \frac{|\xi_G(p, q)|}{|pq|} \qquad (1)$$

is the detour one encounters when using network G, in order to get from p to q, instead of walking straight. Here, $|\ .\ |$ denotes the Euclidean length.

The *dilation* of G is defined by

$$\sigma(G) := \max_{p \neq q \in V} \sigma(p, q). \qquad (2)$$

This value is also known as the spanning ratio or the stretch factor of G. It should, however, not be confused with the geometric dilation of a network, where the points on the edges are also being considered, in addition to the vertices.

Given a finite set S of points in the plane, one would like to find a plane geometric network $G = (V, E)$ whose dilation $\sigma(G)$ is as small as possible, such that S is contained in V. The value of

$$\Sigma(S) := \inf\{\sigma(G); G = (V, E) \text{ finite plane geometric network where } S \subset V\}$$

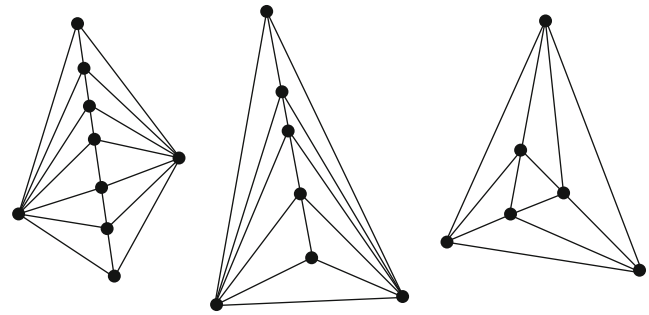
is called the *dilation of point set S*. The problem is in computing, or bounding, $\Sigma(S)$ for a given set S.

Related Work

If edge crossings were allowed, one could use spanners whose stretch can be made arbitrarily close to 1; see the monographs by Eppstein [6] or Narasimhan and Smid [12]. Different types of triangulations of S are known to have their stretch factors bounded from above by small constants, among them the Delaunay triangulation of stretch ≤ 2.42; see Dobkin et al. [3], Keil and Gutwin [10], and Das and Joseph [2]. Eppstein [5] has characterized all triangulations T of dilation $\sigma(T) = 1$; these triangulations are shown in Fig. 1. Trivially, $\Sigma(S) = 1$ holds for each point set S contained in the vertex set of such a triangulation T.

Key Results

The previous remark's converse also turns out to be true.



Dilation of Geometric Networks, Fig. 1 The triangulations of dilation 1

Theorem 1 ([11]) *If S is not contained in one of the vertex sets depicted in Fig. 1, then $\Sigma(S) > 1$.*

That is, if a point set S is not one of these special sets, then each plane network including S in its vertex set has a dilation larger than some lower bound $1 + \eta(S)$. The proof of Theorem 1 uses the following density result. Suppose one connects each pair of points of S with a straight line segment. Let S' be the union of S and the resulting crossing points. Now the same construction is applied to S' and repeated. For the limit point set S^{∞}, the following theorem holds. It generalizes work by Hillar and Rhea [8] and by Ismailescu and Radoičić [9] on the intersections of lines.

Theorem 2 ([11]) *If S is not contained in one of the vertex sets depicted in Fig. 1, then S^{∞} lies dense in some polygonal part of the plane.*

For certain infinite structures can concrete lower bounds be proven.

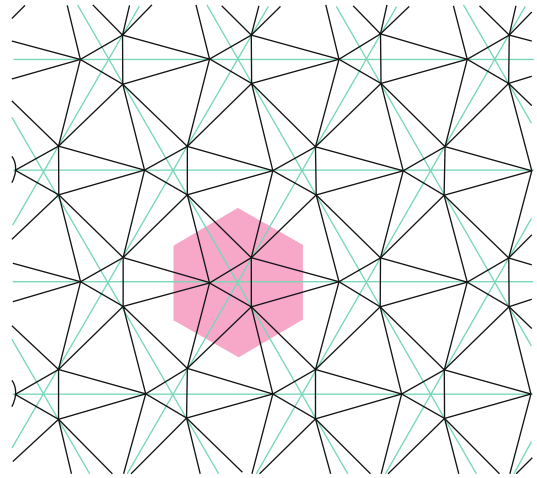
Theorem 3 ([4]) *Let N be an infinite plane network all of whose faces have a diameter bounded from above by some constant. Then $\sigma(N) > 1.00156$ holds.*

Theorem 4 ([4]) *Let C denote the (infinite) set of all points on a closed convex curve. Then $\Sigma(C) > 1.00157$ holds.*

Theorem 5 ([4]) *Given n families $F_i, 2 \leq i \leq n$, each consisting of infinitely many equidistant parallel lines. Suppose that these families are in general position.*

Then their intersection graph G is of dilation at least $2/\sqrt{3}$.

The proof of Theorem 5 makes use of Kronecker's theorem on simultaneous approximation. The bound is attained by the packing of equiangular triangles.



Dilation of Geometric Networks, Fig. 2 A network of dilation ∼1.1247

Finally, there is a general upper bound to the dilation of finite point sets.

Theorem 6 ([4]) *Each finite point set S is of dilation $\Sigma(S) < 1.1247$.*

To prove this upper bound, one can embed any given finite point set S in the vertex set of a scaled, and slightly deformed, finite part of the network depicted in Fig. 2. It results from a packing of equilateral triangles by replacing each vertex with a small triangle and by connecting neighboring triangles as indicated.

Applications

A typical university campus contains facilities like lecture halls, dorms, library, mensa, and

supermarkets, which are connected by some path system. Students in a hurry are tempted to walk straight across the lawn, if the shortcut seems worth it. After a while, this causes new paths to appear. Since their intersections are frequented by many people, they attract coffee shops or other new facilities. Now, people will walk across the lawn to get quickly to a coffee shop, and so on.

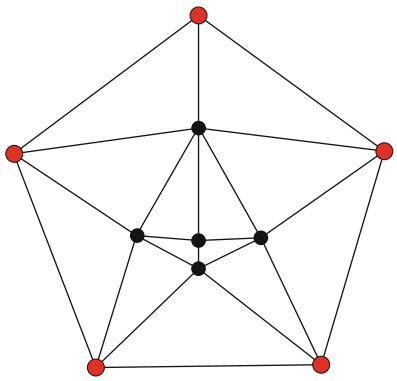
D. Eppstein [5] has asked what happens to the lawn if this process continues. The above results show that (1) part of the lawn will be completely destroyed, and (2) the temptation to walk across the lawn cannot, in general, be made arbitrarily small by a clever path design.

Open Problems

For practical applications, upper bounds to the weight (= total edge length) of a geometric network would be valuable, in addition to upper dilation bounds. Some theoretical questions require further investigation, too. Is $\Sigma(S)$ always attained by a finite network? How to compute, or approximate, $\Sigma(S)$ for a given finite set S? Even for a set as simple as S_5, the corners of a regular 5-gon, is the dilation unknown. The smallest dilation value known, for a triangulation containing S_5 among its vertices, equals 1.0204; see Fig. 3. Finally, what is the precise value of $\sup\{\Sigma(S); S \text{finite}\}$?

Cross-References

▸ Geometric Dilation of Geometric Networks



Dilation of Geometric Networks, Fig. 3 The best known embedding for S_5

Recommended Reading

1. Aronov B, de Berg M, Cheong O, Gudmundsson J, Haverkort H, Vigneron A (2005) Sparse geometric graphs with small dilation. In: Deng X, Du D (eds) Algorithms and computation: proceedings of the 16th international symposium (ISAAC 2005), Sanya. LNCS, vol 3827, pp 50–59. Springer, Berlin
2. Das G, Joseph D (1989) Which triangulations approximate the complete graph? In: Proceedings of the international symposium on optimal algorithms, Varna. LNCS, vol 401, pp 168–192. Springer, Berlin
3. Dobkin DP, Friedman SJ, Supowit KJ (1990) Delaunay graphs are almost as good as complete graphs. Discret Comput Geom 5:399–407
4. Ebbers-Baumann A, Gruene A, Karpinski M, Klein R, Knauer C, Lingas A (2007) Embedding point sets into plane graphs of small dilation. Int J Comput Geom Appl 17(3):201–230
5. Eppstein D, The geometry junkyard. http://www.ics.uci.edu/~eppstein/junkyard/dilation-free/
6. Eppstein D (1999) Spanning trees and spanners. In: Sack J-R, Urrutia J (eds) Handbook of computational geometry. Elsevier, Amsterdam, pp 425–461
7. Eppstein D, Wortman KA (2005) Minimum dilation stars. In: Proceedings of the 21st ACM symposium on computer geometry (SoCG), Pisa, pp 321–326
8. Hillar CJ, Rhea DL (2006) A result about the density of iterated line intersections. Comput Geom Theory Appl 33(3):106–114
9. Ismailescu D, Radoičić R (2004) A dense planar point set from iterated line intersections. Comput Geom Theory Appl 27(3):257–267
10. Keil JM, Gutwin CA (1992) The Delaunay triangulation closely approximates the complete Euclidean graph. Discret Comput Geom 7:13–28
11. Klein R, Kutz M, Penninger R (2015) Most finite point sets have dilation >1. Discret Comput Geom 53:80–106
12. Narasimhan G, Smid M (2007) Geometric spanner networks. Cambridge University Press, Cambridge/New York

Direct Routing Algorithms

Costas Busch
Department of Computer Science, Lousiana State University, Baton Rouge, LA, USA

Keywords

Bufferless packet switching; Collision-free packet scheduling; Hot-potato routing

Years and Authors of Summarized Original Work

2006; Busch, Magdon-Ismail, Mavronicolas, Spirakis

Problem Definition

The performance of a communication network is affected by the *packet collisions* which occur when two or more packets appear simultaneously in the same network node (router) and all these packets wish to follow the same outgoing link from the node. Since network links have limited available bandwidth, the collided packets wait on buffers until the collisions are resolved. Collisions cause delays in the packet delivery time and also contribute to the network performance degradation.

Direct routing is a packet delivery method which avoids packet collisions in the network. In direct routing, after a packet is injected into the network it follows a path to its destination without colliding with other packets, and thus without delays due to buffering, until the packet is absorbed at its destination node. The only delay that a packet experiences is at the source node while it waits to be injected into the network.

In order to formulate the direct routing problem, the network is modeled as a graph where all the network nodes are synchronized with a common time clock. Network links are bidirectional, and at each time step any link can be crossed by at most two packets, one packet in each direction. Given a set of packets, the *routing time* is defined to be the time duration between the first packet injection and the last packet absorbtion.

Consider a set of N packets, where each packet has its own source and destination node. In the *direct routing problem*, the goal is first to find a set of paths for the packets in the network, and second, to find appropriate injection times for the packets, so that if the packets are injected at the prescribed times and follow their paths they will be delivered to their destinations without collisions. The *direct scheduling problem* is a variation of the above problem, where the paths for the packets are given a priori, and the only task is to compute the injection times for the packets.

A *direct routing algorithm* solves the direct routing problem (similarly, a *direct scheduling algorithm* solves the direct scheduling problem). The objective of any direct algorithm is to minimize the routing time for the packets. Typically, direct algorithms are *offline*, that is, the paths and the injection schedule are computed ahead of time, before the packets are injected into the network, since the involved computation requires knowledge about all packets in order to guarantee the absence of collisions between them.

Key Results

Busch, Magdon-Ismail, Mavronicolas, and Spirakis, present in [6] a comprehensive study of direct algorithms. They study several aspects of direct routing such as the computational complexity of direct problems and also the design of efficient direct algorithms. The main results of their work are described below.

Hardness of Direct Routing

It is shown in [Sect. 4 in 6] that the optimal direct scheduling problem, where the paths are given and the objective is to compute an optimal injection schedule (that minimizes the routing time) is an NP-complete problem. This result is obtained with a reduction from vertex coloring, where vertex coloring problems are transformed to appropriate direct scheduling problems in a 2-dimensional grid. In addition, it is shown in [6] that approximations to the direct scheduling problem are as hard to obtain as approximations to vertex coloring. A natural question is what kinds of approximations can be obtained in polynomial time. This question is explored in [6] for general and specific kinds of graphs, as described below.

Direct Routing in General Graphs

A direct algorithm is given in [Section 3 in 6] that solves approximately the optimal direct scheduling problem in general network

topologies. Suppose that a set of packets and respective paths are given. The injection schedule is computed in polynomial time with respect to the size of the graph and the number of packets. The routing time is measured with respect to the *congestion* C of the packet paths (the maximum number of paths that use an edge), and the *dilation* D (the maximum length of any path).

The result in [6] establishes the existence of a simple greedy direct scheduling algorithm with routing time $rt = O(C \cdot D)$. In this algorithm, the packets are processed in an arbitrary order and each packet is assigned the smallest available injection time. The resulting routing time is worst-case optimal, since there exist instances of direct scheduling problems for which no direct scheduling algorithm can achieve a better routing time. A trivial lower bound on the routing time of any direct scheduling problem is $\Omega(C + D)$, since no algorithm can deliver the packets faster than the congestion or dilation of the paths. Thus, in the general case, the algorithm in [6] has routing time $rt = O((rt^*)^2)$, where rt^* is the optimal routing time.

Direct Routing in Specific Graphs

Several direct algorithms are presented in [6] for specialized network topologies. The algorithms solve the direct routing problem where first good paths are constructed and then an efficient injection schedule is computed. Given a set of packets, let C^* and D^* denote the optimal congestion and dilation, respectively, for all possible sets of paths for the packets. Clearly, the optimal routing time is $rt^* = \Omega(C^* + D^*)$. The upper bounds in the direct algorithm in [6] are expressed in terms of this lower bound. All the algorithms run in time polynomial to the size of the input.

Tree

The graph G is an arbitrary tree. A direct routing algorithm is given in [Section 3.1 in 6], where each packet follows the shortest path from its source to the destination. The injection schedule is obtained using the greedy algorithm with a particular ordering of the packets. The routing time of the algorithm is asymptotically optimal: $rt \leq 2C^* + D^* - 2 < 3 \cdot rt^*$.

Mesh

The graph G is a d-dimensional mesh (grid) with n nodes [10]. A direct routing algorithm is proposed in [Section 3.2 in 6], which first constructs efficient paths for the packets with congestion $C = O(d \log n \cdot C^*)$ and dilation $D = O(d^2 \cdot D^*)$ (the congestion is guaranteed with high probability). Then, using these paths the injection schedule is computed giving a direct algorithm with the routing time:

$$\begin{aligned} rt &= O(d^2 \log^2 n \cdot C^* + d^2 \cdot D^*) \\ &= O(d^2 \log^2 n \cdot rt^*). \end{aligned}$$

This result follows from a more general result which is shown in [6], that says that if the paths contain at most b "bends", i.e., at most b dimension changes, then there is a direct scheduling algorithm with routing time $O(b \cdot C + D)$. The result follows because the constructed paths have $b = O(d \log n)$ bends.

Butterfly

The graph G is a butterfly network with n input and n output nodes [10]. In [Section 3.3 in 6] the authors examine permutation routing problems in the butterfly, where each input (output) node is the source (destination) of exactly one packet. An efficient direct routing algorithm is presented in [6] which first computes good paths for the packets using Valiant's method [14, 15]: two butterflies are connected back to back, and each path is formed by choosing a random intermediate node in the output of the first butterfly. The chosen paths have congestion $C = O(\lg n)$ (with high probability) and dilation $D = 2 \lg n = O(D^*)$. Given the paths, there is a direct schedule with routing time very close to optimal: $rt \leq 5 \lg n = O(rt^*)$.

Hypercube

The graph G is a hypercube with n nodes [10]. A direct routing algorithm is given in [Section 3.4 in 6] for permutation routing problems. The algorithm first computes good paths for the packets by selecting a single random intermediate node for each packet. Then an appropriate injection

schedule gives routing time $rt < 14 \lg n$, which is worst-case optimal since there exist permutations for which $D^* = \Omega(\lg n)$.

Lower Bound for Buffering

In [Section 5 in 6] an additional problem has been studied about the amount of buffering required to provide small routing times. It is shown in [6] that there is a direct scheduling problem for which every direct algorithm requires routing time $\Omega(C \cdot D)$; at the same time, $C + D = \Theta(\sqrt{C \cdot D}) = o(C \cdot D)$. If buffering of packets is allowed, then it is well known that there exist packet scheduling algorithms [11, 12] with routing time very close to the optimal $O(C + D)$. In [6] it is shown that for the particular packet problem, in order to convert a direct injection schedule of routing time $O(C \cdot D)$ to a packet schedule with routing time $O(C + D)$, it is necessary to buffer packets in the network nodes in total $\Omega(N^{4/3})$ times, where a packet buffering corresponds to keeping a packet in an intermediate node buffer for a time step, and N is the number of packets.

Related Work

The only previous work which specifically addresses direct routing is for permutation problems on trees [3, 13]. In these papers, the resulting routing time is $O(n)$ for any tree with n nodes. This is worst-case optimal, while the result in [6] is asymptotically optimal for all routing problems in trees.

Cypher et al. [7] study an online version of direct routing in which a worm (packet of length L) can be re-transmitted if it is dropped (they also allow the links to have bandwidth $B \geq 1$). Adler et al. [1] study time constrained direct routing, where the task is to schedule as many packets as possible within a given time frame.They show that the time constrained version of the problem is NP-complete, and also study approximation algorithms on trees and meshes. Further, they discuss how much buffering could help in this setting.

Other models of bufferless routing are *matching routing* [2] where packets move to their destinations by swapping packets in adjacent nodes, and *hot-potato routing* [4, 5, 8, 9] in which packets follow links that bring them closer to the destination, and if they cannot move closer (due to collisions) they are deflected toward alternative directions.

Applications

Direct routing represent collision-free communication protocols, in which packets spend the smallest amount of time possible time in the network once they are injected. This type of routing is appealing in power or resource constrained environments, such as optical networks, where packet buffering is expensive, or sensor networks where energy resources are limited. Direct routing is also important for providing quality of service in networks. There exist applications where it is desirable to provide guarantees on the delivery time of the packets after they are injected into the network, for example in streaming audio and video. Direct routing is suitable for such applications.

Cross-References

- Oblivious Routing
- Packet Routing

Recommended Reading

1. Adler M, Khanna S, Rajaraman R, Rosén A (2003) Timeconstrained scheduling of weighted packets on trees and meshes. Algorithmica 36: 123–152
2. Alon N, Chung F, Graham R (1994) Routing permutations on graphs via matching. SIAM J Discret Math 7(3):513–530
3. Alstrup S, Holm J, de Lichtenberg K, Thorup M (1998) Direct routing on trees. In: Proceedings of the ninth annual ACM-SIAM, symposium on discrete algorithms (SODA 98), San Francisco, pp 342–349
4. Ben-Dor A, Halevi S, Schuster A (1998) Potential function analysis of greedy hot-potato routing. Theory Comput Syst 31(1):41–61

5. Busch C, Herlihy M, Wattenhofer R (2000) Hard-potato routing. In: Proceedings of the 32nd annual ACM symposium on theory of computing, Portland, pp 278–285
6. Busch C, Magdon-Ismail M, Mavronicolas M, Spirakis P (2006) Direct routing: algorithms and complexity. Algorithmica 45(1):45–68
7. Cypher R, Meyer auf der Heide F, Scheideler C, Vöcking, B (1996) Universal algorithms for store-and-forward and wormhole routing. In: Proceedings of the 28th ACM symposium on theory of computing, Philadelphia, pp 356–365
8. Feige U, Raghavan P (1992) Exact analysis of hot-potato routing. In: IEEE (ed) Proceedings of the 33rd annual, symposium on foundations of computer science, Pittsburgh, pp 553–562
9. Kaklamanis C, Krizanc D, Rao S (1993) Hot-potato routing on processor arrays. In: Proceedings of the 5th annual ACM, symposium on parallel algorithms and architectures, Velen, pp 273–282
10. Leighton FT (1992) Introduction to parallel algorithms and architectures: arrays – trees – hypercubes. Morgan Kaufmann, San Mateo
11. Leighton FT, Maggs BM, Rao SB (1994) Packet routing and jobscheduling in O(congestion+dilation) steps. Combinatorica 14:167–186
12. Leighton T, Maggs B, Richa AW (1999) Fast algorithms for finding O(congestion + dilation) packet routing schedules. Combinatorica 19:375–401
13. Symvonis A (1996) Routing on trees. Inf Process Lett 57(4):215–223
14. Valiant LG (1982) A scheme for fast parallel communication. SIAM J Comput 11:350–361
15. Valiant LG, Brebner GJ (1981) Universal schemes for parallel communication. In: Proceedings of the 13th annual ACM, symposium on theory of computing, Milwaukee, pp 263–277

Directed Perfect Phylogeny (Binary Characters)

Jesper Jansson
Laboratory of Mathematical Bioinformatics, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto, Japan

Keywords

Binary character; Character state matrix; Perfect phylogeny; Phylogenetic reconstruction; Phylogenetic tree

Years and Authors of Summarized Original Work

1991; Gusfield
1995; Agarwala, Fernández-Baca, Slutzki
2004; Pe'er, Pupko, Shamir, Sharan

Problem Definition

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of elements called *objects* and let $C = \{c_1, c_2, \ldots, c_m\}$ be a set of functions from S to $\{0, 1\}$ called *characters*. For each object $s_i \in S$ and character $c_j \in C$, we say that s_i *has* c_j if $c_j(s_i) = 1$ or that s_i *does not have* c_j if $c_j(s_i) = 0$, respectively (in this sense, characters are *binary*). Then the set S and its relation to C can be naturally represented by a matrix M of size $(n \times m)$ satisfying $M[i, j] = c_j(s_i)$ for every $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, m\}$. Such a matrix M is called a *binary character state matrix*.

Next, for each $s_i \in S$, define the set $C_{s_i} = \{c_j \in C : s_i \text{ has } c_j\}$. A *phylogeny for* S is a tree whose leaves are bijectively labeled by S, and a *directed perfect phylogeny for* (S, C) (if one exists) is a rooted phylogeny T for S in which each $c_j \in C$ is associated with exactly one edge of T in such a way that for any $s_i \in S$, the set of all characters associated with the edges on the path in T from the root to leaf s_i is equal to C_{s_i}. See Figs. 1 and 2 for two examples.

Now, define the following problem.

Problem 1 (The Directed Perfect Phylogeny Problem for Binary Characters)

INPUT: An $(n \times m)$-binary character state matrix M for some S and C.

OUTPUT: A directed perfect phylogeny for (S, C), if one exists; otherwise, *null*.

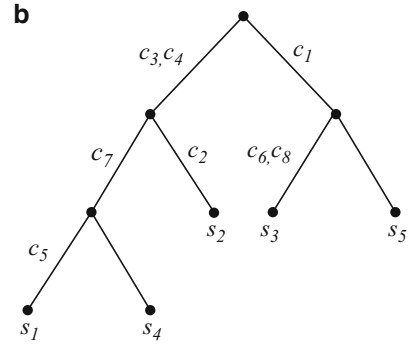
Key Results

In the presentation below, define a set S_{c_j} for each $c_j \in C$ by $S_{c_j} = \{s_i \in S : s_i \text{ has } c_j\}$. The next lemma is the key to solving the Directed

a

M	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
s_1	0	0	1	1	1	0	1	0
s_2	0	1	1	1	0	0	0	0
s_3	1	0	0	0	0	1	0	1
s_4	0	0	1	1	0	0	1	0
s_5	1	0	0	0	0	0	0	0

b



Directed Perfect Phylogeny (Binary Characters), Fig. 1 (**a**) A (5×8)-binary character state matrix M. (**b**) A directed perfect phylogeny for (S, C)

M	c_1	c_2
s_1	1	0
s_2	1	1
s_3	0	1

Directed Perfect Phylogeny (Binary Characters), Fig. 2 This binary character state matrix admits no directed perfect phylogeny

Perfect Phylogeny Problem for Binary Characters efficiently. It is also known in the literature as *the pairwise compatibility theorem* [5].

Lemma 1 *There exists a directed perfect phylogeny for (S, C) if and only if for each pair $c_j, c_k \in C$, it holds that $S_{c_j} \cap S_{c_k} = \emptyset$, $S_{c_j} \subseteq S_{c_k}$, or $S_{c_k} \subseteq S_{c_j}$.*

Short constructive proofs of the lemma can be found in, e.g., [8] and [14]. An algebraic proof of a slightly more general version of the lemma was given earlier by Estabrook, Johnson, and McMorris [3,4].

Using Lemma 1, it is trivial to construct a top-down algorithm for the problem that runs in $O(nm^2)$ time. As one might expect, a faster algorithm is possible. Gusfield [7] observed that after sorting the columns of M in nonincreasing lexicographic order, all duplicate copies of a column appear in a consecutive block of columns and column j is to the right of column k if S_{c_j} is a proper subset of S_{c_k}, and then exploited these two facts together with Lemma 1 to obtain the following result:

Theorem 1 ([7]) *The Directed Perfect Phylogeny Problem for Binary Characters can be solved in $O(nm)$ time.*

For a description of the original algorithm and a proof of its correctness, see [7] or [14]. A conceptually simplified version of the algorithm based on keyword trees can be found in Chapter 17.3.4 in [8]. Gusfield [7] also gave an adversary argument to prove a corresponding lower bound of $\Omega(nm)$ on the running time, showing that his algorithm is time optimal:

Theorem 2 ([7]) *Any algorithm that decides if a given binary character state matrix M admits a directed perfect phylogeny must, in the worst case, examine all entries of M.*

Agarwala, Fernández-Baca, and Slutzki [1] noted that the input binary character state matrix is often sparse, i.e., in general, most of the objects will not have most of the characters. In addition, they noted that for the sparse case, it is more efficient to represent the input (S, C) by all the sets S_{c_j} for $j \in \{1, 2, \ldots, m\}$, where each set S_{c_j} is defined as above and each S_{c_j} is specified as a linked list, than by using a binary character state matrix. Agarwala et al. [1] proved that with this alternative representation of S and C, the algorithm of Gusfield can be modified to run in time proportional to the total number of 1s in the corresponding binary character state matrix:

Theorem 3 ([1]) *The variant of the Directed Perfect Phylogeny Problem for Binary*

Characters in which the input is given as linked lists representing all the sets S_{c_j} for $j \in \{1, 2, \ldots, m\}$ can be solved in $O(h)$ time, where $h = \sum_{j=1}^{m} |S_{c_j}|$.

For a description of the algorithm, refer to [1] or [6]. Observe that Theorem 3 does not contradict Theorem 2; in fact, Gusfield's lower bound argument for proving Theorem 2 considers an input matrix consisting mostly of 1s.

When only a portion of an $(n \times m)$-binary character state matrix is available, an $\tilde{O}(nm)$-time algorithm by Pe'er et al. [13] can fill in the missing entries with 0s and 1s so that the resulting matrix admits a directed perfect phylogeny, if possible. A ZDD-based algorithm for enumerating all such solutions was recently developed by Kiyomi et al. [11].

Theorem 4 ([13]) *The variant of the Directed Perfect Phylogeny Problem for Binary Characters in which the input consists of an incomplete binary character state matrix can be solved in $\tilde{O}(nm)$ time.*

Applications

Directed perfect phylogenies for binary characters are used to describe the evolutionary history for a set of objects (e.g., biological species) that share some observable traits and that have evolved from a "blank" ancestral object which has none of the traits. Intuitively, the root of a directed perfect phylogeny corresponds to the blank ancestral object, and each directed edge $e = (u, v)$ corresponds to an evolutionary event in which the hypothesized ancestor represented by u gains the characters associated with e, transforming it into the hypothesized ancestor or object represented by v. For simplicity, it may be assumed that each character can emerge once only during the evolutionary history and is never lost after it has been gained, so that a leaf s_i is a descendant of the edge associated with a character c_j if and only if s_i has c_j. When this requirement is too strict, one can relax it to permit errors, for example, by letting each character be associated with more than one edge in the phylogeny (i.e., allow each character to emerge many times) while minimizing the total number of such associations (*Camin-Sokal optimization*) or by keeping the requirement that each character emerges only once but allowing it to be lost multiple times (*Dollo parsimony*) [5, 6]. Such relaxations generally increase the computational complexity of the underlying computational problems; see, e.g., [2] and [15].

Binary characters are commonly used by biologists and linguists. Traditionally, morphological traits or directly observable features of species were employed by biologists as binary characters, and recently, binary characters based on genomic information such as substrings in DNA or protein sequences, SNP markers, protein regulation data, and shared gaps in a given multiple alignment have become more and more prevalent. Chapter 17.3.2 in [8] mentions several examples where phylogenetic trees have been successfully constructed based on such types of binary character data. In the context of reconstructing the evolutionary history of natural languages, linguists often use phonological and morphological characters with just two states [10].

The Directed Perfect Phylogeny Problem for Binary Characters is closely related to *the Perfect Phylogeny Problem*, a fundamental problem in computational evolutionary biology and phylogenetic reconstruction [5, 6, 14]. This problem (also described in more detail in Encyclopedia entry ▸ Perfect Phylogeny (Bounded Number of States)) introduces nonbinary characters so that each character $c_j \in C$ has a set of allowed states $\{0, 1, \ldots, r_j - 1\}$ for some integer r_j, and for each $s_i \in S$, character c_j is in one of its allowed states. Generalizing the notation used above, define the set $S_{c_j,\alpha}$ for every $\alpha \in \{0, 1, \ldots, r_j - 1\}$ by $S_{c_j,\alpha} = \{s_i \in S : \text{the state of } s_i \text{ on } c_j \text{ is } \alpha\}$. Then, the objective of *the Perfect Phylogeny Problem* is to construct (if possible) an *unrooted* phylogeny T for S such that the following holds: for each $c_j \in C$ and distinct states α, β of c_j, the minimal subtree of T that connects $S_{c_j,\alpha}$ and the minimal subtree of T that connects $S_{c_j,\beta}$ are vertex-disjoint. McMorris [12] showed that the special case with $r_j = 2$ for all $c_j \in C$ can be reduced to the

Directed Perfect Phylogeny Problem for Binary Characters in $O(nm)$ time: for each $c_j \in C$, if the number of 1s in column j of M is greater than the number of 0s, then set entry $M[i, j]$ to $1 - M[i, j]$ for all $i \in \{1, 2, \ldots, n\}$. Therefore, another application of Gusfield's algorithm [7] is as a subroutine for solving the Perfect Phylogeny Problem in $O(nm)$ time when $r_j = 2$ for all $c_j \in C$. Even more generally, the Perfect Phylogeny Problem for directed as well as undirected *cladistic* characters can be solved in polynomial time by a similar reduction to the Directed Perfect Phylogeny Problem for Binary Characters (see [6]).

In addition to the above, it is possible to apply Gusfield's algorithm to determine whether two given trees describe compatible evolutionary history, and if so, merge them into a single tree so that no branching information is lost (see [7] for details). Finally, Gusfield's algorithm has also been used by Hanisch, Zimmer, and Lengauer [9] to implement a particular operation on documents defined in their Protein Markup Language (ProML) specification.

Cross-References

- ▸ Directed Perfect Phylogeny (Binary Characters)
- ▸ Perfect Phylogeny Haplotyping

Acknowledgments JJ was funded by the Hakubi Project at Kyoto University and KAKENHI grant number 26330014.

Recommended Reading

1. Agarwala R, Fernández-Baca D, Slutzki G (1995) Fast algorithms for inferring evolutionary trees. J Comput Biol 2(3):397–407
2. Bonizzoni P, Braghin C, Dondi R, Trucco G (2012) The binary perfect phylogeny with persistent characters. Theor Comput Sci 454:51–63
3. Estabrook GF, Johnson CS Jr, McMorris FR (1976) An algebraic analysis of cladistic characters. Discret Math 16(2):141–147
4. Estabrook GF, Johnson CS Jr, McMorris FR (1976) A mathematical foundation for the analysis of cladistic character compatibility. Math Biosci 29(1–2): 181–187
5. Felsenstein J (2004) Inferring phylogenies. Sinauer Associates, Sunderland
6. Fernández-Baca D (2001) The perfect phylogeny problem. In: Cheng X, Du DZ (eds) Steiner trees in industry. Kluwer Academic, Dordrecht, pp 203–234
7. Gusfield DM (1991) Efficient algorithms for inferring evolutionary trees. Networks 21:19–28
8. Gusfield DM (1997) Algorithms on strings, trees, and sequences. Cambridge University Press, New York
9. Hanisch D, Zimmer R, Lengauer T (2002) ProML – the protein markup language for specification of protein sequences, structures and families. In Silico Biol 2:0029. http://www.bioinfo.de/isb/2002/02/0029/
10. Kanj IA, Nakhleh L, Xia G (2006) Reconstructing evolution of natural languages: complexity and parameterized algorithms. In: Proceedings of the 12th annual international computing and combinatorics conference (COCOON 2006). Lecture notes in computer science, vol 4112. Springer, Berlin/Heidelberg, pp 299–308
11. Kiyomi M, Okamoto Y, Saitoh T (2012) Efficient enumeration of the directed binary perfect phylogenies from incomplete data. In: Proceedings of the 11th international symposium on experimental algorithms (SEA 2012). Lecture notes in computer science, vol 7276. Springer, Berlin/Heidelberg, pp 248–259
12. McMorris FR (1977) On the compatibility of binary qualitative taxonomic characters. Bull Math Biol 39(2):133–138
13. Pe'er I, Pupko T, Shamir R, Sharan R (2004) Incomplete directed perfect phylogeny. SIAM J Comput 33(3):590–607
14. Setubal JC, Meidanis J (1997) Introduction to computational molecular biology. PWS Publishing Company, Boston
15. Sridhar S, Dhamdhere K, Blelloch GE, Halperin E, Ravi R, Schwartz R (2007) Algorithms for efficient near-perfect phylogenetic tree reconstruction in theory and practice. IEEE/ACM Trans Comput Biol Bioinform 4(4):561–571

Discrete Ricci Flow for Geometric Routing

Jie Gao[1], Xianfeng David Gu[1], and Feng Luo[2]
[1]Department of Computer Science, Stony Brook University, Stony Brook, NY, USA
[2]Department of Mathematics, Rutgers University, Piscataway, NJ, USA

Keywords

Geometric routing; Greedy routing; Greedy embedding; Virtual coordinates; Wireless networks

Years and Authors of Summarized Original Work

2009; Sarkar, Yin, Gao, Luo, Gu
2010; Sarkar, Zeng, Gao, Gu
2010; Zeng, Sarkar, Luo, Gu, Gao
2011; Jiang, Ban, Goswami, Zeng, Gao, Gu
2011; Yu, Ban, Sarkar, Zeng, Gu, Gao
2012; Yu, Yin, Han, Gao, Gu
2013; Ban, Goswami, Zeng, Gu, Gao
2013; Li, Zeng, Zhou, Gu, Gao

Problem Definition

The problem is concerned about computing virtual coordinates for greedy routing in a wireless ad hoc network. Consider a set of wireless nodes S densely deployed inside a geometric domain $\mathcal{R} \subseteq \mathbb{R}^2$. Nodes within communication range can directly communicate with each other. We ask whether one can compute a set of virtual coordinates for S such that greedy routing has guaranteed delivery. In particular, each node forwards the message to the neighbor whose distance to the destination, computed under the virtual coordinates and some metric function d, is the smallest. If such a neighbor can always be found, greedy routing successfully delivers the message to the destination. The problem can be phrased as finding a *greedy embedding* of S in some geometric space, such that greedy routing always succeeds.

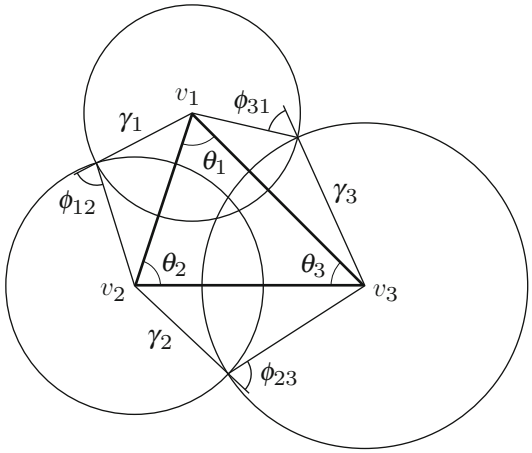
In the setting of this entry, we assume that the nodes are a dense sample of the domain $\mathcal{R}$ such that the communication graph on S contains a triangulated mesh Σ as a discrete approximation of $\mathcal{R}$.

Key Results

The key result is a family of distributed algorithms for computing the greedy embedding using discrete Ricci flow. Given a triangular mesh Σ with vertex set V, edge set E, and face set F, we can define a piecewise linear metric by the edge lengths on Σ: $l : E \to \mathbb{R}^+$ that satisfies the triangle inequality for each triangle face. The piecewise linear metric determines the corner angles of the triangles on Σ, by the cosine law. The *discrete curvature* K_i at a vertex v_i is defined as the angle deficit on the mesh. If v_i is an interior vertex, $K_i = 2\pi - \sum_j \theta_j$, where θ_j's are the corner angles at v_i. If v_i is a vertex on the boundary, $K_i = \pi - \sum_j \theta_j$, where θ_j's are the corner angles at v_i. Thus, the curvature at an interior vertex v_i is 0 if the surface is flat at v_i. The curvature at a boundary vertex v_i is 0 if the boundary is locally a straight line at v_i (see Fig. 1). The famous Gauss-Bonnet theorem states that the total curvature is a topological invariant: $\sum_{v_i \in V} K_i = 2\pi\chi(\Sigma)$, where $\chi(\Sigma)$ is the Euler characteristic number (The Euler characteristics number of a surface is $2 - 2g - h$, where g is the genus or the number of handles and h is the number of holes.) of Σ. Ricci flow is a process that deforms the surface metric to meet any target curvature that is admissible by the Gauss-Bonnet theorem.

A conformal map in the continuous surface preserves the intersection angle of any two curves. In the discrete case, the "intersection angle" is defined using the circle packing metric [10, 11]. We place a circle at each vertex v_i with radius γ_i such that for each edge e_{ij}, the circles at v_i, v_j intersect or are tangent to each other. The intersection angle is denoted by $\phi(e_{ij})$. The pair of vertex radii and the intersection angles on a mesh Σ, (Γ, Φ), are called a *circle packing metric* of Σ (see Fig. 1). Two circle packing metrics (Γ_1, Φ_1) and (Γ_2, Φ_2) on the same mesh are *conformal equivalent*, if $\Phi_1 \equiv \Phi_2$. Therefore, a conformal deformation of a circle packing metric only modifies the vertex radii γ_i's and preserves the intersection angles. Note that the circle packing metric and the edge lengths (the piecewise linear metric) on one mesh can be converted to each other by using the cosine law.

Now we are ready to introduce the discrete Ricci flow algorithm. Let u_i be $\log \gamma_i$ for each vertex. Then the discrete Ricci flow, introduced in the work of [2], is defined as follows: $\frac{du_i(t)}{dt} = \bar{K}_i - K_i$, where $K_i, \bar{K}_i$ are the current and target curvature at vertex v_i, respectively. Discrete Ricci



Discrete Ricci Flow for Geometric Routing, Fig. 1 The circle packing metric

flow can be formulated in the variational setting, namely, it is a negative gradient flow of some special energy form: $f(\mathbf{u}) = \int_{\mathbf{u_0}}^{\mathbf{u}} \sum_{i=1}^{n} (\bar{K}_i - K_i) d u_i$, where $\mathbf{u_0}$ is an arbitrary initial metric and $\bar{K}$ is the prescribed target curvature. The integration above is well defined and called the *Ricci energy*. The discrete Ricci flow is the negative gradient flow of the discrete Ricci energy. The discrete metric which induces $\bar{K}$ is the minimizer of the energy. Computing the desired circle packing metric with prescribed curvature $\bar{K}$ is equivalent to minimizing the discrete Ricci energy. The discrete Ricci energy is strictly convex (namely, its Hessian is positive definite after a normalization). The global minimum uniquely exists, corresponding to the metric $\bar{\mathbf{u}}$, which induces $\bar{K}$. The discrete Ricci flow converges to this global minimum and the convergence is exponentially fast [2], i.e., $|\bar{K}_i - K_i(t)| < c_1 e^{-c_2 t}$, where c_1, c_2 are two positive constants. This represents a centralized algorithm for computing the discrete Ricci flow on Σ. In the following, we describe the distributed algorithm for different types of greedy routing scenarios.

Discrete Ricci Flow Algorithm

To apply discrete Ricci flow for greedy routing, we take a triangular mesh Σ as a subgraph from the communication graph. All non-triangular faces are considered as network holes that will be mapped to circular holes in the embedding. All nodes not on hole boundaries have zero curvature under the mapping. Thus, the embedding is denoted as a circular domain. With the virtual coordinates and Euclidean distance metric, greedy routing guarantees delivery. (For a node in the interior of the triangulation, if the corner angle is greater than $2\pi/3$, we will adopt greedy routing on an edge that has provably guaranteed delivery.)

In particular, we set all edge lengths to be initially 1, which determines the initial curvature at each node. In particular, we choose the circle packing metric by placing a circle of initial radius $1/2$ on each node. The circles at adjacent nodes are tangent to each other. Thus, the intersection angle is kept at 0. We now set the target curvature at interior nodes to be zero and at hole boundary nodes to be $2\pi/k$ with k as the number of nodes on the hole boundary. The algorithms run in a gossip style. In each round, each node exchanges its radius with neighbors and computes its own Gaussian curvature. The algorithm stops when the current curvature is within error ϵ from the specified target curvature.

At each gossip round, node v_i is associated with a disk with radius e^{u_i}, where u_i is a scalar value. The length of the edge connecting v_i and v_j equals to $e^{u_i} + e^{u_j}$. The corner angles of each triangle can be estimated using cosine law by each node locally. That is, the angle θ_i^{jk} in triangle $[v_i, v_j, v_k]$ is

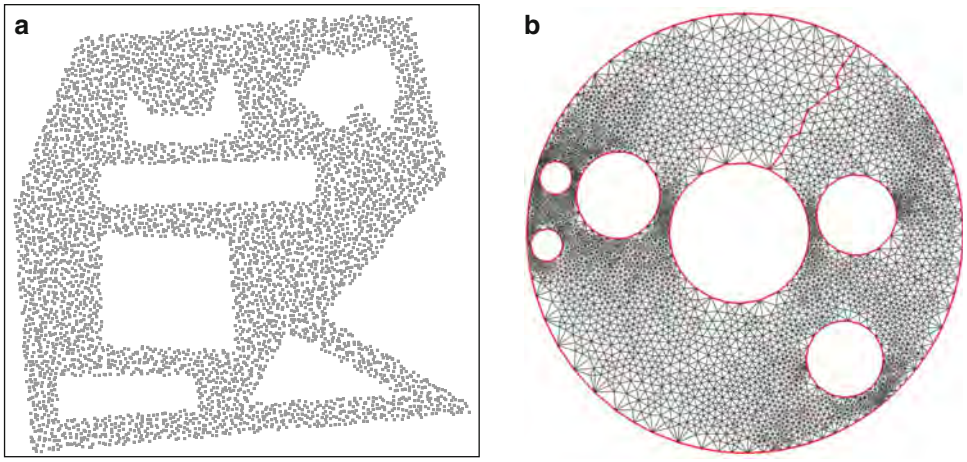
$$\theta_i^{jk} = \cos^{-1} \frac{l_{ij}^2 + l_{ki}^2 - l_{jk}^2}{2 l_{ij} l_{ki}}$$

The curvature k_i at v_i is

$$k_i = \begin{cases} 2\pi - \sum_{jk} \theta_i^{jk}, & v_i \notin \partial M \\ \pi - \sum_{jk} \theta_i^{jk}, & v_i \in \partial M \end{cases}$$

When the target curvature is not met, u_i is modified proportionally to the difference between the target curvature and the current curvature.

$$u_i \Leftarrow u_i + \delta(\bar{k}_i - k_i)$$



Discrete Ricci Flow for Geometric Routing, Fig. 2 (**a**) A network of 7,000 nodes with many holes; (**b**) virtual coordinates

Once the curvatures are computed, the triangulation is then flattened out by a simple flooding from a triangle root. Given three edge lengths of the root triangle $[v_0, v_1, v_2]$, the node coordinates can be constructed directly. Then the neighboring triangle of the root, e.g., $[v_1, v_0, v_i]$, can be flattened; the virtual coordinates of v_i are the intersection of two circles, one is centered at v_0 with radius l_{0i} and the other is centered at v_1 with radius l_{1i}. In a similar way, the neighbors of the newly flattened triangles can be further embedded. The virtual coordinates of the whole network are thus computed (Fig. 2).

Discrete Hyperbolic Ricci Flow

The key result in conformal geometry says that any surface with a Riemannian metric admits a Riemannian metric of constant Gaussian curvature, which is conformal to the original metric. Such metric is called the uniformization metric. Thus, depending on the surface topology, the uniformization metric has either positive constant, zero, or negative constant curvature everywhere. Simply connected surfaces with constant curvature are only of three canonical types: the sphere (constant positive curvature everywhere), the Euclidean plane (zero curvature everywhere), and the hyperbolic plane (negative curvature everywhere). Discrete Ricci flow is a powerful tool to compute the uniformization metric.

In our setting, when the triangulation Σ has two or more holes, it has negative total curvature. Thus, its uniformization metric is hyperbolic. To actually embed the surface and realize the uniformization metric, the holes in the network are cut open to get a simply connected triangulation T. Using discrete hyperbolic Ricci flow, we embed T in a convex region S in hyperbolic space. Each node is given a hyperbolic coordinate. Each edge uv has a length $d(u, v)$ as the geodesic between u, v in the hyperbolic space. In this way, greedy routing with the hyperbolic metric (i.e., send the message to the neighbor closer to the destination measured by hyperbolic distance) has *guaranteed delivery*.

The hyperbolic Ricci flow is very similar to the Euclidean version with a few modifications. First all metrics are hyperbolic. The edge length l_{ij} of e_{ij} is determined by the hyperbolic cosine law:

$$\cosh l_{ij} = \cosh \gamma_i \cosh \gamma_j + \sinh \gamma_i \sinh \gamma_j \cos \phi_{ij}. \tag{1}$$

Let $u_i = \log \tanh \frac{\gamma_i}{2}$; *the discrete Ricci flow* is defined as

$$\frac{du_i(t)}{dt} = -K_i, \tag{2}$$

where K_i is the discrete Gaussian curvature at v_i. Once the hyperbolic metric is computed, we can embed the triangulation isometrically onto the Poincare disk.

Generalized Discrete Surface Ricci Flow

There are many schemes for discrete surface Ricci flow [14], including tangential circle packing, Thurston's circle packing, inversive distance circle packing, Yamabe flow, virtual radius circle packing, and mixed typed schemes. All of them can be unified as follows. The combinatorial structure of the triangulation is Σ; it is with one of three background geometries: Euclidean $\mathbb{E}^2$, hyperbolic $\mathbb{H}^2$, and spherical $\mathbb{S}^2$. Each vertex is associated with a circle; the vertex radii function is $\gamma : V \to \mathbb{R}^+$. Each vertex is also associated with a constant ϵ, which indicates the scheme. Each edge has a conformal structure coefficient $\eta : E \to \mathbb{R}$. So a circle packing metric is given by $(\Sigma, \gamma, \eta, \epsilon)$. The discrete conformal factor is given by

$$u_i = \begin{cases} \log \gamma_i & , \mathbb{E}^2 \\ \log \tanh \frac{\gamma_i}{2} & , \mathbb{H}^2 \\ \log \tan \frac{\gamma_i}{2} & , \mathbb{S}^2 \end{cases}$$

The length of $[v_i, v_j]$ is given by

$$\begin{cases} l_{ij}^2 & = 2\eta_{ij} e^{u_i+u_j} + \epsilon_i e^{2u_i} + \epsilon_j e^{2u_j} & , \mathbb{E}^2 \\ \cosh l_{ij} & = \frac{4\eta_{ij} e^{u_i+u_j} + (1+\epsilon_i e^{2u_i})(1+\epsilon_j e^{2u_j})}{(1-\epsilon_i e^{2u_i})(1-\epsilon_j e^{2u_j})} & , \mathbb{H}^2 \\ \cos l_{ij} & = \frac{4\eta_{ij} e^{u_i+u_j} + (1-\epsilon_i e^{2u_i})(1-\epsilon_j e^{2u_j})}{(1+\epsilon_i e^{2u_i})(1+\epsilon_j e^{2u_j})} & , \mathbb{S}^2 \end{cases}$$

The discrete Ricci flow is given by

$$\frac{du_i(t)}{dt} = \bar{K}_i - K_i(t),$$

where $\bar{K} : V \to \mathbb{R}$ is the prescribed target curvature, which is the negative gradient flow of the discrete Ricci energy

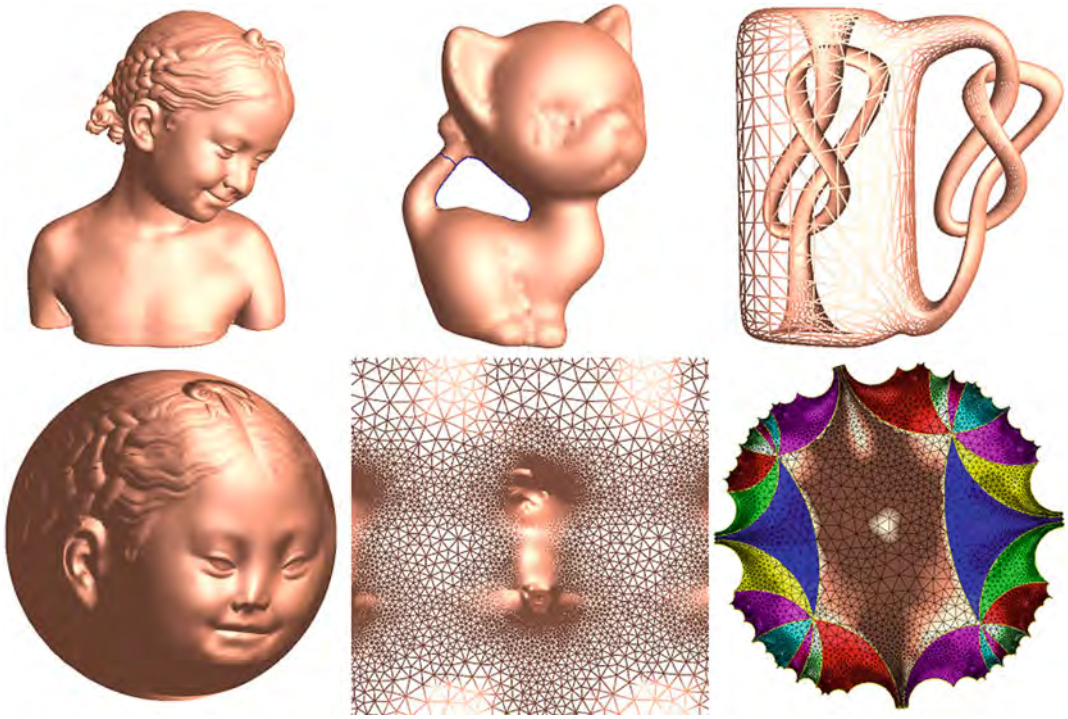
$$E(u) = \int^{\mathbf{u}} \sum_i (\bar{K}_i - K_i) du_i.$$

For the discrete surfaces with Euclidean background geometry, the Ricci energy is convex on the space $\sum_i u_i = 0$. For those with hyperbolic background geometry, the energy is convex. For spherical case, the energy is indefinite.

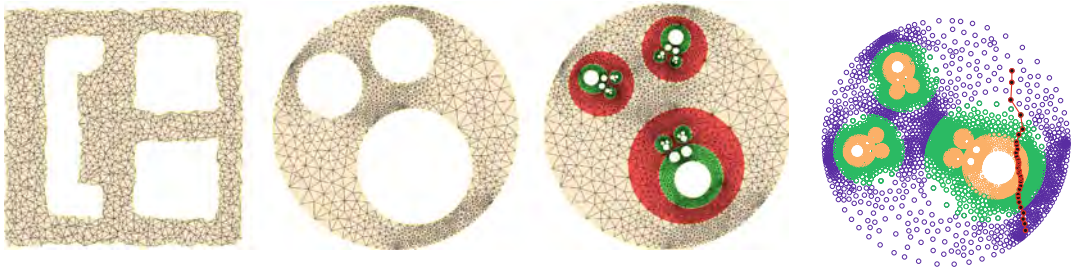
For Yamabe scheme (where $\epsilon \equiv 0$), the combinatorial structure Σ is Delaunay, if for each edge $[v_i, v_j]$ share by two faces $[v_i, v_j, v_k]$ and $[v_j, v_i, v_l]$, $\theta_{ij}^k + \theta_{ji}^l \leq \pi$. If during the Yamabe flow, the combinatorial structure can be updated to ensure the Delaunay condition, then for any $\bar{K} : V \to (-\infty, 2\pi)$ satisfying the Gauss-Bonnet constraint $\sum_{v \in V} \bar{K}(v) = 2\pi\chi(\Sigma)$, the Yamabe flow with surgery can lead to the discrete metric that realizes the target curvature; the convergence is exponentially fast. This theorem implies the discrete uniformization theorem: any closed polyhedral surface admits a polyhedral metric discretely conformal to the original one, which induces constant Gaussian curvature everywhere [4,5] (Fig. 3).

Applications

The presented Ricci flow algorithms can be applied for a variety of routing primitives for large-scale wireless sensor networks with nonuniform node distribution. Besides guaranteed delivery [8], we can also achieve multiple additional desirable routing objectives, all derived from the unique property of a conformal mapping. For example, greedy routing on a circular domain may accumulate high traffic load on the interior hole boundaries. To alleviate that, we can reflect the network along a hole boundary using a Mobius transformation and map a copy of the network to cover the interior of the hole, recursively [9] (see Fig. 4). Routing on this covering space makes traffic load more balanced as hole boundaries essentially "disappear." In another case, when there are sudden link or node failures, we can apply a Mobius transformation to generate a different circular domain, with the sizes and positions of the holes rearranged, on which greedy routing generates a different path [6]. Thus, quick recovery from a spontaneous failure is possible. The hyperbolic Ricci flow can be used to map the domain with the holes cut open to a convex polygon that can tile up the entire hyperbolic



Discrete Ricci Flow for Geometric Routing, Fig. 3 Discrete surface uniformization

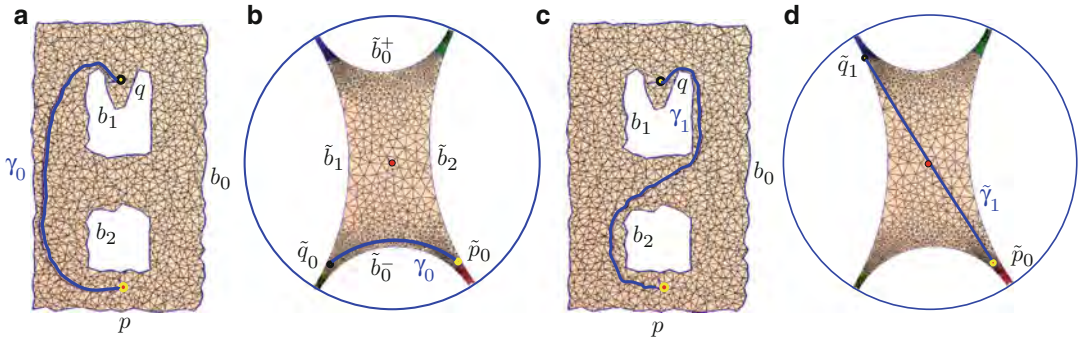


Discrete Ricci Flow for Geometric Routing, Fig. 4 Three-level circular reflections and a routing path

plane. This mapping supports greedy routing with specified "homotopy types," i.e., routes that go around holes in different ways [13] (see Fig. 5). Hyperbolic embedding can be generalized to 3D sensor networks with complex topology as in the case of monitoring underground tunnels [12]. Additional applications include generation of "space filling curves" for arbitrary domains [1], supporting greedy routing in mobile networks [7] and load balanced routing [3].

Open Problems

Given a smooth surface S with a Riemannian metric $\mathbf{g}$, the smooth Ricci flow leads to the uniformization metric $e^{2\lambda}\mathbf{g}$, where λ is the smooth conformal factor. If the surface is tessellated to get a discrete surface M_0 and discrete Ricci flow is performed on M_0, one obtains discrete conformal factor function u_0. When M is subdivided by n times, the discrete conformal factor is u_n, whether $\lim_{n\to\infty} u_n = \lambda$.



Discrete Ricci Flow for Geometric Routing, Fig. 5 Computing the shortest paths using the hyperbolic embedding of a 3-connected domain with 1,286 nodes. Two different paths are generated using greedy routing toward images of the destination in different patches. (**a**) Shortest path 1. (**b**) Geodesic of (**a**). (**c**) Shortest path 2. (**d**) Geodesic of (**c**)

Experimental Results

The convergence rate, i.e., the number of iterations is proportional to $O(\frac{log(1/\epsilon)}{\delta})$, where δ is the step size in the Ricci flow algorithm and ϵ is the error bound on the curvature. In our experiments we take ϵ to be $1e-6$. Routing with the virtual coordinates has 100 % delivery rate and the average path stretch (compared to the shortest path in the network) is no greater than 2.

URLs to Code and Data Sets

http://www.cs.sunysb.edu/~gu/tutorial/RicciFlow.html

Recommended Reading

1. Ban X, Goswami M, Zeng W, Gu XD, Gao J (2013) Topology dependent space filling curves for sensor networks and applications. In: Proceedings of 32nd annual IEEE conference on computer communications (INFOCOM'13), Turin
2. Chow B, Luo F (2003) Combinatorial Ricci flows on surfaces. J Differ Geom 63(1):97–129
3. Goswami M, Ni C-C, Ban X, Gao J, Xianfeng Gu D, Pingali V (2014) Load balanced short path routing in large-scale wireless networks using area-preserving maps. In: Proceedings of the 15th ACM international symposium on mobile ad hoc networking and computing (Mobihoc'14), Philadelphia, Aug 2014. pp 63–72
4. Gu X, Luo F, Sun J, Wu T (2013) A discrete uniformization theorem for polyhedral surfaces. arXiv:13094175
5. Gu X, Guo R, Luo F, Sun J, Wu T (2014) A discrete uniformization theorem for polyhedral surfaces ii. arXiv:14014594
6. Jiang R, Ban X, Goswami M, Zeng W, Gao J, Gu XD (2011) Exploration of path space using sensor network geometry. In: Proceedings of the 10th international symposium on information processing in sensor networks (IPSN'11), Chicago, pp 49–60
7. Li S, Zeng W, Zhou D, Gu XD, Gao J (2013) Compact conformal map for greedy routing in wireless mobile sensor networks. In: Proceedings of 32nd annual IEEE conference on computer communications (INFOCOM'13), Turin
8. Sarkar R, Yin X, Gao J, Luo F, Gu XD (2009) Greedy routing with guaranteed delivery using Ricci flows. In: Proceedings of the 8th international symposium on information processing in sensor networks (IPSN'09), San Francisco, pp 97–108
9. Sarkar R, Zeng W, Gao J, Gu XD (2010) Covering space for in-network sensor data storage. In: Proceedings of the 9th international symposium on information processing in sensor networks (IPSN'10), Stockholm, pp 232–243
10. Stephenson K (2005) Introduction to circle packing: the theory of discrete analytic functions. Cambridge University Press, New York
11. Thurston WP (1997) Three-dimensional geometry and topology, vol 1. Princeton University Press, Princeton
12. Yu X, Yin X, Han W, Gao J, Gu XD (2012) Scalable routing in 3d high genus sensor networks using graph embedding. In: Proceedings of the 31st annual IEEE conference on computer communications (INFOCOM'12), Orlando, pp 2681–2685
13. Zeng W, Sarkar R, Luo F, Gu XD, Gao J (2010) Resilient routing for sensor networks using hyperbolic embedding of universal covering space. In:

Proceedings of the 29th annual IEEE conference on computer communications (INFOCOM'10), San Diego, pp 1694–1702
14. Zhang M, Guo R, Zeng W, Luo F, Yau ST, Gu X (2014) The unified discrete surface Ricci flow. Graph Models. doi:. http://www.sciencedirect.com/science/article/pii/S1524070314000344

Distance Oracles for Sparse Graphs

Liam Roditty
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Keywords

Data structures; Distance oracles; Graph algorithms; Shortest paths

Years and Authors of Summarized Original Work

2005; Thorup, Zwick
2012; Pătraşcu, Roditty
2012; Pătraşcu, Roditty, Thorup

Problem Definition

Let $G = (V, E)$ be a weighted undirected graph with n vertices and m edges. A distance oracle is a data structure capable of representing almost shortest paths efficiently, both in terms of space requirement and query time. Thorup and Zwick [7] showed that for any integer $k \geq 1$ it is possible to preprocess the graph in $\tilde{O}(mn^{1/k})$ time and generate a compact data structure of size $O(kn^{1+1/k})$ that answers approximate distance queries with $2k - 1$ multiplicative stretch in $O(k)$ time. This means that for every $u, v \in V$, it is possible to retrieve an estimate $\hat{d}(u, v)$ to the distance $d(u, v)$ in $O(k)$ time, such that $d(u, v) \leq \hat{d}(u, v) \leq (2k - 1)d(u, v)$. Recently, [8] showed, using a clever query algorithm, that the query time of Thorup and Zwick can be reduced from $O(k)$ to $O(\log k)$. Even more recently, [1] showed that the query time of Thorup and Zwick can be reduced to $O(1)$. Thorup and Zwick [7] showed, based on the girth conjecture of [2], that there are dense enough graphs which cannot be represented by a data structure of size less than $n^{1+1/k}$ without increasing the stretch above $2k - 1$ for any integer k. Therefore, for dense graphs their distance oracle is optimal assuming the girth conjecture holds.

This suggests that the distance oracle of Thorup and Zwick can be improved only for sparse graphs and in particular, graphs with less than $n^{1+1/k}$ edges. Alternatively, it might be possible to get below the $2k - 1$ multiplicative stretch by allowing an additive stretch as well.

Notice, however, that we cannot gain from introducing also additive stretch without getting an improved multiplicative stretch distance oracles for sparse graphs (i.e., $m = O(n)$). A data structure with size $S(m, n)$ and stretch (α, β), where α is multiplicative stretch and β is additive stretch, implies a data structure with size $S((\beta + 1)m, n + \beta m)$ and multiplicative stretch of α, as if we divide every edge into $\beta + 1$ edges then all distances become a multiply of $\beta + 1$ and additive stretch of β is useless. For graphs with $m = O(n)$, the size of the data structure is asymptotically the same.

Key Results

Pătraşcu and Roditty [4] obtained a distance oracle for sparse unweighted graphs with $m = \tilde{O}(n)$ of size $\tilde{O}(m^{5/3})$ that can supply in $O(1)$ time an estimate of the distance with multiplicative stretch 2. For dense graphs, the distance oracle has size of $\tilde{O}(n^{5/3})$ and stretch $(2, 1)$.

Pătraşcu et al. [5] extended this result for weighted graphs and generalized it. In particular, they show that for any fixed positive integers k and ℓ, there is a distance oracle with stretch $\alpha = 2k + 1 \pm \frac{2}{\ell} = 2k + 1 - \frac{2}{\ell}, 2k + 1 + \frac{2}{\ell}$, that uses $\tilde{O}(m^{1+2/(\alpha+1)})$ space. The query time is $O(k + \ell)$.

Sommer et al. [6] proved a three-way tradeoff between space, stretch, and query time of approximate distance oracles. They show that any distance oracle that can give stretch α answers to distance queries in time $O(t)$ must use $n^{1+\Omega(1/(t\alpha))}/\log n$ space. Their result is obtained by a reduction from lopsided set disjointness to distance oracles, using the framework introduced by [3]. Any improvement to this lower bound requires a major breakthrough in lower bounds techniques. In particular, it does not imply anything even for slightly non-constant query time as $\Omega(\log n)$ and slightly non-linear space as $n^{1.01}$.

Pătraşcu and Roditty [4] showed also a conditional lower bound for distance oracle that is based on a conjecture on the hardness of the set intersection problem. They showed that a distance oracle for unweighted graphs with $m = \tilde{O}(n)$ edges, which can distinguish between distances of 2 and 4 in constant time (as multiplicative stretch strictly less than 2 implies) requires $\tilde{\Omega}(n^2)$ space, assuming the conjecture holds. Thus, non-constant query time is essential to get stretch smaller than 2.

Pătraşcu et al. [5] showed, based also on a conjecture on the hardness of the set intersection problem, for any fixed positive integer ℓ, that there are graphs with m edges such that a distance oracle with constant query time and stretch below $3 - 2/(\ell + 1)$ must use space $\tilde{\Omega}(m^{1+1/(2-1/\ell)})$.

Open Problems

The conditional lower bounds of [5] for sparse graphs do not say anything on stretch 3. The best space upper bound for stretch 3 in sparse graphs and dense graphs is $\tilde{O}(n^{1.5})$. While in dense graphs this is tight due to the existence of graphs with $\Omega(n^{1.5})$ edges and girth 6, for sparse graphs nothing is known. Therefore, we have the following two open problems:

Can we get a $o(n^{1.5})$ space for stretch 3 in sparse graphs? Can we get stretch less than 3 for space $\tilde{O}(n^{1.5})$ in sparse graphs?

Recommended Reading

1. Chechik S (2014) Approximate distance oracles with constant query time. In: STOC, New York
2. Erdős P (1964) Extremal problems in graph theory. In: Simonovits M (ed) Theory of graphs and its applications, pp 29–36. https://www.renyi.hu/~p_erdos/1970-22.pdf
3. Pătraşcu M (2008) (Data) structures. In: Proceedings of 49th FOCS, Philadelphia, pp 434–443
4. Pătraşcu M, Roditty L (2014) Distance oracles beyond the Thorup-Zwick bound. SIAM J Comput 43(1):300–311
5. Pătraşcu M, Roditty L, Thorup M (2012) A new infinity of distance oracles for sparse graphs. In: FOCS, New Brunswick, pp 738–747
6. Sommer C, Verbin E, Yu W (2009) Distance oracles for sparse graphs. In: Proceedings of 50th FOCS, Atlanta, pp 703–712
7. Thorup M, Zwick U (2005) Approximate distance oracles. JACM 52(1):1–24
8. Wulff-Nilsen C (2013) Approximate distance oracles with improved query time. In: SODA, New Orleans, pp 539–549

Distance-Based Phylogeny Reconstruction (Fast-Converging)

Miklós Csürös
Department of Computer Science, University of Montréal, Montréal, QC, Canada

Keywords

Learning an evolutionary tree

Years and Authors of Summarized Original Work

2003; King, Zhang, Zhou

Problem Definition

Introduction

From a mathematical point of view, a phylogeny defines a probability space for random sequences observed at the leaves of a binary tree T. The tree T represents the unknown hierarchy

of common ancestors to the sequences. It is assumed that (unobserved) ancestral sequences are associated with the inner nodes. The tree along with the associated sequences models the evolution of a molecular sequence, such as the protein sequence of a gene. In the conceptually simplest case, each tree node corresponds to a species, and the gene evolves within the organismal lineages by vertical descent.

Phylogeny reconstruction consists of finding T from observed sequences. The possibility of such reconstruction is implied by fundamental principles of molecular evolution, namely, that random mutations within individuals at the genetic level spreading to an entire mating population are not uncommon, since often they hardly influence evolutionary fitness [15]. Such mutations slowly accumulate, and, thus, differences between sequences indicate their evolutionary relatedness.

The reconstruction is theoretically feasible in several known situations. In some cases, distances can be computed between the sequences, and used in a distance-based algorithm. Such an algorithm is fast-converging if it almost surely recovers T, using sequences that are polynomially long in the size of T. Fast-converging algorithms exploit statistical concentration properties of distance estimation.

Formal Definitions

An evolutionary *topology* $U(X)$ is an unrooted binary tree in which leaves are bijectively mapped to a set of species X. A *rooted topology* T is obtained by rooting a topology U on one of the edges uv: a new node ρ is added (the *root*), the edge uv is replaced by two edges ρv and ρu, and the edges are directed outwards on paths from ρ to the leaves. The edges, vertices, and leaves of a rooted or unrooted topology T are denoted by $\mathrm{E}(T)$, $\mathrm{V}(T)$ and $\mathrm{L}(T)$, respectively.

The edges of an unrooted topology U may be equipped with a a positive *edge length* function $d\colon E(U) \mapsto (0,\infty)$. Edge lengths induce a *tree metric* $d\colon \mathrm{V}(U) \times \mathrm{V}(U) \mapsto [0,\infty)$ by the extension $d(u,v) = \sum_{e \in u \rightsquigarrow v} d(e)$, where $u \rightsquigarrow v$ denotes the unique path from u to v. The value $d(u,v)$ is called the *distance* between u and v. The pairwise distances between leaves form a *distance matrix*.

An *additive tree metric* is a function $\delta\colon X \times X \mapsto [0,\infty)$ that is equivalent to the distance matrix induced by some topology $U(X)$ and edge lengths. In certain random models, it is possible to define an additive tree metric that can be estimated from dissimilarities between sequences observed at the leaves.

In a *Markov model of character evolution* over a rooted topology T, each node u has an associated *state*, which is a random variable $\xi(u)$ taking values over a fixed alphabet $\mathrm{A} = \{1, 2, \ldots r\}$. The vector of leaf states constitutes the *character* $\xi = \big(\xi(u)\colon u \in \mathrm{L}(T)\big)$. The states form a first-order Markov chain along every path. The joint distribution of the node states is specified by the marginal distribution of the root state, and the conditional probabilities $\mathbb{P}\{\xi(v) = b | \xi(u) = a\} = p_e(a \to b)$ on each edge e, called *edge transition probabilities*.

A *sample* of length ℓ consists of independent and identically distributed characters $\Xi = \big(\xi_i\colon i = 1, \ldots \ell\big)$. The *random sequence* associated with the leaf u is the vector $\Xi(u) = \big(\xi_i(u)\colon i = 1, \ldots \ell\big)$.

A *phylogeny reconstruction algorithm* is a function F mapping samples to unrooted topologies. The *success probability* is the probability that $\mathrm{F}(\Xi)$ equals the true topology.

Popular Random Models

Neyman Model [14]

The edge transition probabilities are

$$p_e(a \to b) = \begin{cases} 1 - \mu_e & \text{if a = b;} \\ \frac{\mu_e}{r-1} & \text{if a} \neq \text{b} \end{cases}$$

with some edge-specific mutation probability $0 < \mu_e < 1 - 1/r$. The root state is uniformly distributed. A distance is usually defined by

$$d(u,v) = -\tfrac{r-1}{r} \ln\Big(1 - \tfrac{r}{r-1}\mathbb{P}\{\xi(u) \neq \xi(v)\}\Big).$$

General Markov Model

There are no restrictions on the edge transition probabilities in the general Markov model. For identifiability [1, 16], however, it is usually assumed that $0 < \det \mathbf{P}_e < 1$, where $\mathbf{P}_e$ is the stochastic matrix of edge transition probabilities. Possible distances in this model include the *paralinear* distance [1, 12] and the *LogDet* distance [13, 16]. This latter is defined by $d(u, v) = -\ln \det \mathbf{J}_{uv}$, where J_{uv} is the matrix of joint probabilities for $\xi(u)$ and $\xi(v)$.

It is often assumed in practice that sequence evolution is effected by a continuous-time Markov process operating on the edges. Accordingly, the edge length directly measures time. In particular, $\mathbf{P}_e = e^{\mathbf{Q}\cdot d(e)}$ on every edge e, where $\mathbf{Q}$ is the instantaneous rate matrix of the underlying process.

Key Results

It turns out that the hardness of reconstructing an unrooted topology U from distances is determined by its *edge depth* $\rho(U)$. Edge depth is defined as the smallest integer k for which the following holds. From each endpoint of every edge $e \in \mathrm{E}(U)$, there is a path leading to a leaf, which does not include e and has at most k edges.

Theorem 1 (Erdős, Steel, Székely, Warnow [6]) *If U has n leaves, then $\rho(U) \leq 1 + \log_2(n-1)$. Moreover, for almost all random n-leaf topologies under the uniform or Yule-Harding distributions, $\rho(U) \in O(\log \log n)$*

Theorem 2 (Erdős, Steel, Székely, Warnow [6]) *For the Neyman model, there exists a polynomial-time algorithm that has a success probability $(1-\delta)$ for random samples of length*

$$\ell = O\Big(\frac{\log n + \log \frac{1}{\delta}}{f^2(1-2g)^{4\rho+6}}\Big), \qquad (1)$$

where $0 < f = \min_e \mu_e$ and $g = \max_e \mu_e < 1/2$ are extremal edge mutation probabilities, and ρ is the edge depth of the true topology.

Theorem 2 can be extended to the general Markov model with analogous success rates for LogDet distances [7], as well as to a number of other Markov models [2].

Equation (1) shows that phylogenies can be reconstructed with high probability from polynomially long sequences. Algorithms with such sample size requirements were dubbed *fast-converging* [9]. Fast convergence was proven for the short quartet methods of Erdős et al. [6, 7], and for certain variants [11] of the so-called disk-covering methods introduced by Huson et al. [9]. All these algorithms run in $\Omega(n^5)$ time. Csürös and Kao [3] initiated the study of computationally efficient fast-converging algorithms, with a cubic-time solution. Csürös [2] gave a quadratic-time algorithm. King et al. [10] designed an algorithm with an optimal running time of $O(n \log n)$ for producing a phylogeny from a matrix of estimated distances.

The short quartet methods were revisited recently: [4] described an $O(n^4)$-time method that aims at succeeding even if only a short sample is available. In such a case, the algorithm constructs a forest of "trustworthy" edges that match the true topology with high probability.

All known fast-converging distance-based algorithms have essentially the same sample bound as in (1), but Daskalakis et al. [5] recently gave a twist to the notion of fast convergence. They described a polynomial-time algorithm, which outputs the true topology almost surely from a sample of size $O(\log n)$, given that edge lengths are not too large. Such a bound is asymptotically optimal [6]. Interestingly, the sample size bound does not involve exponential dependence on the edge depth: the algorithm does not rely on a distance matrix.

Applications

Phylogenies are often constructed in molecular evolution studies, from aligned DNA or protein sequences. Fast-converging algorithms have mostly a theoretical appeal at this point. Fast convergence promises a way to handle the increasingly important issue of constructing

large-scale phylogenies: see, for example, the CIPRES project (http://www.phylo.org/).

Cross-References

Similar algorithmic problems are discussed under the heading

▸ Distance-Based Phylogeny Reconstruction: Safety and Edge Radius

Recommended Reading

Joseph Felsenstein wrote a definitive guide [8] to the methodology of phylogenetic reconstruction.

1. Chang JT (1996) Full reconstruction of Markov models on evolutionary trees: identifiability and consistency. Math Biosci 137:51–73
2. Csürös M (2002) Fast recovery of evolutionary trees with thousands of nodes. J Comput Biol 9(2):277–297, Conference version at RECOMB 2001
3. Csürös M, Kao M-Y (2001) Provably fast and accurate recovery of evolutionary trees through harmonic greedy triplets. SIAM J Comput 31(1):306–322, Conference version at SODA (1999)
4. Daskalakis C, Hill C, Jaffe A, Mihaescu R, Mossel E, Rao S (2006) Maximal accurate forests from distance matrices. In: Proceedings of research in computational biology (RECOMB), pp 281–295
5. Daskalakis C, Mossel E, Roch S (2006) Optimal phylogenetic reconstruction. In: Proceedings of ACM symposium on theory of computing (STOC), pp 159–168
6. Erdős PL, Steel MA, Székely LA, Warnow TJ (1999) A few logs suffice to build (almost) all trees (I). Random Struct Algorithm 14:153–184, Preliminary version as DIMACS TR97-71
7. Erdős PL, Steel MA, Székely LA, Warnow TJ (1999) A few logs suffice to build (almost) all trees (II). Theor Comput Sci 221:77–118, Preliminary version as DIMACS TR97-72
8. Felsenstein J (2004) Inferring pylogenies. Sinauer Associates, Sunderland
9. Huson D, Nettles S, Warnow T (1999) Disk-covering, a fast converging method of phylogenetic reconstruction. J Comput Biol 6(3–4):369–386, Conference version at RECOMB (1999)
10. King V, Zhang L, Zhou Y (2003) On the complexity of distance based evolutionary tree reconstruction. In: Proceedings of ACM-SIAM symposium on discrete algorithms (SODA), pp 444–453
11. Lagergren J (2002) Combining polynomial running time and fast convergence for the disk-covering method. J Comput Syst Sci 65(3):481–493
12. Lake JA (1994) Reconstructing evolutionary trees from DNA and protein sequences: paralinear distances. Proc Natl Acad Sci USA 91:1455–1459
13. Lockhart PJ, Steel MA, Hendy MD, Penny D (1994) Recovering evolutionary trees under a more realistic model of sequence evolution. Mol Biol Evol 11:605–612
14. Neyman J (1971) Molecular studies of evolution: a source of novel statistical problems. In: Gupta SS, Yackel J (eds) Statistical decision theory and related topics. Academic, New York, pp 1–27
15. Ohta T (2002) Near-neutrality in evolution of genes and gene regulation. Proc Natl Acad Sci USA 99:16134–16137
16. Steel MA (1994) Recovering a tree from the leaf colourations it generates under a Markov model. Appl Math Lett 7:19–24

Distance-Based Phylogeny Reconstruction: Safety and Edge Radius

Olivier Gascuel[1], Fabio Pardi[1], and Jakub Truszkowski[2,3]

[1]Institut de Biologie Computationnelle, Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), CNRS and Université de Montpellier, Montpellier cedex 5, France

[2]Cancer Research UK Cambridge Institute, University of Cambridge, Cambridge, UK

[3]European Molecular Biology Laboratory, European Bioinformatics Institute (EMBL-EBI), Wellcome Trust Genome Campus, Hinxton, Cambridge, UK

Keywords

Distance methods; Optimal radius; Performance analysis; Phylogeny reconstruction; Robustness; Safety radius approach

Years and Authors of Summarized Original Work

1999; Atteson
2005; Elias, Lagergren
2006; Dai, Xu, Zhu

2010; Pardi, Guillemot, Gascuel 2013; Bordewich, Mihaescu

Problem Definition

A phylogeny is an evolutionary tree tracing the shared history, including common ancestors, of a set of extant species or "taxa." Phylogenies are increasingly reconstructed on the basis of molecular data (DNA and protein sequences) using statistical techniques such as likelihood and Bayesian methods. Algorithmically, these techniques suffer from the discrete nature of tree topology space. Since the number of tree topologies increases exponentially as a function of the number of taxa, and each topology requires a separate likelihood calculation, it is important to restrict the search space and to design efficient heuristics. Distance methods for phylogeny reconstruction serve this purpose by inferring trees in a fraction of the time required for the more statistically rigorous methods. Distance methods also provide fairly accurate starting trees to be further refined by more sophisticated methods. Moreover, the input to a distance method is the matrix of pairwise evolutionary distances among taxa, which are estimated by maximum likelihood, so that distance methods also have sound statistical justifications.

Mathematically, a phylogenetic tree is a triple $T = (V, E, l)$ where V is the set of nodes (extant taxa correspond to leaves, ancestors to internal nodes), E is the set of edges (branches) representing relations of descent, and l is a function that assigns positive lengths to each edge in E, representing a measure of evolutionary divergence, for example, in terms of time, or amount of change between DNA and protein sequences. Any phylogenetic tree T defines a metric D_T on its leaf set L: let $P_T(u, v)$ define the unique path through T from u to v; then the distance from u to v is set to $D_T(u, v) = \sum_{e \in P_T(u,v)} l(e)$.

Distance methods for phylogeny reconstruction rely on the fundamental result [22] that the map $T \rightarrow D_T$ is reversible; i.e., a tree T can be reconstructed from its tree metric, a problem that can be solved in $O(n \log n)$ time [14]. However, in practice D_T is not known, and one must use molecular sequence data to estimate a distance matrix D that approximates D_T [9]. As the amount of sequence data increases, D can be assumed to converge to D_T. A minimal requirement for any distance method is *consistency*: for any tree T, and for distance matrices D "close enough" to D_T, the algorithm should output a tree with the same topology as T (i.e., with the same underlying graph (V,E)). The present chapter deals with the question of when any distance algorithm for phylogeny reconstruction can be guaranteed to output the correct phylogeny as a function of the divergence between D and D_T. Atteson [1] demonstrated that this question can be precisely answered for neighbor joining (NJ) [18], one of the most cited algorithms in computational biology (with more than 35,000 citations up to 2014), and a number of NJ's variants.

The Neighbor Joining (NJ) Algorithm of Saitou and Nei [18]

NJ is *agglomerative*: it works by using the input matrix D to identify a pair of taxa $x, y \in L$ that are neighbors in T, i.e., there exists a node $u \in V$ such that $\{(u,x), (u,y)\} \subset E$. Then, the algorithm creates a node c that is connected to x and y, extends the distance matrix to c, and solves the reduced problem on $L \cup \{c\} \setminus \{x,y\}$. The pair (x, y) is chosen to minimize the following sum:

$$S_D(x, y) = (|L| - 2) \cdot D(x, y) - \sum_{z \in L} (D(z, x) + D(z, y)).$$

The soundness of NJ is based on the observation that, if $D = D_T$ for a tree T, the value $S_D(x, y)$ will be minimized for a pair (x, y) that are neighbors in T.

Balanced Minimum Evolution and Algorithms Inspired by It

A number of papers (reviewed in [11]) have been dedicated to the various interpretations and properties of the S_D criterion. One of these interpreta-

tions consists of observing that agglomerating the pair of nodes that minimizes S_D is equivalent to choosing, among all the trees that can be obtained in this way, the one that minimizes a simple linear formula [16] to calculate the length of a tree from the distances between its leaves [11], thus connecting distance and parsimony methods [9]. As the optimization principle seeking the tree that minimizes this formula has been named balanced minimum evolution (BME) [6], NJ can then be seen as a greedy algorithm for BME.

This remarkable connection between NJ and BME naturally spurred the proposal of alternative algorithms for BME. One of these, GreedyBME, consists of iteratively adding taxa to a tree so that, at each step, the resulting tree is the one that minimizes BME among all the binary trees that can be obtained in this way [6]. More involved algorithms can be obtained by combining a simple tree construction algorithm such as NJ or GreedyBME, with a local search based on the traditional tree rearrangements used in phylogenetics [9], such as *nearest-neighbor interchange* (NNI) or *subtree pruning and regrafting* (SPR).

The Fast Neighbor Joining (FNJ) Algorithm of Elias and Lagergren [7]

Standard implementations of NJ require $O(n^3)$ computations, where n is the number of taxa in the data set. Since a distance matrix only has n^2 entries, many attempts have been made to construct a distance algorithm that would only require $O(n^2)$ computations while retaining the accuracy of NJ. To this end, one of the most interesting results is the fast neighbor joining (FNJ) algorithm of Elias and Lagergren [7].

Most of the computation of NJ is used in the recalculations of the sums $S_D(x, y)$ after each agglomeration step. Although each recalculation can be performed in constant time, and although it is not necessary to consider all pairs of taxa (x, y) in order to find the one that minimizes this sum [20], the number of pairs to consider remains, in the worst case, $O(k^2)$ when k nodes are left to agglomerate. Thus, summing over k, $O(n^3)$ computations are required in all.

Elias and Lagergren take a related approach to agglomeration, which does not exhaustively seek the minimum value of $S_D(x, y)$ at each step, but instead uses a heuristic to maintain a list of candidates of "visible pairs" (x, y) for agglomeration. At the $(n - k)$th step, when two neighbors are agglomerated from a k-taxa tree to form a $(k - 1)$-taxa tree, FNJ has a list of $O(k)$ visible pairs for which $S_D(x, y)$ is calculated. The pair joined is selected from this list. By trimming the number of pairs considered, Elias and Lagergren achieved an algorithm which requires only $O(n^2)$ computations. Other similar improvements to neighbor joining have also been proposed in recent years [8, 13, 20].

Safety Radius Performance Analysis (Atteson [1])

In order to provide accuracy guarantees for distance-based algorithms, Atteson [1] tackled the following question: if D is a distance matrix that approximates a tree metric D_T, can one have some confidence in the algorithm's ability to reconstruct T, or parts of it, given D, based on some measure of the distance between D and D_T? For two matrices, D_1 and D_2, the L_∞ distance between them is defined by $||D_1 - D_2||_\infty = \max_{i,j} |D_1(i, j) - D_2(i, j)|$. Moreover, let $\mu(T)$ denote the length of the shortest internal edge of a tree T. This is an important quantity, as short branches in a phylogeny are difficult to resolve, because of the relatively few (if any) molecular changes occurring on a short branch.

The *safety radius* of an algorithm A is then the greatest value of r with the property that given any phylogeny T, and any distance matrix D satisfying $||D - D_T||_\infty < r \cdot \mu(T)$, A will return a tree $\hat{T}$ with the same topology as T. Similarly, the *edge radius* of A is the greatest value of r, for which the presence in $\hat{T}$ of an edge $e \in E$ is guaranteed whenever $||D - D_T||_\infty < r \cdot l(e)$. As an easy consequence of these definitions, the safety radius is always at least as large as the edge radius. Moreover, both the safety radius and the edge radius can also be attributed to an optimization principle, assuming an exact optimization algorithm.

Key Results

Atteson [1] proved the following theorems:

Theorem 1 *The safety radius of NJ is* $1/2$.

Theorem 2 *The largest possible safety radius for any algorithm is* $1/2$.

Indeed, given any μ, one can find two different trees T_1, T_2 and a distance matrix D such that $\mu = \mu(T_1) = \mu(T_2)$, and $\|D - D_{T_1}\|_\infty = \mu/2 = \|D - D_{T_2}\|_\infty$. Since D is equidistant from two distinct tree metrics, no algorithm could assign it to the "closest" tree.

In their presentation of FNJ, Elias and Lagergren updated Atteson's results for their algorithm. They showed:

Theorem 3 *The safety radius of FNJ is* $1/2$.

An insight on the above results on neighbor-joining-type algorithms is provided by the fact that the optimization principle they are linked to, BME, has itself safety radius $1/2$ [15]. A simple consequence of this [15] is the fact that also GreedyBME has safety radius $1/2$, a result first proven by Shigezumi [19]. Finally, performing a local search guided by BME and based on SPR leads to an algorithm with safety radius greater or equal to $1/3$, regardless of the method used to construct the initial tree [2].

The edge radius of a number of algorithms has also been studied. As conjectured by Atteson [1] and formally proven by Dai et al. [5], the edge radius of NJ is $1/4$. Interestingly, other heuristics, related to NJ via the principle they seek to optimize (BME), perform better than NJ in terms of edge radius: GreedyBME has edge radius $1/3$ [3]; moreover, building an initial tree with GreedyBME and then performing a local search guided by BME and based on NNI or SPR operations constitute an algorithm with edge radius $1/3$ [3].

Finally, we note that the safety radius framework has also been applied to the ultrametric setting where the correct tree T is rooted and all tree leaves are at the same distance from the root [10]. These trees are called "molecular clock" trees in phylogenetics and "indexed hierarchies" in data analysis. In this setting, the optimal safety radius is equal to 1 (instead of $1/2$), and a number of standard algorithms (e.g., UPGMA, with time complexity in $O(n^2)$) have a safety radius of 1.

Open Problems

With increasing amounts of sequence data becoming available for an increasing number of species, distance algorithms such as NJ should be useful for quite some time. Currently, the bottleneck in the process of building phylogenies is estimating distances, rather than exploring tree topologies. Two algorithms were recently developed to reconstruct trees from incomplete distance matrices. These algorithms use character information as well as distances and hence cannot be categorized as pure distance methods.

FastTree [17] is an NJ-like heuristic that avoids computing the full distance matrix. For each taxon, FastTree computes the distances to $O(\sqrt{n})$ close neighbors. FastTree also uses sequence profiles to approximate $S_D(x, y)$ values in constant time. The overall algorithm takes $O(san\sqrt{n}\log n)$ time and $O(san + n\sqrt{n})$ memory, where s is the length of the input sequences and a is their alphabet size. FastTree has been shown to be highly accurate with simulated data [17], but no formal guarantee has yet been shown for this algorithm.

The only known $o(n^2)$ algorithm with theoretical guarantees is LSHTree [4]. It uses locality-sensitive hashing to rapidly find candidate pairs of close sequences for merging. After each merge, LSHTree reconstructs ancestral sequences at new internal nodes to ensure that a close pair of sequences can be found at each iteration. LSHTree is guaranteed to reconstruct the correct tree from sequences of logarithmic length under a Markov model of sequence evolution. The exact running time of LSHTree depends on the branch lengths.

As we have shown, a number of distance-based tree building algorithms have been analyzed in the safety radius framework. However,

computer simulations (e.g., [6, 7]) have shown that not all algorithms with optimal safety radius achieve the same accuracy: for example, NJ is slightly more accurate than FNJ (both having safety radius $= 1/2$), but is beaten by heuristics based on NNI or SPR moves (with demonstrated safety radius $>= 1/3$, but possibly $= 1/2$). Moreover, some well-established methods (e.g., based on least squares [10, 21]) have safety radius converging to 0 when the number of taxa increases, which contradicts the common practice. These experimental observations indicate that the safety radius approach should be sharpened to provide better theoretical analysis of method performance (see [12] for a work in this direction). In particular, the choice of the L_∞ norm to measure the error in a distance matrix seems to have little statistical or biological justification.

An alternative analysis framework, strictly linked to the one presented here, is the one seeking to estimate the minimum sequence length required for accurate reconstruction of the correct tree. It is discussed in a separate entry of this encyclopedia [A].

Cross-References

▸ Distance-Based Phylogeny Reconstruction (Fast-Converging)

Recommended Reading

1. Atteson K (1999) The performance of neighbor-joining methods of phylogenetic reconstruction. Algorithmica 25:251–278
2. Bordewich M, Gascuel O, Huber KT, Moulton V (2009) Consistency of topological moves based on the balanced minimum evolution principle of phylogenetic inference. IEEE/ACM Trans Comput Biol Bioinformatics 6:110–117
3. Bordewich M, Mihaescu R (2013) Accuracy guarantees for phylogeny reconstruction algorithms based on balanced minimum evolution. IEEE/ACM Trans Comput Biol Bioinformatics 10:576–583
4. Brown DG, Truszkowski J (2012) Fast phylogenetic tree reconstruction using locality-sensitive hashing. Algorithms in bioinformatics. Springer, Berlin/Heidelberg, pp 14–29
5. Dai W, Xu Y, Zhu B (2006) On the edge l∞ radius of Saitou and Nei's method for phylogenetic reconstruction. Theor Comput Sci 369:448–455
6. Desper R, Gascuel O (2002) Fast and accurate phylogeny reconstruction algorithms based on the minimum- evolution principle. J Comput Biol 9:687–706
7. Elias I, Lagergren J (2005) Fast neighbor joining. In: Proceedings of the 32nd international colloquium on automata, languages, and programming (ICALP), Lisbon, pp 1263–1274
8. Evans J, Sheneman L, Foster J (2006) Relaxed neighbor joining: a fast distance-based phylogenetic tree construction method. J Mol Evol 62:785–792
9. Felsenstein J (2004) Inferring phylogenies. Sinauer Associates, Sunderland
10. Gascuel O, McKenzie A (2004) Performance analysis of hierarchical clustering algorithms. J Classif 21:3–18
11. Gascuel O, Steel M (2006) Neighbor-joining revealed. Mol Biol Evol 23:1997–2000
12. Gascuel O, Steel M (2014) A 'stochastic safety radius' for distance-based tree reconstruction. Algorithmica 1–18. http://dx.doi.org/10.1007/s00453-015-0005-y
13. Gronau I, Moran S (2007) Neighbor joining algorithms for inferring phylogenies via LCA distances. J Comput Biol 14:1–15
14. Hein J (1989) An optimal algorithm to reconstruct trees from additive distance data. Bull Math Biol 51:597–603
15. Pardi F, Guillemot S, Gascuel O (2010) Robustness of phylogenetic inference based on minimum evolution. Bull Math Biol 72:1820–1839
16. Pauplin Y (2000) Direct calculation of a tree length using a distance matrix. J Mol Evol 51:41–47
17. Price MN, Dehal PS, Arkin AP (2009) FastTree: computing large minimum evolution trees with profiles instead of a distance matrix. Mol Biol Evol 26:1641–1650
18. Saitou N, Nei M (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol Biol Evol 4:406–425
19. Shigezumi T (2006) Robustness of greedy type minimum evolution algorithms. In: Computational science–ICCS, Reading, pp 815–821
20. Simonsen M, Mailund T, Pedersen CNS (2011) Inference of large phylogenies using neighbour-joining. In: Fred A, Felipe J, Gamboa H (eds) Biomedical engineering systems and technologies. Communications in computer and information science, vol 127. Springer, Berlin/Heidelberg, pp 334–344
21. Willson S (2005) Minimum evolution using ordinary least-squares is less robust than neighbor-joining. Bull Math Biol 67:261–279
22. Zarestkii K (1965) Reconstructing a tree from the distances between its leaves. (In Russian) Usp Math Nauk 20:90–92

Distributed Algorithms for Minimum Spanning Trees

Sergio Rajsbaum
Instituto de Matemáticas, Universidad Nacional Autónoma de México (UNAM), México City, México

Keywords

Minimum weight spanning tree

Years and Authors of Summarized Original Work

1983; Gallager RG, Humblet PA, Spira PM

Problem Definition

Consider a communication network, modeled by an undirected weighted graph $G = (V, E)$, where $|V| = n$, $|E| = m$. Each vertex of V represents a processor of unlimited computational power; the processors have unique identity numbers (ids), and they communicate via the edges of E by sending messages to each other. Also, each edge $e \in E$ has associated a weight $w(e)$, known to the processors at the endpoints of e. Thus, a processor knows which edges are incident to it and their weights, but it does not know any other information about G. The network is *asynchronous:* each processor runs at an arbitrary speed, which is independent of the speed of other processors. A processor may wake up spontaneously or when it receives a message from another processor. There are no failures in the network. Each message sent arrives at its destination within a finite but arbitrary delay. A *distributed algorithm* A for G is a set of local algorithms, one for each processor of G, that include instructions for sending and receiving messages along the edges of the network. Assuming that A terminates (i.e., all the local algorithms eventually terminate), its *message complexity* is the total number of messages sent over any execution of the algorithm, in the worst case. Its *time complexity* is the worst-case execution time, assuming processor steps take negligible time, and message delays are normalized to be at most 1 unit.

A *minimum spanning tree* (MST) of G is a subset E' of E such that the graph $T = (V, E')$ is a tree (connected and acyclic) and its total weight, $w(E') = \sum_{e \in E'} w(e)$, is as small as possible. The computation of an MST is a central problem in combinatorial optimization, with a rich history dating back to 1926 [2], and up to now, the book [12] collects properties, classical results, applications, and recent research developments.

In the *distributed MST problem*, the goal is to design a distributed algorithm A that terminates always and computes an MST T of G. At the end of an execution, each processor knows which of its incident edges belong to the tree T and which do not (i.e., the processor writes in a local output register the corresponding incident edges). It is remarkable that in the distributed version of the MST problem, a communication network is solving a problem where the input is the network itself. This is one of the fundamental starting points of network algorithms.

It is not hard to see that if all edge weights are different, the MST is unique. Due to the assumption that processors have unique ids, it is possible to assume that all edge weights are different: whenever two edge weights are equal, ties are broken using the processor ids of the edge endpoints. Having a unique MST facilitates the design of distributed algorithms, as processors can locally select edges that belong to the unique MST. Notice that if processors do not have unique ids and edge weights are not different, there is no deterministic MST (nor any spanning tree) distributed algorithm, because it may be impossible to break the symmetry of the graph, for example, in the case it is a cycle with all edge weights equal.

Key Results

The distributed MST problem has been studied since 1977, and dozens of papers have been

written on the subject. In 1983, the fundamental distributed *GHS algorithm* in [5] was published, the first to solve the MST problem with $O(m + n \log n)$ message complexity. The paper has had a very significant impact on research in distributed computing and won the 2004 Edsger W. Dijkstra Prize in Distributed Computing.

It is not hard to see that any distributed MST algorithm must have $\Omega(m)$ message complexity (intuitively, at least one message must traverse each edge). Also, results in [3, 4] imply an $\Omega(n \log n)$ message complexity lower bound for the problem. Thus, the GHS algorithm is optimal in terms of message complexity.

The $\Omega(m + n \log n)$ message complexity lower bound for the construction of an MST applies also to the problem of finding an arbitrary spanning tree of the graph. However, for specific graph topologies, it may be easier to find an arbitrary spanning tree than to find an MST. In the case of a complete graph, $\Omega(n^2)$ messages are necessary to construct an MST [8], while an arbitrary spanning tree can be constructed in $O(n \log n)$ messages [7].

The time complexity of the GHS algorithm is $O(n \log n)$. In [1] it is described how to improve its time complexity to $O(n)$ while keeping the optimal $O(m + n \log n)$ message complexity. It is clear that $\Omega(D)$ time is necessary for the construction of a spanning tree, where D is the diameter of the graph. And in the case of an MST, the time complexity may depend on other parameters of the graph. For example, due to the need for information flow among processors residing on a common cycle, as in an MST construction, at least one edge of the cycle must be excluded from the MST. If messages of unbounded size are allowed, an MST can be easily constructed in $O(D)$ time, by collecting the graph topology and edge weights in a root processor. The problem becomes interesting in the more realistic model where messages are of size $O(\log n)$ and an edge weight can be sent in a single message. When the number of messages is not important, one can assume without loss of generality that the model is synchronous. For near-time optimal algorithms and lower bounds, see [10] and references herein.

Applications

The distributed MST problem is important to solve, both theoretically and practically, as an MST can be used to save on communication, in various tasks such as broadcast and leader election, by sending the messages of such applications over the edges of the MST.

Also, research on the MST problem, and in particular the MST algorithm of [5], has motivated a lot of work. Most notably, the algorithm of [5] introduced various techniques that have been in widespread use for multicasting, query and reply, cluster coordination and routing, protocols for handshake, synchronization, and distributed phases. Although the algorithm is intuitive and is easy to comprehend, it is sufficiently complicated and interesting that it has become a challenge problem for formal verification methods, e.g., [11].

Open Problems

There are many open problems in this area, and only a few significant ones are mentioned. As far as message complexity, although the asymptotically tight bound of $O(m + n \log n)$ for the MST problem in general graphs is known, finding the actual constants remains an open problem. There are smaller constants known for general spanning trees than for MST though [6].

As mentioned above, near-time optimal algorithms and lower bounds appear in [10] and references herein. The optimal time complexity remains an open problem. Also, in a synchronous model for overlay networks, where all processors are directly connected to each other, an MST can be constructed in sublogarithmic time, namely, $O(\log \log n)$ communication rounds [9], and no corresponding lower bound is known.

Cross-References

▸ Synchronizers, Spanners

Recommended Reading

1. Awerbuch B (1987) Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems (detailed summary). In: Proceedings of the 19th annual ACM symposium on theory of computing, New York City. ACM, New York, pp 230–240
2. Borůvka O (2001) Otakar Borůvka on minimum spanning tree problem (translation of both the 1926 papers, comments, history). Discret Math 233: 3–36
3. Burns JE (1980) A formal model for message-passing systems, TR-91. Indiana University, Bloomington
4. Frederickson G, Lynch N (1984) The impact of synchronous communication on the problem of electing a leader in a ring. In: Proceedings of the 16th annual ACM symposium on theory of computing, Washington, DC. ACM, New York, pp 493–503
5. Gallager RG, Humblet PA, Spira PM (1983) A distributed algorithm for minimum-weight spanning trees. ACM Trans Prog Lang Syst 5(1): 66–77
6. Johansen KE, Jorgensen UL, Nielsen SH (1987) A distributed spanning tree algorithm. In: Proceedings of the 2nd international workshop on distributed algorithms (DISC), Amsterdam. Lecture notes in computer science, vol 312. Springer, Berlin/Heidelberg, pp 1–12
7. Korach E, Moran S, Zaks S (1984) Tight upper and lower bounds for some distributed algorithms for a complete network of processors. In: Proceedings of the 3rd symposium on principles of distributed computing (PODC), Vancouver. ACM, New York, pp 199–207
8. Korach E, Moran S, Zaks S (1985) The optimality of distributive constructions of minimum weight and degree restricted spanning trees in a complete network of processors. In: Proceedings of the 4th symposium on principles of distributed computing (PODC), Minaki. ACM, New York, pp 277–286
9. Lotker Z, Patt-Shamir B, Pavlov E, Peleg D (2005) Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. SIAM J Comput 35(1):120–131
10. Lotker Z, Patt-Shamir B, Peleg D (2006) Distributed MST for constant diameter graphs. Distrib Comput 18(6):453–460
11. Moses Y, Shimony B (2006) A new proof of the GHS minimum spanning tree algorithm. In: 20th international symposium on distributed computing (DISC), Stockholm, 18–20 Sept 2006. Lecture notes in computer science, vol 4167. Springer, Berlin/Heidelberg, pp 120–135
12. Wu BY, Chao KM (2004) Spanning trees and optimization problems. Discrete mathematics and its applications. Chapman Hall, Boca Raton

Distributed Computing for Enumeration

Alexandre Termier
IRISA, University of Rennes, 1, Rennes, France

Keywords

Cluster; Enumeration; Multicore; Parallelism; Work sharing; Work stealing

Years and Authors of Summarized Original Work

2006; Buehrer, Parthasarathy, Chen
2012; Martins, Manquiho, Lynce
2014; Negrevergne, Termier, Rousset, Mehaut

Problem Definition

This entry considers enumeration of combinatorial problems, which can be formulated as follows. Given a large search space $\mathcal{C}$ and a predicate of interest $P : \mathcal{C} \mapsto \{\text{true, false}\}$ the goal is to enumerate the solutions $\mathcal{S} \subseteq \mathcal{C}$ such that $\forall ?s \in \mathcal{S} \quad P(s) = \text{true}$. In most settings, $\mathcal{C}$ is the complete set of combinations of an initial set $\mathcal{G}$; hence, $|\mathcal{C}| = 2^{|\mathcal{G}|}$ and the problem is NP-hard. There are also cases where the elements to enumerate are not sets but other combinatorial structures such as sequences or graphs.

We restrict ourselves to the case where $\mathcal{C}$ can be organized as an *enumeration tree*:

- There exists a distinguished element of $\mathcal{C}$ called the root
- There exists (at least) a *parent* function *parent*: $\mathcal{C} \setminus \{\text{root}\} \mapsto \mathcal{C}$

Finding $\mathcal{S}$ implies, in the worst case, to enumerate all elements of $\mathcal{C}$. A large body of research has exploited properties of P together with *parent* to avoid enumerating some parts of $\mathcal{C}$, as described in the ▸ Reverse Search; Enumeration

Algorithms entry. The focus of the current entry is to exploit parallel computing devices in order to speedup the enumeration process. Due to the prevalence of multicore processors nowadays, they will be the main focus of this entry. However, many solutions presented also apply to a cluster setting.

Key Results

The main challenges of distributed enumeration, as well as existing solutions, are presented below.

Synchronization

For most distributed algorithms, one important challenge is to ensure the sharing of information between parallel processes, either by message passing in a cluster setting or by access to shared memory locations in a multicore setting.

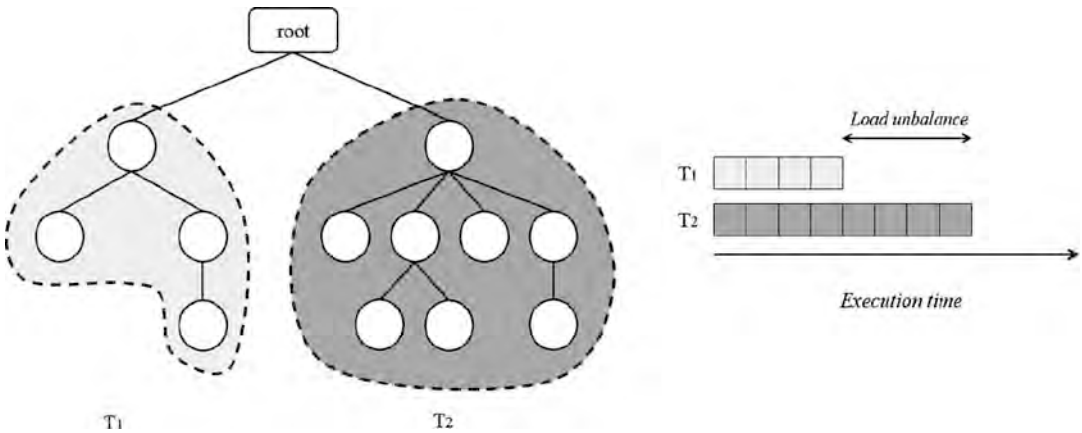
This entry considers a tree-shaped enumeration, where all branches of the enumeration tree are independent. In such setting, complex synchronization mechanisms are not needed. The main difficulty is thus to guarantee that such tree-shaped enumeration can take place, i.e., find (at least) one *parent* function (cf. ▸ Reverse Search; Enumeration Algorithms entry).

Note that in some parallel SAT solvers [6] using *portfolio*-based approaches, each computing resource exploits a different strategy (and thus a different *parent* function) to explore the search space. Ideally these strategies are orthogonal: they explore different parts of the space, and they exchange information to help mutual pruning.

Load Balancing

Most recent algorithms explore the enumeration tree with Depth-First Search (DFS). The simplest way to perform distributed enumeration is to partition the enumeration tree and assign each subtree to a parallel process. A simple example is shown in Fig. 1. The figure shows the enumeration tree. Each subtree of the root is assigned to a different computing resource (either to a node in a cluster or to a core of a multicore processor).

However, depending on the choice of *parent* function and of pruning strategies, each subtree is likely to be of a different size, resulting on large running time differences for the parallel processes. Such phenomenon is called *load unbalance*. On the right of Fig. 1, the execution time is represented for both computing resources, assuming that computing for any node of the enumeration tree takes exactly one time unit. Computing resource T_1 computes for four time units, while computing resource T_2 computes for eight time units. This means that T_1 has been idle for four time units: for more than half of the execution time, the execution has only exploited



Distributed Computing for Enumeration, Fig. 1 Partitioning of the enumeration tree

one of the two available computing resources, resulting in a longer execution time (in an ideal case, as there are 12 nodes to explore, the execution time should be 6 time units). The solution to this problem is to use dynamic load balancing strategies such as *work sharing* or *work stealing*.

Work sharing is based on a simple producer/consumer principle: tasks (here nodes of the enumeration tree) are not assigned statically but made available in a single pool. Each idle computing resource can request one or more tasks from this pool. By reducing the granularity of the work each computing resource has to perform, this technique effectively reduces load unbalance. A disadvantage of this technique is that if computation for a single node of the enumeration tree is short, the computing resource will make frequent requests to the central work pool, with an increased risk of synchronization overheads over accesses to the pool. This can be limited by passing more than one enumeration tree node to each computing resource [1]; however, if the computation time of each of these nodes is unpredictable, a new load unbalance situation may arise.

Work stealing [2] is an "optimistic" improvement over work sharing in cases where the computation time of each task is unpredictable (which is often the case in distributed enumeration). The idea is that each computing resource enqueues a list of tasks to perform, which can either come from a central pool (as in work sharing) or from a partition of the search space (as in a static partitioning). When a computing resource finishes its tasks and becomes idle, it will query the work queues of the other resources and "steal" part of a nonempty queue (usually from the biggest queue).

Recent works from Hanusse et al. [3] have proposed an enumeration approach that mixes DFS and BFS. The parallelism that takes place in the BFS steps naturally lead to a good load balancing without needing work-stealing mechanisms. Their approach show promising results.

Data Locality

In some classes of applications, such as pattern mining, testing the predicate P requires to access a large dataset $\mathcal{D}$. The enumeration tree is organized such that each branch explores a subset of the dataset, which allows to benefit from data locality effects and efficiently exploit the cache in case of multicore processors.

Techniques to limit load unbalance presented above tend to dispatch nodes from the same branch to different computing resources, which destroys data locality. This can lead to a vastly increased bandwidth usage (memory bus bandwidth for multicores and network bandwidth in clusters), resulting in a severe loss of performance.

Several solutions have been proposed by the pattern mining community in order to combine good load balance and good data locality. These solutions are designed for multicore processors. In the case of work stealing, Buehrer et al. [4] propose a method which dynamically decides, for a given node of the enumeration tree, if it has to be mined by the same thread as its parent (preserving locality) or if it can be enqueued and possibly stolen by another thread. This method takes into account the system load, which is a function of the size of the queues for the threads.

In the case of work sharing, Négrevergne et al. [5] divide the task pool in one queue per thread. Task assignement to threads prioritizes data locality.

Applications

The main applications of distributed enumeration are pattern mining (a field of data mining) and SAT problem solving [6]. Pattern mining consists in finding regularities in data, whereas solving SAT consists in finding if there exists a truth assignment for variables of a propositional formula. Both problems explore a huge search space and have numerous applications, they thus require to exploit as much as possible available parallel computing power.

Open Problems

Having an optimal parallel scaling for applications using distributed enumeration is still an

open problem. The techniques presented in this entry allow to design algorithms having satisfying results on existing multicore computers with tens of cores. However as the number of cores will grow towards *many-core processors* (hundreds of cores), the current algorithms are unlikely to exhibit a good parallel scalability. Novel approaches with fewer overheads for handling the parallelism will be required.

Cross-References

▸ Reverse Search; Enumeration Algorithms

Recommended Reading

1. Zaki MJ (1999) Parallel and distributed association mining: a survey. IEEE Concurr 7(4):14–25
2. Blumofe RD, Leiserson CE (1999) Scheduling multithreaded computations by work stealing. J ACM 46(5):720–748
3. Hanusse N, Maabout S (2011) A parallel algorithm for computing borders. In: Proceedings of the 20th conference on information and knowledge management (CIKM'11), Glasgow, pp 1639–1648
4. Buehrer G, Parthasarathy S, Chen Y-K (2006) Adaptive parallel graph mining for CMP architectures. In: Proceedings of the 6th IEEE international conference on data mining (ICDM'06), Hong Kong, pp 97–106
5. Négrevergne B, Termier A, Rousset M-C, Méhaut J-F (2014) ParaMiner: a generic pattern mining algorithm for mutli-core architectures. Data Min Knowl Discov 28(3):593–633
6. Martins R, Manquiho VM, Lynce I (2012) An overview of parallel SAT solving. Constraints 17(3):304–347

Distributed Randomized Broadcasting in Wireless Networks under the SINR Model

Tomasz Jurdziński[1] and Dariusz R. Kowalski[2]
[1]Institute of Computer Science, University of Wrocław, Wrocław, Poland
[2]Department of Computer Science, University of Liverpool, Liverpool, UK

Keywords

Ad hoc network; Broadcasting; Distributed algorithms; SINR model; Wireless network

Years and Authors of Summarized Original Work

2012; Yu, Hua, Wang, Tan, Lau
2013; Daum, Gilbert, Kuhn, Newport
2013; Jurdziński, Kowalski, Różański, Stachowiak
2013; Jurdziński, Kowalski, Stachowiak

Problem Definition

Broadcasting is a fundamental problem in communication networks, where one distinguished node, called the *source*, holds a piece of information, and the goal is to disseminate this message to all other nodes in the network.

The *signal-to-interference-and-noise-ratio model*, *SINR* for short, generalizes the abstract radio networks model (RN) in the following way: nodes located in a metric space communicate by transmitting a signal to the wireless medium, and the quantitative accumulation of interference and signal attenuation are taken into account when deciding which nodes successfully receive the signal.

In more detail, a wireless network consists of n *nodes*, deployed into the Euclidean plane; each node v has its *transmission power* P_v, which is a positive real number. A network is *uniform* when transmission powers P_v are equal or *nonuniform* otherwise. In the following, the uniform networks are considered.

Nodes work synchronously in rounds; each node can either act as a transmitter or as a receiver during a round.

Interferences and collisions are determined by three fixed model parameters: path loss $\alpha > 2$, threshold $\beta \geq 1$, and ambient noise $\mathcal{N} > 0$. The $\mathrm{SINR}(v, u, \mathcal{T})$ ratio, for given nodes u, v and a set of transmitting nodes $\mathcal{T}$, is defined as

$$\mathrm{SINR}(v,u,\mathcal{T}) = \frac{P_v \mathrm{dist}(v,u)^{-\alpha}}{\mathcal{N} + \sum_{w \in \mathcal{T}\setminus\{v\}} P_w \mathrm{dist}(w,u)^{-\alpha}},$$

where $\mathrm{dist}(\cdot,\cdot)$ is the distance function on the plane. A node u successfully receives a message from a node v in a round if $v \in \mathcal{T}$, $u \notin \mathcal{T}$, and

$$\mathrm{SINR}(v, u, \mathcal{T}) \geq \beta, \tag{1}$$

where $\mathcal{T}$ is the set of nodes transmitting at that round.

A single message sent in an execution of any algorithm can carry the source message and at most logarithmic, in the size of the network, number of control bits. A node other than the source starts executing the broadcast protocol after the first successful receipt of the source message; it is often called a *non-spontaneous wake-up model*.

In an *ad hoc network*, there is no central knowledge of network topology. A node v participating in an execution of a protocol knows the size of a network n or only a linear upper bound on the size. In the case that a network is deployed in the Euclidean space, one can distinguish between the case that each node knows its coordinates and the case that it does not know them.

Assuming that the transmission power of nodes is equal to P, the largest distance from the transmitter in which a message can be received is equal to $r = (P/(\mathcal{N}\beta))^{1/\alpha}$, provided only one node is transmitting in the whole network.

Sensitivity. Due to physical constraints, it is often assumed that the actual distance on which message can be received is smaller than $r = (P/(\mathcal{N}\beta))^{1/\alpha}$. This assumption is expressed by the *sensitivity parameter* $0 < \varepsilon_s < 1$ such that a message transmitted by v is received at a node u in a round with the set of transmitters $\mathcal{T}$ if $\mathrm{SINR}(v, u, \mathcal{T}) \geq \beta$ and $\mathrm{dist}(v, u) \leq (1 - \varepsilon_s)r$.

The setting with $\varepsilon_s = 0$ is called the model with *strong sensitivity*, and $\varepsilon_s > 0$ defines the model with *weak sensitivity*.

Connectivity. In order to determine which nodes are connected in a network, the notion of a *communication graph* is introduced. To this aim, the *connectivity parameter* ε_c is introduced such that $1 > \varepsilon_c \geq \varepsilon_s \geq 0$. An edge (u, v) appears in the communication graph iff $\mathrm{dist}(u, v) \leq (1 - \varepsilon_c)r$. The setting with $\varepsilon_s = \varepsilon_c$ is called the model with *weak connectivity*, and the inequality $\varepsilon_s < \varepsilon_c$ defines the model with *strong connectivity*.

Time complexity of a randomized broadcasting algorithm is the number of rounds after which, for all communication networks defined by given SINR parameters α, β, and $\mathcal{N}$, and the parameters ε_s, ε_c, the source message is delivered to all nodes accessible from the source node in the communication graph with high probability (whp), i.e., with the probability at least $1 - 1/n$, where n is the number of nodes in the network.

Key Results

Complexity of the broadcasting problem significantly differs in various models obtained by constraints imposed on the sensitivity and connectivity parameters.

The Model with Strong Sensitivity and Strong Connectivity

In this setting reception of messages is determined merely by Eq. (1), and the communication graph does not contain links of distance close to r due to $\varepsilon_c > 0$.

Theorem 1 ([3]) *The broadcasting problem can be solved in time* $O\left(D \log n + \log^2 n\right)$ *with high probability in the model with strong connectivity and strong sensitivity for networks in Euclidean two-dimensional space with known coordinates.*

For the setting that nodes do not know their coordinates, Daum et al. [2] provided a broadcasting algorithm which relies on a parameter R_s equal to the maximum ratio among distances between pairs of nodes connected by an edge in the communication graph. That is, $R_s = \max\{\mathrm{dist}(u, v)/\mathrm{dist}(x, y) \mid (u, v), (x, y) \in E\}$, where $G(V, E)$ is the communication graph of a network.

Theorem 2 ([2]) *The broadcasting problem can be solved in time* $O\left(D \log n \log^{\alpha+1} R_s\right)$ *with high probability in the model with strong connectivity and strong sensitivity.*

As R_s might be even exponential with respect to n, the solution from Theorem 2 is inefficient for some networks. The following theorem gives a

solution independent of geometric properties of a network.

Theorem 3 ([6]) *The broadcasting problem can be solved in time* $O\left(D\log^2 n\right)$ *with high probability in the model with strong sensitivity and strong connectivity.*

The Model with Strong Sensitivity and Weak Connectivity

In this setting reception of messages is determined merely by Eq. (1), and an edge (u, v) belongs to the communication graph iff $\mathrm{SINR}(u, v, \emptyset) \geq \beta$.

Theorem 4 ([2]) *There exist families of networks with diameter* 2 *in the model with strong sensitivity and weak connectivity in which each broadcasting algorithm requires* $\Omega(n)$ *rounds to accomplish broadcast.*

Theorem 5 ([2]) *The broadcasting problem can be solved in time* $O(n\log^2 n)$ *with high probability in the model with strong sensitivity and weak connectivity.*

The Model with Weak Sensitivity and Strong Connectivity

In this setting, transmissions on unreliable links (i.e., on distance very close to r) are "filtered out." Moreover, the communication graph connects only nodes in distance at most $(1-\varepsilon_c)r$, which is strictly smaller than r.

Theorem 6 ([7]) *The broadcasting problem can be solved in time* $O\left(D+\log^2 n\right)$ *with high probability in the model with weak sensitivity and strong connectivity for networks deployed on the Euclidean plane, provided* $0 < \varepsilon_s < \varepsilon_c = 2/3$. *This solution works in the model with* ***spontaneous wake-up*** *and requires* ***power control mechanism****.*

When applied directly to the case of non-spontaneous wake-up, the algorithm from [7] gives time bound $O\left(D\log^2 n\right)$.

Corollary 1 *The broadcasting problem can be solved in time* $O\left(D\log^2 n\right)$ *with high probability in the model with weak sensitivity and strong connectivity for networks deployed on the Euclidean plane, provided* $0 < \varepsilon_s < \varepsilon_c = 2/3$. *This solution works in the model with* ***non-spontaneous wake-up*** *and requires* ***power control mechanism****.*

The Model with Weak Sensitivity and Weak Connectivity

In this setting, the maximal distances for a successful transmission and for connecting nodes by an edge are equal and both smaller than the largest theoretically possible range r following from Eq. (1).

Theorem 7 ([5]) *There exist (i) an infinite family of* n*-node networks requiring* $\Omega(n\log n)$ *rounds to accomplish broadcast whp and, (ii) for every* $D, \Delta = O(n)$*, an infinite family of* n*-node networks of diameter* D *and maximum degree of the communication graph* Δ *requiring* $\Omega(D\Delta)$ *rounds to accomplish broadcast whp in the model with weak sensitivity and weak connectivity (i.e.,* $0 < \varepsilon_s = \varepsilon_c < 1$*) for networks on the plane.*

Using appropriate combinatorial structures, deterministic broadcasting algorithms were obtained with complexities close to the above lower bounds, provided nodes know their coordinates.

Theorem 8 ([5]) *The broadcasting problem can be solved deterministically in time* $O\left(\min\left\{D\Delta\log^2 N, n\log N\right\}\right)$ *in the model with weak sensitivity and weak connectivity (i.e.,* $0 < \varepsilon_s = \varepsilon_c < 1$*) for networks in Euclidean two-dimensional space with known coordinates, with IDs in the range* $[1, N]$.

The above result translates to a randomized algorithm with complexity $O\left(\min\left\{D\Delta\log^2 n, n\log n\right\}\right)$, since nodes can choose unique IDs in the polynomial range with high probability.

Recently, Chlebus et al. [1] provided a randomized algorithm for the setting without knowledge of coordinates.

Theorem 9 ([1]) *The broadcasting problem can be solved in time* $O\left(n\log^2 n\right)$ *with high probability in the model with weak sensitivity and weak connectivity.*

Distributed Randomized Broadcasting in Wireless Networks under the SINR Model, Table 1 Complexity of randomized broadcasting with non-spontaneous wake-up for various sensitivity and connectivity settings. The result from [7] marked with $\star$ requires power control mechanism and $\varepsilon_c = 2/3$. The positive results requiring knowledge of coordinates apply only to the Euclidean space

	Coordinates	Strong connectivity: $\varepsilon_c > \varepsilon_s$	Weak connectivity: $\varepsilon_c = \varepsilon_s$
Strong sensitivity: $\varepsilon_s = 0$	Known	$O\left(D \log n + \log^2 n\right)$	$\Omega(n)$
	Unknown	$O\left(D \log^2 n\right), O\left(D \log n \log^{\alpha+1} R_s\right)$	$O\left(n \log^2 n\right)$
Weak sensitivity: $\varepsilon_s > 0$	Known		$\Omega\left(\min\{D\Delta, n\}\right)$ $O\left(\min\left\{D\Delta \log^2 n, n \log n\right\}\right)$
	Unknown	$O\left(D \log^2 n\right)$ $\star$	$O\left(n \log^2 n\right)$

Applications

Using similar techniques to those in [5], an efficient deterministic broadcasting algorithm was obtained in the model with strong connectivity and strong sensitivity.

Theorem 10 ([4]) *The broadcasting problem can be solved deterministically in time* $O\left(D \log^2 N\right)$ *in the model with strong connectivity and strong sensitivity for networks in Euclidean two-dimensional space with known coordinates, with IDs in the range* $[1, N]$.

The solution in [7] applies to a more general problem of multi-broadcast and was further generalized in [8] to the setting in which nodes wake up in various time steps.

Positive results from [1, 2, 6] work also in a more general setting when nodes are deployed in a bounded-growth metric space.

Open Problems

In all considered models, there is at least $\log n$ gap between the established lower and upper bounds for the complexity of broadcasting. A natural open problem is to tighten these bounds.

As seen in Table 1, it is not known whether the complexity of broadcasting depends on sensitivity for each connectivity setting.

Another interesting research direction is to explore the impact of additional features such as power control or carrier sensing on the complexity of the problem.

Cross-References

▸ Broadcasting in Geometric Radio Networks

Recommended Reading

1. Chlebus BS, Kowalski DR, Vaya S (2014) Distributed communication in bare-bones wireless networks, manuscript
2. Daum S, Gilbert S, Kuhn F, Newport CC (2013) Broadcast in the ad hoc SINR model. In: DISC, Jerusalem, Lecture notes in computer science, vol 8070. Springer, pp 358–372
3. Jurdzinski T, Kowalski DR, Rozanski M, Stachowiak G (2013) Distributed randomized broadcasting in wireless networks under the SINR model. In: DISC, Jerusalem, Lecture notes in computer science, vol 8070. Springer, pp 373–387
4. Jurdzinski T, Kowalski DR, Stachowiak G (2013) Distributed deterministic broadcasting in uniform-power ad hoc wireless networks. In: Gasieniec L, Wolter F (eds) FCT, Liverpool. Lecture notes in computer science, vol 8070. Springer, pp 195–209
5. Jurdzinski T, Kowalski DR, Stachowiak G (2013) Distributed deterministic broadcasting in wireless networks of weak devices. In: ICALP (2), Riga. Lecture notes in computer science, vol 7966. Springer, pp 632–644
6. Jurdzinski T, Kowalski DR, Rozanski M, Stachowiak G (2014) On the impact of geometry on ad hoc communication in wireless networks. In: Halldórsson MM, Dolev S (eds) PODC, Paris. ACM, pp 357–366
7. Yu D, Hua QS, Wang Y, Tan H, Lau FCM (2012) Distributed multiple-message broadcast in wireless ad-hoc networks under the SINR model. In: Even G, Halldórsson MM (eds) SIROCCO, Reykjavik. Lecture notes in computer science, vol 7355. Springer, pp 111–122
8. Yu D, Hua QS, Wang Y, Yu J, Lau FCM (2013) Efficient distributed multiple-message broadcasting in unstructured wireless networks. In: INFOCOM, Turin, pp 2427–2435

Distributed Snapshots

Michel Raynal
Institut Universitaire de France and IRISA,
Université de Rennes, Rennes, France

Keywords

Asynchronous message-passing system; Consistent checkpointing; Consistent global state; Lattice of global states; Stable property

Years and Authors of Summarized Original Work

1985; Chandy, Lamport

Preliminary Remark

The presentation of this entry of the Encyclopedia follows Chapter 6 of [15], to which it borrows the presentation style and all figures. The reader interested on this topic will find developments in [15].

The Notion of a Global State

Modeling the Execution of a Process: The Event Point of View

A distributed computation involving n asynchronous sequential processes $p_1, \ldots, p_n$, communicating by directed channels (hence, a directional channel can be represented by two directed channels). Channels can be FIFO (first in first out) or non-FIFO.

A distributed computation can be modeled by a (reflexive) partial order on the events produced by the processes. An event corresponds to the sending of a message, the reception of a message, or a nonempty sequence of operations which does not involve the sending or the reception of a message. This partial order, due to Lamport and called *happened before* relation [12], is defined as follows. Let e_1 and e_2 be two events; $e_1 \xrightarrow{ev} e_2$ is the smallest order relation such that:

- Process order: e_1 and e_2 are the same event or have been produced by the same process, and e_1 was produced before e_2.
- Message order: e_1 is the sending of a message m and e_2 is its reception.
- Transitive closure: There is an event e such that $e_1 \xrightarrow{ev} e \wedge e \xrightarrow{ev} e_2$.

Modeling the Execution of a Process: The Local State Point of View

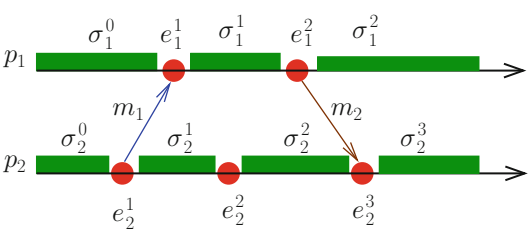
Let us consider a process p_i, which starts in the initial state σ_i^0. Let e_i^x be its xth event. The transition function $\delta_i()$ associated with p_i is consequently such that $\sigma_i^x = \delta_i(\sigma_i^x, e_i^x)$, where $x \geq 1$.

While a distributed computation can be modeled the partial order $\xrightarrow{ev}$ ("action" point of view), it follows from the previous definition that it can also be modeled by a partial order on its local states ("state" point of view). This partial order, denoted $\xrightarrow{\sigma}$, is defined as follows: $\sigma_i^x \xrightarrow{\sigma} \sigma_j^y \stackrel{\text{def}}{=} e_i^{x+1} \xrightarrow{ev} e_j^y$.

A two-process distributed execution is described in Fig. 1. The relation $\xrightarrow{ev}$ on event and the relation $\xrightarrow{\sigma}$ on local states can be easily extracted from it. As an example, we have $e_1^1 \xrightarrow{ev} e_2^3$ and $\sigma_2^0 \xrightarrow{\sigma} \sigma_1^2$. Two local states σ and σ' which are not related by $\xrightarrow{\sigma}$ are independent. This is denoted $\sigma || \sigma'$.

Orphan and In-Transit Messages

Let us consider an ordered pair of local states $\langle \sigma_i, \sigma_j \rangle$ from different processes p_i and p_j and a message m sent by p_i to p_j.



Distributed Snapshots, Fig. 1 A two-process distributed execution

- If m is sent by p_i after σ_i and received by p_j before σ_j, this message is *orphan* with respect to $\langle \sigma_i, \sigma_j \rangle$. This means that m is received and not sent with respect to the ordered pair $\langle \sigma_i, \sigma_j \rangle$.
- If m is sent by p_i before σ_i and received by p_j after σ_j, this message is *in-transit* with respect to $\langle \sigma_i, \sigma_j \rangle$. This means that m is sent and not yet received with respect to the ordered pair $\langle \sigma_i, \sigma_j \rangle$.

As an example, the message m_1 is orphan with respect to the ordered pair $\langle \sigma_2^0, \sigma_1^1 \rangle$, while the message m_2 is in-transit with respect to the ordered pair $\langle \sigma_1^2, \sigma_2^1 \rangle$.

Consistent Global State

A *global state* is a vector of n local states (one per process), plus a set of channel states (one per directed channel). The state of a channel is a sequence of messages if the channel is FIFO or a set of messages if the channel is non-FIFO.

A *consistent global state* (also called *snapshot*) is a global state in which the computation has passed or could have passed. More formally, a global state is a pair (Σ, M) where the vector of local states $\Sigma = [\sigma_1, \ldots, \sigma_n]$ and the set of channel states $M = \cup_{\{(i,j)\}} cs(i,j)$ are such that for any directed pair (i, j) we have:

- $\sigma_i || \sigma_j$. (This means that there is no orphan message with respect to the ordered pair $\langle \sigma_i, \sigma_j \rangle$.)
- $cs(i, j)$ contains all the messages which are in transit with respect to the ordered pair $\langle \sigma_i, \sigma_j \rangle$ and only them. (This means that $cs(i, j)$ contains all the messages (and only them) sent by p_i before σ_i and not yet received by p_j when it enters σ_j.)

As an example, when looking at Fig. 1, $\big([\sigma_1^2, \sigma_2^1], \{cs(1,2) = m_2,\ cs(2,1) = \emptyset\}\big)$ is a consistent global state, while both $\big([\sigma_1^1, \sigma_2^0], \{cs(1,2) = \ldots,\ cs(2,1) = \ldots\}\big)$ and $\big([\sigma_1^0, \sigma_2^2], \{cs(1,2) = \emptyset,\ cs(2,1) = \emptyset\}\big)$ are not consistent (the first because, as the message m_1 is orphan with respect to the ordered pair $\langle \sigma_1^1, \sigma_2^0 \rangle$, we do not have $\sigma_1^1 || \sigma_2^0$; the second because the message m_1 does not belong to the channel state $cs(2, 1)$.).

The Lattice of Global States

Let us consider a vector of local states $\Sigma = [\sigma_1, \ldots, \sigma_n]$ belonging to a consistent global state. The consistent global state, where $\Sigma' = [\sigma'_1, \ldots, \sigma'_n]$, is directly reachable from Σ if there is a process p_i whose next event e_i, and we have $\forall\ j \neq j :\ \sigma'_j = \sigma_j$ and $\sigma_i = \delta_i(\sigma_i, e_i)$. This is denoted $\Sigma \xrightarrow{GS} \Sigma'$. By definition $\Sigma \xrightarrow{GS} \Sigma$. More generally, $\Sigma \xrightarrow{GS} \Sigma_a \xrightarrow{GS} \Sigma_b \cdots \Sigma_y \xrightarrow{GS} \Sigma_z$ is denoted $\Sigma \xrightarrow{GS^*} \Sigma_z$.

It can be shown that the set of all the vectors Σ associated with the consistent global states produced by a distributed computation is a lattice [3, 16]. The lattice obtained from the computation of Fig. 1 is described in Fig. 2. In this lattice, the notation $\Sigma = [a, b]$ means $\Sigma = [\sigma_1^a, \sigma_2^b]$.

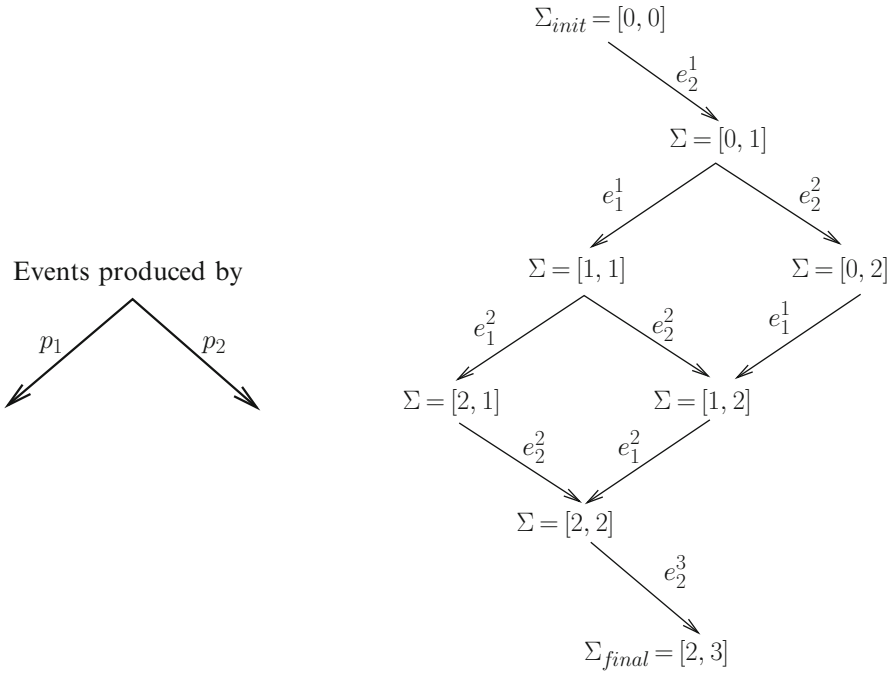
Problem Definition

Specification of the Computation of a Consistent Global State

The problem to determine on-the-fly a consistent global state (in short CGS) was introduced, precisely defined, and solved by Chandy and Lamport in 1985 [2]. It can be defined by the following properties. The first is liveness property, while the last two are safety properties.

- Termination. If one or more processes launch the computation of a consistent global state, then this global state computation terminates.
- Consistency. If a global state is computed, then it is consistent.
- Validity. Let Σ_{start} be the global state of the computation when CGS starts, Σ_{end} be its global state when CGS terminates, and Σ be the global state returned by CGS. We have $\Sigma_{\text{start}} \xrightarrow{GS^*} \Sigma$ and $\Sigma \xrightarrow{GS^*} \Sigma_{\text{end}}$.

The validity property states that the global state which is returned depends on the time at



Distributed Snapshots, Fig. 2 The lattice associated with the computation of Fig. 2

which its computation is launched. Without this property, returning always the initial global state would be a correct solution.

Principles of CGS Algorithms

To compute a consistent global state, each process p_i is in charge of (a) recording a copy of its local state σ_i (sometimes called its local snapshot) and (b) the states of its input (or output) channels. In order that the computed global state satisfies the safety properties, in one way or another, all CGS algorithms have two things to do.

- Synchronization. In order to ensure that there is no orphan messages with respect to each ordered pair of local states $\langle \sigma_i, \sigma_j \rangle$, such that there is a directed channel from p_i to p_j, the processes must synchronize the recording of their local states which will define the consistent global state that is computed.
- Message recording. Each process has to record all the messages it receives (or messages it sends) which are in transit with respect to the computed global state.

Key Result 1: Chandy-Lamport's Algorithm

Chandy and Lamport 's algorithm is denoted CL85 in the following.

Assumption

CL85 considers a failure-free asynchronous system. Asynchronous means that each process proceeds to its speed which can vary arbitrarily with time and remains always unknown to the other processes. Message transfer delays also are arbitrary (but finite).

CL85 assumes that the processes are connected by a directed communication graph, which is strongly connected (there is a directed path from any process to any other process). Each process has consequently a nonempty set of input channels and a nonempty set of output

channels. Moreover, each directed channel is a FIFO channel.

The Algorithm in Two Rules

CL85 requires that each process computes the state of its input channels. At a high abstraction level, it can formulated with two rules.

- "Local state recording" rule. When a process p_i records its local state σ_i, it sends a special control message (called *marker*) on each of its outgoing channels.

  It is important to notice that as channels are FIFO, a marker partitions the messages sent on a channel in two sets: the messages sent before the marker and the messages sent after the marker.
- "Input channel state recording" rule. When a process p_i receives a marker on one of its input channels $c(j,i)$, there are two cases.
  - If not yet done, it records its local state (i.e., it applies the first rule) and defines $cs(j,i)$ (the state of the input channel $c(j,i)$) as the empty sequence.
  - If it has already recorded its local state (i.e., executed the first rule), p_i defines $cs(j,i)$ as the sequence of application messages received on $c(j,i)$ between the recording of its local state and the reception of the marker on this input channel.

Properties of the Computed Global State

If one or more processes execute the first rule, it follows from this rule, and the fact that the communication graph is strongly connected, all the processes will execute this rule. Hence, a marker is sent on each directed channel, and each process records the state of its input channels. This proves the liveness property.

The consistency property follows from the following simple observation. Let us color processes and messages as follows. A process is initially green and becomes red when it sends a marker on each of its output channels. Moreover, a message has the color of its sender at the time the message is sent.

It is easy to see that the previous two rules guarantee that a green process turns red before receiving a red message (hence, there is no orphan messages). Moreover, the green messages received on a channel $c(j,i)$ by a red process p_i are the messages that are in transit with respect to the ordered pair $\langle \sigma_j, \sigma_i \rangle$. Hence, all the *in-transit* messages are recorded and only them.

The Inherent Uncertainty on the Computed Global State

The proof of the validity property is a little bit more involved. The interested reader will consult [2, 15]. When looking at the lattice of Fig. 2, let us consider that the CL85 algorithm is launched when the observed computation is in the global state $\Sigma_{\text{start}} = [0,1]$ and terminates when it is in the global state $\Sigma_{\text{end}} = [2,2]$. The consistency property states that the global state Σ which is returned is one of the following global states: $[0,1]$, $[1,1]$, $[0,2]$, $[2,1]$, $[1,2]$, or $[2,2]$.

This uncertainty on the computed global state is intrinsic to the nature of distributed computing. (Eliminate would require to freeze the execution of the application we are observing, which in some sense forces it to execute sequentially.)

The main property of the consistent global state Σ that is computed is that the application has passed through it or could have passed through it. While an external omniscient observer can know if the application passed or not through Σ, no process can know it. This noteworthy feature characterizes the relativistic nature of the observation of distributed computations.

Message-Passing Snapshot Versus Shared Memory Snapshot

The notion of a shared memory snapshot has been introduced in [1]. A snapshot object is an object that consists of an array of atomic multi-reader/single-write atomic registers, one per process (a process can read any register but can write only the register it is associated with). A snapshot object provides the processes with two operations denoted update() and snapshot(). The update() operation allows the invoking process to store a new value in its register, while the snapshot() operation allows it to obtain the values of all the atomic read/write registers as if that operation was executed instantaneously.

More precisely, the invocations of the operations update() and snapshot() are linearizable [8].

Differently from the snapshot values returned in a message-passing system (whose global structure is a lattice), the arrays of values returned by the snapshot() operations of a shared memory system can be totally ordered. This is a fundamental difference, which is related to the communication medium. In one case, the underlying shared memory is a centralized component, while in the second case, the underlying message-passing system is inherently distributed, making impossible to totally order all the message-passing snapshots.

Other Assumptions and Algorithms

Algorithms that compute consistent global states in systems equipped with non-FIFO channels have been designed. Such algorithms are described in [11, 13, 15].

A communication-induced (CI) algorithm is a distributed algorithm that does not use additional control messages (such as markers). In these algorithms, control information (if needed) has to be carried by application messages. CI algorithms that compute consistent global states have been investigated in [6].

Global states computation in large-scale distributed systems is addressed in [9].

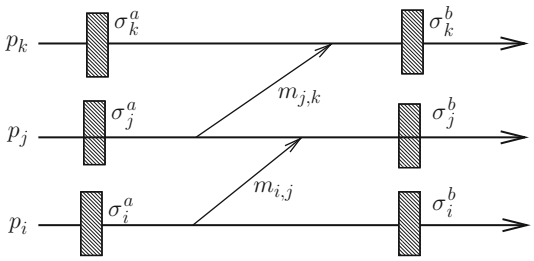
Key Result 2: A Necessary and Sufficient Condition

The Issue

An important question is the following one: Given a set of x, $1 \leq x \leq n$, local states from different processes do these local states belong to a consistent global state?

If $x = n$, the answer is easy: any ordered pair of local states $\langle \sigma_i, \sigma_j \rangle$ has to be such that $\sigma_i || \sigma_j$ (none of them causally depends on the other). Hence, the question is interesting when $1 \leq x < n$. This problem was addressed and solved by Netzer and Xu [14] and generalized in [7].

As a simple example, let us consider the execution in Fig. 3, where there are three processes p_i, p_j, and p_k that have recorded the local states



Distributed Snapshots, Fig. 3 A simple zigzag path

σ_x^a and σ_x^b, where $x \in \{i, j, k\}$. These local states produced by the computation are the only local states which have been recorded. Said another way, these recorded local states can be seen as local checkpoints.

An instance of the previous question is the following one: can the set $\{\sigma_i^a, \sigma_k^b\}$ be extended by the addition of a recorded local state σ_j of p_j (i.e., $\sigma_j = \sigma_j^a$ or $\sigma_j = \sigma_j^b$) such that the resulting global state $\Sigma = [\sigma_i^a, \sigma_j, \sigma_k^b]$ is a consistent?

It is easy to see that, despite the fact that the local states σ_i^a and σ_k^b are independent ($\sigma_i^a || \sigma_k^b$), neither $[\sigma_i^a, \sigma_j^a, \sigma_k^b]$ nor $[\sigma_i^a, \sigma_j^b, \sigma_k^b]$ is consistent. More precisely, $[\sigma_i^a, \sigma_j^a, \sigma_k^b]$ is not consistent because, due the message $m_{j,k}$, the local states σ_j^a and σ_k^b are not independent, while $[\sigma_i^a, \sigma_j^b, \sigma_k^b]$ is not consistent because, due the message $m_{i,j}$, σ_i^a and σ_j^b are not independent.

The Result

The notion of a *zigzag* path has been introduced by Netzer and Xu in [14]. An example of a simple zigzag path is the sequence of messages $\langle m_{i,j}, m_{j,k} \rangle$ of Fig. 3, where we see that the local states σ_i^a and σ_k^b are related by this zigzag path. A zigzag path captures hidden dependencies linking recorded local states. These dependencies are hidden in the sense that not all of them can be captured by the relation $\xrightarrow{\sigma}$ defined on local states (as shown in the figure).

The main result due to Netzer and Xu [14] is the following: a set x local states, with $1 \leq x < n$ and at most one local state per process, can be extended to a consistent global state if and only if no two of them are related by a zigzag path. This result has been extended in [7].

Applications: Global Snapshots in Action

Distributed snapshots are a key element to understand and master the uncertainty created by asynchrony. From a practical point of view, they are important in distributed checkpointing, in the detection of stable properties defined on the set of global states, and in the debugging of distributed programs.

Detection of Stable Properties

A stable property is a property that, one true, remains true forever. In the distributed context, examples of distributed stable properties are deadlock (once deadlocked, an application remains forever deadlocked), termination (once terminated, an application remains forever terminated) [4, 5], object inaccessibility, etc.

Algorithms that compute consistent global states satisfying the liveness, consistency, and validity properties previously stated can be used to detect stable properties. This follows from the observation that if the computed global state Σ satisfies a stable property P, then the global state Σ_{end} also satisfies P.

Checkpointing

A checkpoint is a global state from which a computation can be resumed. Trivially, checkpointing and consistent global states computation are problems which are very close [7]. The interested reader can consult [10, 15] for more details.

Cross-References

- Causal Order, Logical Clocks, State Machine Replication
- Distributed Computing for Enumeration
- Snapshots in Shared Memory

Recommended Reading

1. Afek Y, Attiya H, Dolev D, Gafni E, Merritt M, Shavit S (1993) Atomic snapshots of shared memory. J ACM 40(4):873–890
2. Chandy KM, Lamport L (1985) Distributed snapshots: determining global states of distributed systems. ACM Trans Comput Syst 3(1):63–75
3. Cooper R, Marzullo K (1991) Consistent detection of global predicates. In: Proceedings of the ACM/ONR workshop on parallel and distributed debugging, Santa Cruz. ACM, pp 163–173
4. Dijkstra EWD, Scholten CS (1980) Termination detection for diffusing computations. Inf Process Lett 11(1):1–4
5. Francez N (1980) Distributed termination. ACM Trans Program Lang Syst 2(1):42–55
6. Hélary J-M, Mostéfaoui A, Raynal M (1999) Communication-induced determination of consistent snapshots. IEEE Trans Parallel Distrib Syst 10(9):865–877
7. Hélary J-M, Netzer RHB, Raynal M (1999) Consistency criteria for distributed checkpoints. IEEE Trans Softw Eng 2(2):274–281
8. Herlihy MP, Wing JM (1990) Linearizability: a correctness condition for concurrent objects. ACM Trans Program Lang Syst 12(3):463–492
9. Kshemkalyani AD (2010) Fast and message-efficient global snapshot algorithms for large-scale distributed systems. IEEE Trans Parallel Distrib Syst 21(9):1281–1289
10. Kshemkalyani AD, Singhal M (2008) Distributed computing: principles, algorithms and systems. Cambridge University Press, Cambridge, 736p
11. Lai TH, Yang TH (1987) On distributed snapshots. Inf Process Lett 25:153–158
12. Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Commun ACM 21(7):558–565
13. Mattern F (1993) Efficient algorithms for distributed snapshots and global virtual time approximation. J Parallel Distrib Comput 18:423–434
14. Netzer RHB, Xu J (1995) Necessary and sufficient conditions for consistent global snapshots. IEEE Trans Parallel Distrib Syst 6(2):165–169
15. Raynal M (2013) Distributed algorithms for message-passing systems. Springer, 515p. ISBN:978-3-642-38122-5
16. Schwarz R, Mattern F (1994) Detecting causal relationships in distributed computations: in search of the Holy Grail. Distrib Comput 7(3):149–174

Distributed Vertex Coloring

Devdatt Dubhashi
Department of Computer Science, Chalmers University of Technology, Gothenburg, Sweden
Gothenburg University, Gothenburg, Sweden

Keywords

Distributed computation; Vertex coloring

Years and Authors of Summarized Original Work

2004; Finocchi, Panconesi, Silvestri

Problem Definition

The *vertex coloring problem* takes as input an undirected graph $G := (V, E)$ and computes a *vertex coloring*, i.e., a function, $c : V \to [k]$ for some positive integer k such that adjacent vertices are assigned different colors (that is, $c(u) \neq c(v)$ for all $(u, v) \in E$). In the $(\Delta + 1)$ vertex coloring problem, k is set equal to $\Delta + 1$ where Δ is the maximum degree of the input graph G. In general, $(\Delta + 1)$ colors could be necessary as the example of a clique shows. However, if the graph satisfies certain properties, it may be possible to find colorings with far fewer colors. Finding the minimum number of colors possible is a computationally hard problem: the corresponding decision problems are NP-complete [5]. In Brooks–Vizing colorings, the goal is to try to find colorings that are near optimal.

In this paper, the model of computation used is the synchronous, message passing framework as used in standard distributed computing [11]. The goal is then to describe very simple algorithms that can be implemented easily in this distributed model that simultaneously are *efficient* as measured by the number of rounds required and have good performance *quality* as measured by the number of colors used. For efficiency, the number of rounds is require to be poly-logarithmic in n, the number of vertices in the graph and for performance quality, the number of colors used is should be near-optimal.

Key Results

Key theoretical results related to distributed $(\Delta + 1)$-vertex coloring are due to Luby [9] and Johansson [7]. Both show how to compute a $(\Delta + 1)$-coloring in $O(\log n)$ rounds with high probability. For Brooks–Vizing colorings, Kim [8] showed that if the graph is square or triangle free, then it is possible to color it with $O(\Delta / \log \Delta)$ colors. If, moreover, the graph is regular of sufficiently high degree ($\Delta \gg \log n$), then Grable and Panconesi [6] show how to color it with $O(\Delta / \log \Delta)$ colors in $O(\log n)$ rounds. See [10] for a comprehensive discussion of probabilistic techniques to achieve such colorings.

The present paper makes a comprehensive experimental analysis of distributed vertex coloring algorithms of the kind analyzed in these papers on various classes of graphs. The results are reported in section "Experimental Results" below and the data sets used are described in section "Data Sets."

Applications

Vertex coloring is a basic primitive in many applications: classical applications are *scheduling problems* involving a number of pairwise restrictions on which jobs can be done simultaneously. For instance, in attempting to schedule classes at a university, two courses taught by the same faculty member cannot be scheduled for the same time slot. Similarly, two course that are required by the same group of students also should not conflict. The problem of determining the minimum number of time slots needed subject to these restrictions can be cast as a vertex coloring problem. One very active application for vertex coloring is *register allocation*. The register allocation problem is to assign variables to a limited number of hardware registers during program execution. Variables in registers can be accessed much quicker than those not in registers. Typically, however, there are far more variables than registers so it is necessary to assign multiple variables to registers. Variables conflict with each other if one is used both before and after the other within a short period of time (for instance, within a subroutine). The goal is to assign variables that do not conflict so as to minimize the use of non-register memory. A simple approach to this is to create a graph where the nodes represent variables and an edge represents conflict between its nodes. A coloring is then a conflict-free

assignment. If the number of colors used is less than the number of registers then a conflict-free register assignment is possible. Modern applications include *assigning frequencies* to mobile radios and other users of the electro-magnetic spectrum. In the simplest case, two customers that are sufficiently close must be assigned different frequencies, while those that are distant can share frequencies. The problem of minimizing the number of frequencies is then a vertex coloring problem. For more applications and references, see Michael Trick's coloring page [12].

Open Problems

The experimental analysis shows convincingly and rather surprisingly that the simplest, trivial, version of the algorithm actually performs best uniformly! In particular, it significantly outperforms the algorithms which have been analyzed rigorously. The authors give some heuristic recurrences that describe the performance of the trivial algorithm. It is a challenging and interesting open problem to give a rigorous justification of these recurrences. Alternatively, and less appealing, a rigorous argument that shows that the trivial algorithm dominates the ones analyzed by Luby and Johansson is called for. Other issues about how local structure of the graph impacts on the performance of such algorithms (which is hinted at in the paper) is worth subjecting to further experimental and theoretical analysis.

Experimental Results

All the algorithms analyzed start by assigning an initial *palette* of colors to each vertex, and then repeating the following simple iteration round:

1. *Wake up!*: Each vertex independently of the others wakes up with a certain probability to participate in the coloring in this round.
2. *Try!*: Each vertex independently of the others, selects a tentative color from its palette of colors at this round.
3. *Resolve conflicts!*: If no neighbor of a vertex selects the same tentative color, then this color becomes final. Such a vertex exits the algorithm, and the remaining vertices update their palettes accordingly. If there is a conflict, then it is resolved in one of two ways: Either all conflicting vertices are deemed unsuccessful and proceed to the next round, or an independent set is computed, using the so-called Hungarian heuristic, amongst all the vertices that chose the same color. The vertices in the independent set receive their final colors and exit. The Hungarian heuristic for independent set is to consider the vertices in random order, deleting all neighbors of an encountered vertex which itself is added to the independent set, see [1, p. 91] for a cute analysis of this heuristic to prove Turan's Theorem.
4. *Feed the Hungry!*: If a vertex runs out of colors in its palette, then fresh new colors are given to it.

Several parameters can be varied in this basic scheme: the wake up probability, the conflict resolution and the size of the initial palette are the most important ones.

In $(\Delta + 1)$-coloring, the initial palette for a vertex v is set to $[\Delta] := \{1, \cdots, \Delta + 1\}$ (global setting) or $[d(v) + 1]$ (where $d(v)$ is the degree of vertex v) (local setting). The experimental results indicate that (a) the best wake-up probability is 1, (b) the local palette version is as good as the global one in running time, but can achieve significant color savings and (c) the Hungarian heuristic can be used with vertex identities rather than random numbers giving good results.

In the Brooks–Vizing colorings, the initial palette is set to $[d(v)/s]$ where s is a *shrinking factor*. The experimental results indicate that uniformly, the best algorithm is the one where the wake-up probability is 1, and conflicts are resolved by the Hungarian heuristic. This is both with respect to the running time, as well as the number of colors used. Realistically useful values of s are between 4 and 6 resulting in Δ/s-colorings. The running time performance is excellent, with even graphs with a thousand vertices colored within 20–30 rounds. When compared to

the best sequential algorithms, these algorithms use between twice or thrice as many colors, but are much faster.

Data Sets

Test data was both generated synthetically using various random graph models, and benchmark real life test sets from the second DIMACS implementation challenge [3] and Joe Culberson's web-site [2] were also used.

Cross-References

- Graph Coloring
- Randomization in Distributed Computing
- Randomized Gossiping in Radio Networks

Recommended Reading

1. Alon N, Spencer J (2000) The probabilistic method. Wiley, New York
2. Culberson JC. http://web.cs.ualberta.ca/~joe/Coloring/index.html
3. Ftp site of DIMACS implementation challenges. ftp://dimacs.rutgers.edu/pub/challenge/
4. Finocchi I, Panconesi A, Silvestri R (2004) An experimental analysis of simple distributed vertex soloring algorithms. Algorithmica 41:1–23
5. Garey M, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman, San Francisco
6. Grable DA, Panconesi A (2000) Fast distributed algorithms for Brooks–Vizing colorings. J Algorithms 37:85–120
7. Johansson Ö (1999) Simple distributed $(\Delta + 1)$-coloring of graphs. Inf Process Lett 70:229–232
8. Kim J-H (1995) On Brook's theorem for sparse graphs. Comb Probab Comput 4:97–132
9. Luby M (1993) Removing randomness in parallel without processor penalty. J Comput Syst Sci 47(2):250–286
10. Molly M, Reed B (2002) Graph coloring and the probabilistic method. Springer, Berlin
11. Peleg D (2000) Distributed computing: a locality-sensitive approach. SIAM monographs on discrete mathematics and applications, vol 5. Society for Industrial and Applied Mathematics, Philadelphia
12. Trick M Michael Trick's coloring page. http://mat.gsia.cmu.edu/COLOR/color.html

Document Retrieval on String Collections

Rahul Shah
Department of Computer Science, Louisiana State University, Baton Rouge, LA, USA

Keywords

Geometric range searching; String matching; Suffix tree; Top-k retrieval

Years and Authors of Summarized Original Work

2002; Muthukrishan
2009; Hon, Shah, Vitter
2012; Navarro, Nekrich
2013; Shah, Sheng, Thankachan, Vitter

Problem Definition

Indexing data so that it can be easily searched is one of the most fundamental problems in computer science. Especially in the fields of databases and information retrieval, indexing is at the heart of query processing. One of the most popular indexes, used by all search engines, is the inverted index. However, in many cases like bioinformatics, eastern language texts, and phrase queries for Web, one may not be able to assume word demarcations. In such cases, these documents are to be seen as a string of characters. Thus, more sophisticated solutions are required for these string documents.

Formally, we are given a collection of D documents $\mathcal{D} = \{d_1, d_2, d_3, \ldots, d_D\}$. Each document d_i is a string drawn from the character set Σ of size σ and the total number of characters across all the documents is n. Our task is to preprocess this collection and build a data structure so that queries can be answered as quickly as possible. The query consists of a pattern string P, of length p, drawn from Σ. As the answer to the query, we are supposed output all the documents d_i in which this pattern P occurs as a substring. This

is called the *document listing* problem. In a more advanced top-k version, the query consists of a tuple (P, k) where k is an integer. Now, we are supposed to output only the k most *relevant* documents. This is called the *top-k document retrieval* problem.

The notion of relevance is captured by a *score function*. The function score(P, d) denotes the score of the document d with respect to the pattern P. It can be the number of times P occurs in d, known as *term frequency*, or the distance between two closest occurrences of P in d, or any other function. Here, we will assume that score(P, d) is solely dependent on the set of occurrences of P in d and is known at the time of construction of the data structure.

Key Results

The first formal study of this problem was initiated by Muthukrishnan [4]. He took the approach of augmenting the generalized suffix tree with additional information. All subsequent works have used generalized suffix trees as their starting point. A generalized suffix tree GST is a compact trie of all the lexicographically sorted suffixes of all the documents. Thus, n total suffixes are stored and there are n leaves in this trie. Each edge in GST is labeled with a string and each root to leaf path (labels concatenated) represents some suffix of some document. The overall number of nodes in GST is $O(n)$. With each leaf, we associate a document id, which indicates the document to which that particular suffix belongs.

When the pattern P comes as a query, we first traverse from the root downward and find a vertex, which is known as locus(P). This can be done in $O(p)$ time. This is the first vertex below the edge where P finished in the root to leaf traversal. If v is locus(P) then all the leaves in the subtree of v represent the suffixes whose prefix is P. For any vertex v, let path(v) be the string obtained by concatenating all the labels from the root until v.

Document Listing Problem

Let us first see how the document listing problem is solved. One easy solution is to reach the locus v of P and then visit all the leaves in the subtree of v. But this is costly as the number of leaves `occ` may be much more than number of unique document labels `ndoc` among these leaves. Optimally, we want to achieve $O(p + \texttt{ndoc})$ time.

To overcome this issue, Muthukrishnan first proposed to use a document array D_A. To construct this, he traverses all the leaves in GST from left-to-right and takes the corresponding document id. Thus, $D_A[i]$ = document id of ith lexicographically smallest suffix in GST. It is easy to find boundary points sp and ep such that the subtree of locus v corresponds to entries form $D_A[sp, \ldots, ep]$. To uniquely find documents in $D_A[sp, \ldots ep]$, we must not traverse the entire subarray as this would cost us O(`occ`). To avoid this, we construct another array C called a chain array. $C[i] = j$, where $j < i$ is the largest index for which $D_A[i] = D_A[j]$. If no such j exists then $C[i] = -1$. Thus, every document entry $D_A[i]$ links to the previous entry with the same document id. Now, to solve the document listing problem, one needs to get all the i's such that $sp \leq i \leq ep$ and $C[i] < sp$. The second constraint guarantees that every document is output only once. Muthukrishnan shows how to repeatedly apply range minimum queries (RMQ) to achieve constant time per output id.

Theorem 1 ([4]) *Given a collection of documents of total length n, we can construct a data structure taking $O(n)$ space, such that document listing queries for pattern P can be answered in optimal $O(p + \texttt{ndoc})$ time.*

Top-*k* Document Retrieval

Hon et al. [3] brought in an additional constraint of *score function* which can capture various notions of relevance like frequency and proximity. Instead of reporting all the documents, we only care to output the k highest scoring documents, as these would be the most relevant. Thus, $O(p + \texttt{ndoc})$ time is not optimal. We briefly describe their solution.

Let ST_d denote a suffix tree only for suffixes of document d. They augment the GST with *Links*. Link L is a 4-tuple: origin_node o, target_node t, document_id d, score_value s.

Essentially, $(L.o,\ L.t,\ L.d,\ L.s)$ is a link if and only if (t', o') is an edge in the suffix tree ST_d of the document d. Here, t' is the node in ST_d for which $\text{path}(t') = \text{path}(L.t)$. Similarly, $\text{path}(o') = \text{path}(L.o)$. The score value of the link $L.s = \text{score}(\text{path}(o'), d)$. For $L.o$ and $L.t$, we use the preorder-id of those nodes in the GST. The total number of link entries is the same as the sum of number of edges in each individual suffix tree of all the documents, which is $O(n)$. They store these links in an array $\mathcal{L}$, sorted by target. In case of a tie among targets, we sort them further by their origin values.

When they execute the query (P, k), they first find the locus node v in GST. Now, the task is to find the top-k highest scoring documents within the subtree of v. Because the links are essentially edges in individual suffix trees, for any document d_i, there is at most one link whose origin o is within the subtree of v and whose target t is outside the subtree. Moreover, note that if the target t is outside the subtree then it must be one of the ancestors of v. The score of this link is exactly the $\text{score}(P, d_i)$. Then the query is: Among all the links, whose origin starts within the subtree of locus v and whose target is outside the subtree of v, find the top-k highest scoring links. The documents are to be output in sorted order of score values. Let f_v be the preorder value of v and l_v be the preorder value of the last node in the subtree of v, then any qualifying link L would satisfy $f_v \leq L.o \leq l_v$ and $L.t \leq f_v$. And then, among all such links, only get k with the highest scores.

Their main idea is that one needs to look for at most p different target values – one for each ancestor of v. In the sorted array $\mathcal{L}$, these links come as at most p different subarray chunks. Moreover, within every target value the links originating from the subtree of v also come contiguously. Let $(l_1, r_1), (l_2, r_2), \ldots, (l_g, r_g)$ with $g < p$ be the intervals of the array $\mathcal{L}$ in which any link L satisfies $f_v \leq L.o \leq l_v$ and $L.t \leq f_v$. We skip here the description of how these intervals are found in $O(p)$ time. Now, the task is to get top-k highest ranking links from these intervals. For this, they construct a Range Maximum Query (RMQ) structure on score values of $\mathcal{L}$. They apply RMQs over each interval and put these values in a heap. Then they do extract-min from the heap, which at most maintains $O(k)$ elements. If they output an element from (l_a, r_a), the next greatest element from the same interval is put in the heap. They stop when the heap outputs k links. This takes $O(p + k \log k)$ time.

Theorem 2 ([3]) *Given a collection of documents of total length n, we can construct a data structure taking $O(n)$ space, such that top-k document retrieval queries (P, k) can be answered in $O(p + k \log k)$ time.*

Navarro and Nekrich [5] further improved the time to optimal $O(p + k)$. To achieve this, they first change the target attribute of the link to target_depth td. They model the links as two dimensional points (x_i, y_i) with weights w_i as the score. They maintain a global array of these points sorted by their x-coordinates, which are the preorders of origins of the links, while y stands for target_depth. If h is the depth of locus v and v spans the preorders $[a, b]$, then their query is to obtain points in $[a, b] \times [0, h]$ with top-k highest weights. First, they make a basic unit structure for $m \leq n$ points and answer these queries in $O(m^f + k)$ time. Here, $0 < f < 1$ is a constant. This is done by partitioning points by weights. Within each partition, the weights are disregarded. Then they start executing query $Q = [a, b] \times [0, h]$ from the highest partition. If less than $k' < k$ points qualify, then they output these points, change k to $k - k'$ and go to the next partition and so on. At some stage, in some partition more than k points will qualify. One cannot output all of them, and must get only the highest weighted points from that partition. For this, the partitions are further recursively divided into next level partitions. The depth of this recursion is constant and there are at most $O(m^f)$ partitions to be queried and each point is output in constant time. This gives $O(m^f + k)$ time. For sorted reporting, they show how Radix sort can be applied in a constant number of rounds.

Next, with this as a basic unit, they show how to create a layerwise data structure, so that we choose the appropriate layer according to h when

the query comes. The parameters of that layer ensure that we get $O(h + k)$ time for the query.

Theorem 3 ([5]) *Given a collection of documents of total length n, we can construct a data structure taking $O(n)$ space, such that top-k document retrieval queries can be answered in $O(p + k)$ time.*

External Memory Document Retrieval

Shah et al. [6] obtained the first external memory results for this problem. In the external memory model, we take B as the block size and we count I/O (input/output) operations as the performance measure of the algorithm. They achieved optimal $O(p/B + \log_B n + k/B)$ I/Os for unsorted retrieval. They take $O(n \log^* n)$ space, which is slightly super linear. We briefly describe the structure here. They first make ranked components of GST. The rank of a node v with subtree size s_v is $\lfloor \log \lceil s_v/B \rceil \rfloor$. Ranked components are the contiguous set of vertices of the same rank. Apart from rank 0 vertices, all other components form downward paths (this is very similar to heavy path decomposition). Now, for links, instead of global array $\mathcal{L}$, they keep the set of links associated with each component. Basically, every link belongs to the component where its target is. They maintain two structures, one is a 3-sided structure in 2D [2] and the other is a 3D dominance structure [1].

The query processing first finds the locus v. Also, the query parameter k is converted into a score threshold τ using sketching structures. We are interested in links such that $f_v \le L.o \le l_v$, $L.t \le f_v$, and $L.s \ge \tau$. These are four constraints. In external memory, three constraints are manageable, but not four. So they decompose the query into those with three constraints. They categorize the answer set into two kinds of links (i) the links whose targets are in the same component as v, and (ii) the links whose targets are in components ranked higher than v. By the property of rank components there are at most $\log n/B$ such higher components. For the second kind of links, we query all the higher-ranked 3-sided (2D) structures [2] with f_v, l_v, τ as the parameters. As long as the links are coming from the subtree of v, the target values of links need not be checked. For the first kind of links, one cannot drop the target condition, i.e., $L.t \le f_v$ must be satisfied. However, they show that a slight renumbering of pre-orders based on visiting the child in its own rank component allows condition $L.o \le l_v$ to be dropped. Such queries are answered by 3D dominance structures. Using this, they obtain $O(p/B + \log^2(n/B) + k/B)$ query I/Os with linear space structure. They further bootstrap this to remove the middle term $\log \log^2(n/B)$ by doubling space requirements. This recursively leads to the following result.

Theorem 4 ([6]) *Given a document collection of size n, we can construct an $O(n \log^* n)$ space structure in external memory, which can answer the top-k document retrieval queries (P, k) in $O(p/B + \log_B n + k/B)$ I/Os. The output is unsorted.*

As a side effect of this result, they also obtain internal memory sorted top-k retrieval in $O(p + k)$ time like [5], and better, just $O(k)$ time if locus(P) is given. This is because the answers come from at most $\log n$ different components. For dominance and 3-sided queries, one can get sorted outputs. And then, atomic heaps are used to merge at most $\log n$ sorted streams. Since atomic heaps only have $O(\log n)$ elements at a time, they can generate each output in $O(1)$ time.

Cross-References

- Compressed Document Retrieval on String Collections
- Suffix Trees and Arrays

Recommended Reading

1. Afshani P (2008) On dominance reporting in 3d. In: ESA, Karlsruhe, pp 41–51
2. Arge L, Samoladas V, Vitter JS (1999) On two-dimensional indexability and optimal range search indexing. In: PODS, Philadephia, pp 346–357
3. Hon WK, Shah R, Vitter JS (2009) Space-efficient framework for top-k string retrieval problems. In: FOCS, Atlanta, pp 713–722

4. Muthukrishnan S (2002) Efficient algorithms for document retrieval problems. In: SODA, San Francisco, pp 657–666
5. Navarro G, Nekrich Y (2012) Top-k document retrieval in optimal time and linear space. In: SODA, Kyoto, pp 1066–1077
6. Shah R, Sheng C, Thankachan SV, Vitter JS (2013) Top-k document retrieval in external memory. In: ESA, Sophia Antipolis, pp 803–814

Double Partition

Xiaofeng Gao
Department of Computer Science, Shanghai Jiao Tong University, Shanghai, China

Keywords

Approximation; Connected dominating set; Unit disk graph; Wireless network

Years and Authors of Summarized Original Work

2008; Gao, Huang, Zhang, Wu
2009; Huang, Gao, Zhang, Wu

Problem Definition

This problem deals with the design and analysis of a novel approximation algorithm for the minimum weight connected dominating set problem (MWCDS) under unit disk graph (UDG) model. The WCDS is proved to be NP-hard in 1970s, while for a long period researchers could not find a constant-factor approximation until 2006, when Ambühl et al. first introduced an approximation with ratio of 89 under UDG model. Inspired by their subroutines, we proposed a $(10 + \epsilon)$-approximation with double partition technique, which greatly improved the efficiency and effectiveness of the problem.

Given a homogeneous wireless ad hoc network, represented as an undirected graph $G = (V, E)$, where each vertex $v_i \in V$ in the network has the same communication range, we denote it as the *unit* 1 distance. Two nodes v_1, v_2 can communicate with each other when their Euclidean distance is smaller than 1, and correspondingly, the edge set $E = \{(v_i, v_j) \mid \text{dist}(v_i, v_j) \leq 1\}$. If v_i and v_j are connected, then we say that v_j is a neighbor of v_i (and vice versa). Such communication model is named as *unit disk graph* (UDG). Additionally, each vertex v_i has a weight w_i.

Objective

We hope to find a connected dominating subset $U \subseteq V$ in the given graph with the minimum weight, such that each vertex $v_i \in V$ is either in U or has a neighbor in U and the induced graph $G[U]$ is connected. In addition, the weight of U (defined as the sum of weights for elements in U, say, $W(U) = \sum_{v_i \in U} w_i$) is the minimum among all connected dominating subsets satisfying the above requirements.

Constraints

1. **Unit disk graph:** We restrict our discussion on two-dimensional space where each vertex has the same communication range, and edges between vertices are constructed according to the distance constraint.
2. **Weight minimization:** We focus on the weight version of minimum connected dominating set problem, which is much harder than the cardinality version, and thus providing a constant-factor approximation seems more difficult.

Problem 1 (Minimum Weight Connected Dominating Set in Unit Disk Graph)

INPUT: *A unit disk graph $G = (V, E)$ and a weight assigned on each vertex*

OUTPUT: *A minimum weight connected dominating vertex subset $U \subset V$ such that (1) wirelength is minimized and (2) the area-density constraints $D_{ij} \leq K$ are satisfied for all $B_{ij} \in B$*

Key Results

The minimum weight connected dominating set problem (MWCDS) can be divided into two parts: selecting a minimum weight dominating set (MWDS) and connecting the dominating set into a connected dominating set. In this chapter, we will focus on the former part, while the latter part is equivalent to solving a node-weighted Steiner tree problem in unit disk graphs.

The first constant-factor approximation algorithm for MWCDS under UDG was proposed by Ambühl C. et al. in 2006 [1], which is a polynomial-time 89-approximation. Later, Gao X. et al. [2] introduced a better approximation scheme with approximation ratio of $(6+\epsilon)$ for MWDS, and Huang Y. et al. [3] further extended this idea to MWCDS with approximation ratio of $(10+\epsilon)$. The main idea of their methods involves a double partition technique and a shifting strategy to reduce the redundant vertices selected through the algorithms.

In recent year, the approximation for MWDS in UDG received further improvements from $(6+\epsilon)$ to 5 by Dai and Yu [4], to 4 by Erlebach T. et al. [5] and Zou F. et al. [6] independently, and to 3.63 by Willson J. et al. [7]. Meanwhile, to connect the dominating set in UDG, Ambühl C. et al. [1] gave a 12-approximation, Huang Y. et al. [3] provided a 4-approximation, and Zou F. et al. [8] constructed a 2.5ρ-approximation with a known ρ-approximation algorithm for the minimum network Steiner tree problem. Recently, the minimum approximation for network Steiner tree problem has an approximation ratio of 1.39 [9], so the best approximation ratio for MWCDS problem in UDG is 7.105 up to now.

Double Partition Technique

Given a UDG G containing n disks in the plane. Let $\mu < \frac{\sqrt{2}}{2}$ be a real number which is sufficiently close to $\frac{\sqrt{2}}{2}$, say, $\mu = 0.7$. Partition the area into squares with side length μ. If the whole area has boundary $P(n) \times Q(n)$, where $P(n)$ and $Q(n)$ are two polynomial functions on n, then given the integer even constant K and letting $K \times K$ squares form a block, our partition will have at most $\left(\lceil \frac{P(n)}{K} \rceil + 1\right) \times \left(\lceil \frac{Q(n)}{K} \rceil + 1\right)$ blocks. We will discuss algorithm to compute MWDS for each block firstly and then combine them together.

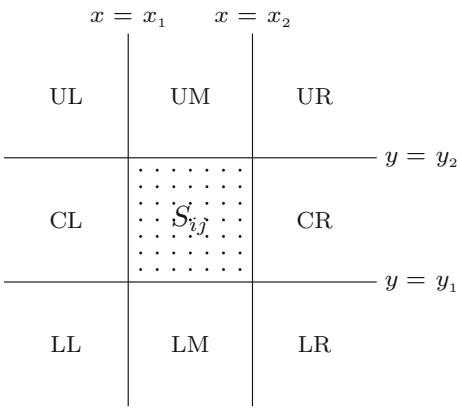
MWDS in $K \times K$ Squares

Assume each block B has K^2 squares S_{ij}, for $i, j \in \{0, 1, \ldots, K-1\}$. Let V_{ij} be the set of disks in S_{ij}. If we have a dominating set D for this block, then for each square S_{ij}, its corresponding dominating set is (1) either a disk from inside S_{ij} (since $\text{dist}(d, d') \leq 1$ for any two disks within this square) or (2) a group of disks from neighbor region around S_{ij}, the union of which can cover all disk centers inside the square. Then if we want to select a minimum weight dominating set, for each square we will have two choices. However, instead of selecting dominating sets square by square, we hope to select them strip by strip to avoid repeated computation for some disks. For this purpose, we have the following lemmas.

Lemma 1 ([1]) *Let P be a set of points located in a strip between lines $y = y_1$ and $y = y_2$ for some $y_1 < y_2$. Let D be a set of weighted disks with uniform radius whose centers are above the line $y = y_2$ or below the line $y = y_1$. Furthermore, assume that the union of the disks in D covers all points in P. Then a minimum weight subset of D that covers all points in P can be computed in polynomial time.*

The proof of result for Lemma 1 is in fact constructive. It gives a polynomial-time algorithm by a dynamic programming. It says that as long as the set of centers P in a horizontal strip can be dominated by a set of centers D above and/or below the strip, then an optimal subset of D dominating P can be found in polynomial time.

Our next work is to select some disks for each square within a strip so that those disks can be covered by disks from the upper and lower strips. To better illuminate the strategy, we divide the neighbor parts of S_{ij} into eight regions $UL, UM, UR, CL, CR, LL, LM, and LR$ as shown in Fig. 1. The four lines forming S_{ij} are $x = x_1$, $x = x_2$, $y = y_1$, and $y = y_2$.



Double Partition, Fig. 1 S_{ij} and its neighbor regions

Denote by $Left = UL \cup CL \cup LL$, $Right = UR \cup CR \cup LR$, $Up = UL \cup UM \cup UR$, and $Down = LL \cup LM \cup LR$. After that, we will have Lemma 2.

Lemma 2 *Suppose $p \in V_{ij}$ is a disk in S_{ij} which can be dominated by a disk $d \in LM$. We draw two lines p_l and p_r, which intersect $y = y_1$ by angle $\frac{\pi}{4}$ and $\frac{3\pi}{4}$. Then the shadow P_{LM} surrounded by $x = x_1$, $x = x_2$, $y = y_1$, p_l, and p_r (shown in Fig. 2) can also be dominated by d. Similar results can be held for shadow P_{UM}, P_{CL}, and P_{CR}, which can be defined with a rotation.*

Proof We split shadow P_{LM} into two halves with vertical line $x = x_p$, where x_p is x-coordinate of disk p. Then we prove that the right half of P_{LM} can be covered by d. The left half can be proved symmetrically. Let o be intersection point of $x = x_p$ and $y = y_1$, a that of p_r and $x = x_2$ (or p_r and $y = y_1$), and b that of $x = x_2$ and $y = y_1$. Intuitionally, the right half can be either a quadrangle $pabo$ or a triangle pao. We will prove both cases as follows:

Quadrangle case: Draw the perpendicular line of the line segment pa, namely, p_m. When d is under p_m as in Fig. 3a, we will have $\text{dist}(d, a) \leq \text{dist}(p, d) \leq 1$. Moreover, it is trivial that $\text{dist}(d, o)$ and $\text{dist}(d, b)$ are all < 1. Thus, d can cover the whole quadrangle. When d is above the line p_m as in Fig. 3b, we draw an auxiliary line $y = y_a$ parallel with $y = y_1$ and $x = x_d$ intersecting $y = y_a$ at point c. Since d lies above p_m, $\angle cad \leq \pi/4$, and thus

$$\text{dist}(d, a) = \frac{\text{dist}(c, a)}{\cos \angle cad} < \frac{\sqrt{2}/2}{\cos \frac{\pi}{4}} = 1.$$

Note that both $\text{dist}(d, o)$ and $\text{dist}(d, b)$ are less than 1, so d can cover the whole quadrangle.

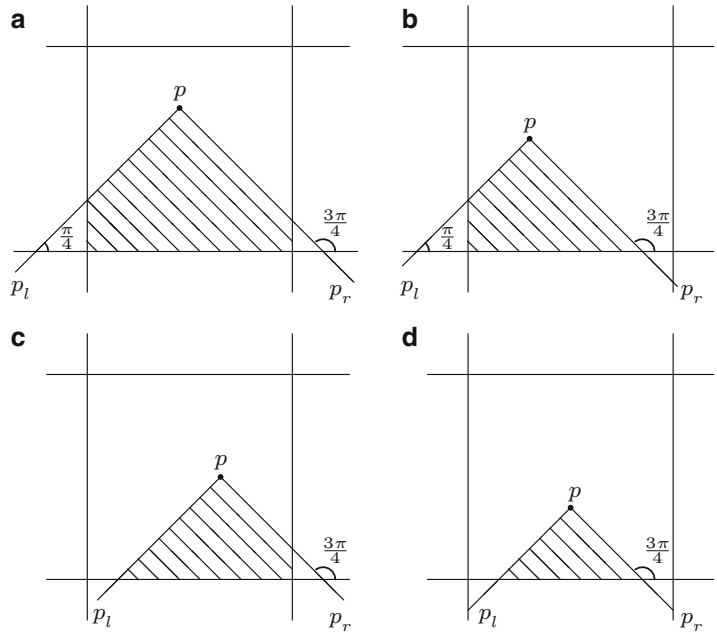
Triangle case: Similarly, draw p_m as described above. The proof remains when d is under p_m (see Fig. 3c). When d is above p_m as in Fig. 3d, we draw auxiliary line $x = x_d$ intersecting $y = y_1$ at c. Then we will get the same conclusion.

With the help of Lemma 2, we can select a region from S_{ij}, where the disks inside this region can be covered by disks from Up and $Down$ neighbor area. We name this region as "sandglass," with formal definition as follows:

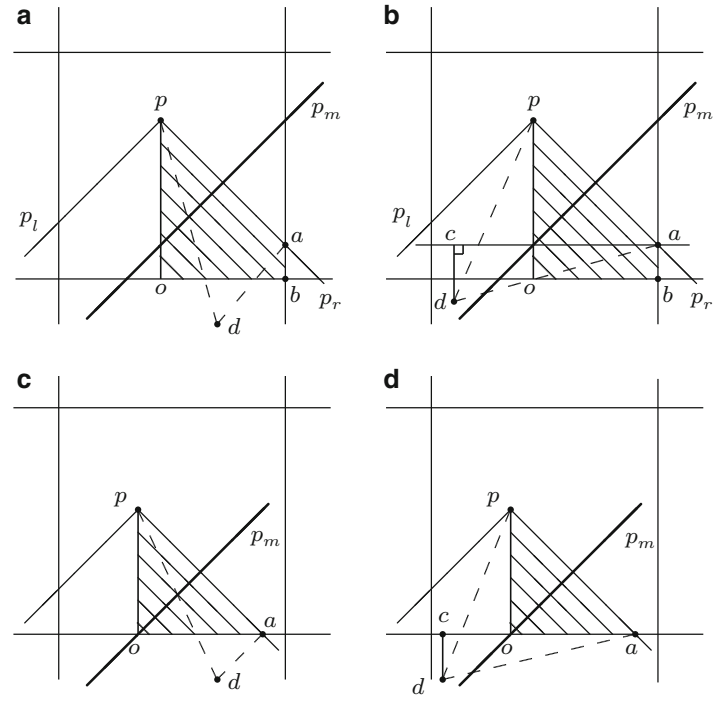
Definition 1 (Sandglass) If D is a dominating set for square S_{ij} and $D \cap V_{ij} = \emptyset$, then there exists a subset $V_M \subseteq V_{ij}$ which can only be covered by disks from UM and LM (we can set $V_M = \emptyset$ if there are no such disks). Choose $V_{LM} \subseteq V_M$ the disks that can be covered by disks from LM, and draw p_l and p_r line for each $p \in V_{LM}$. Choose the leftmost p_l and rightmost p_r and form a shadow similar as that in Lemma 2. Symmetrically, choose V_{UM} and form a shadow with leftmost and rightmost lines. The union of the two shadows form a "sandglass" region Sand_{ij} of S_{ij} (see Fig. 4a, where solid circle represents V_{LM}, while hollow circle represents V_{UM}). Fig. 4b–4d gives other possible shapes of Sand_{ij}.

Lemma 3 *Suppose D is a dominating set for S_{ij} and Sand_{ij}s are chosen in the above way. Then any disks in $Sand_{ij}$ can be dominated by disks only from neighbor region $Up \cup Down$, and disks from $S_{ij} \setminus Sand_{ij}$ can be dominated by disks only from neighbor region $Left \cup Right$.*

Proof Suppose to the contrary, there exists a disk $d \in \text{Sand}_{ij}$ which cannot be dominated by disks from $Up \cup Down$. Since D is a dominating set, there must be a $d' \in CL \cup CR$ which dominates

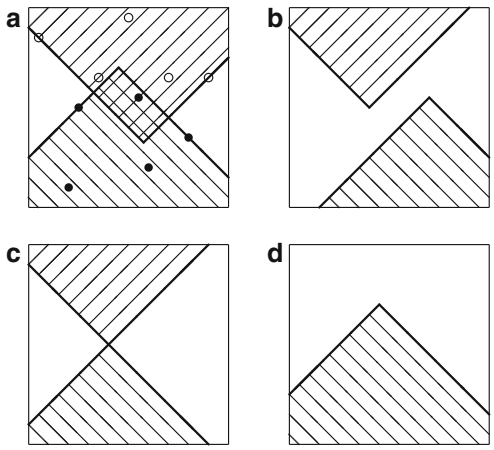


Double Partition, Fig. 2
Different shapes for shadow P_{LM}

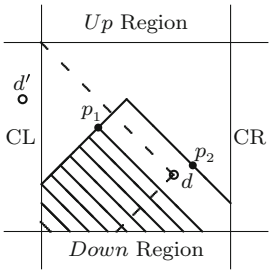


Double Partition, Fig. 3
Shape of shadow and location of d. (**a**) d to left of p_m. (**b**) d to left of p_m. (**c**) d to left of p_m. (**d**) d to left of p_m

d. Without loss of generality, assume d belongs to lower half of the sandglass which is formed by p_1 and p_2, and let $d' \in CL$ (see Fig. 5). Based on our assumption, d cannot locate in p_1's triangle shadow to $Down$ region (otherwise, since p_1 can be dominated by a disk from LM, d can also be dominated by this disk). We then draw d_l and d_r to CL region and form a shadow to CL. Then by Lemma 2 every disk from this shadow can be dominated by d'. Obviously, p_1 belongs



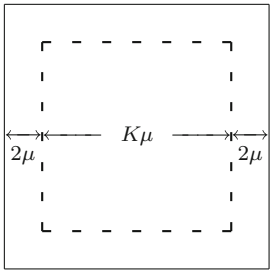
Double Partition, Fig. 4 Sandglass Sand_{ij} for S_{ij}. (**a**) Form of sandglass. (**b**) Sandglass without intersection. (**c**) Sandglass with single disk. (**d**) Sandglass with half side



Double Partition, Fig. 5 Proof for sandglass

to this region, but p_1 is a disk which cannot be dominated by disks from CL, a contradiction.

Till now we already find "sandglass" region in which disks can be dominated by disks only from Up and $Down$ regions. In our algorithm, for each square S_{ij}, we can firstly decide whether to choose a disk inside this square as dominating set or to choose a dominating set from its neighbor region. If we choose the latter case, the algorithm will randomly select 4 disks d_1, d_2, d_3, and d_4 from S_{ij} and make corresponding sandglass (we can also choose less than 4 disks to form the sandglass). By enumeration of all possible sandglasses including the case of choosing one disk inside the square, for all squares within $K \times K$ area, there are at most $[\sum_{i=0}^{4} C_n^i \cdot 2^i]^{K^2}$ choices (n is the number of disks), which can be calculated within polynomial time. Moreover, when considering choosing a dominating set from neighbor



Double Partition, Fig. 6 Block selection

Algorithm 1 Calculate MWDS in $K \times K$ squares

Input: $K \times K$ squares with inner disks
Output: A local MWDS

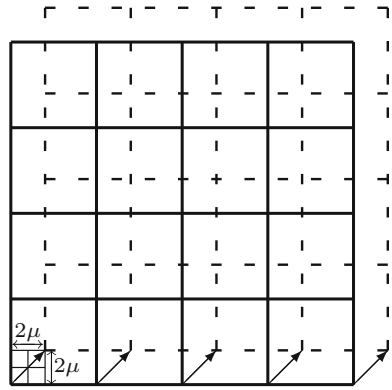
1: For each S_{ij}, choose its sandglass or randomly select $d \in S_{ij}$.
2: If $d \in S_{ij}$ is selected, then remove d and all disks dominated by d.
3: For each strip $\cup_{j=1}^{K} S_{ij}$ from $i = 1$ to K, calculate a dominating set for the union of disks in the sandglasses.
4: For each strip $\cup_{i=1}^{K} S_{ij}$ from $j = 1$ to K, calculate a dominating set for the remaining disks not covered by Step 3.
Return the union of disks chosen in the above steps for $K \times K$ squares.

regions, we should also include regions around this $K \times K$ areas such that we will not miss disks outside the region. Therefore, we should consider $(K+4) \times (K+4)$ area, where the inner region is our selected block and the surrounding four strips are the assistance (shown as Fig. 6).

In all, we will have Algorithm 1 with four steps to calculate an MWDS for $K \times K$ squares. We enumerate all possible cases for each S_{ij} and choose the solution with minimum weight, which forms an MWDS for S_{ij}.

MWDC for the Whole Region

As discussed above, if our plane has size $P(n) \times Q(n)$, then there are at most $\left(\lceil \frac{P(n)}{K} \rceil + 1\right) \times \left(\lceil \frac{Q(n)}{K} \rceil + 1\right)$ blocks in the plane. We name each block B^{xy}, where $0 \le x \le \lceil \frac{P(n)}{K} \rceil + 1$ and $0 \le y \le \lceil \frac{Q(n)}{K} \rceil + 1$. Then, using Algorithm 1 to calculate dominating set for each block and by combining them together, we obtain a dominating set for our original partition.



Double Partition, Fig. 7 Move blocks

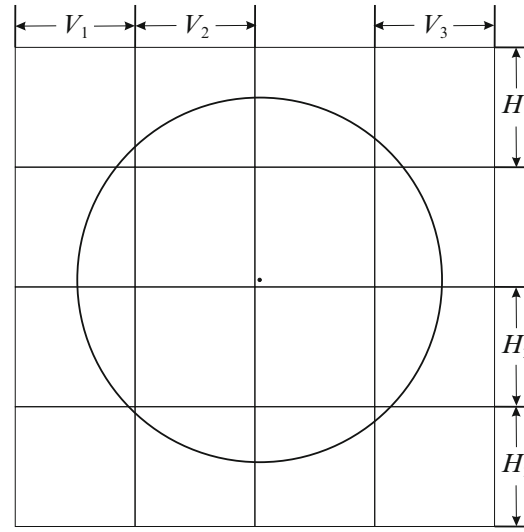
Algorithm 2 Calculate MWDS for the whole plane

Input: G in region $P(n) \times Q(n)$
Output: A global MWDS
1: For a certain partition, calculate MWDS for each block B^{xy}, sum the weight of MWSD for each block, and form a solution.
2: Move each block to two squares to the right and two squares to the top of the original block.
3: Repeat Step 1 for new partition, and get a new solution.
4: Repeat Step 2 for $\lceil \frac{K}{2} \rceil$ times, and choose the minimum solution among those steps.
Return the solution from Step 4 as our final result.

Next, we move our blocks to different positions by shifting policy. Move every block two squares right and two squares up to its original position, which can be seen from Fig. 7. Then calculate dominating set for each block again, and combine the solution together. We do this process $\frac{K}{2}$ times, choose the minimum solution as our final result. The whole process can be shown as Algorithm 2.

Performance Ratio

In the following, we extend our terminology "dominate" to points (a point is a location which is not necessarily a disk). A point p is *dominated* by a set of disks if the distance between p and at least one center of the disks is not more than 1. We say an area is *dominated* by a set of disks if every point in this area is dominated by the set of disks. Let OPT be optimal solution for our problem and $w(OPT)$ the weight of optimal solution.



Double Partition, Fig. 8 An example for disk cover region

Theorem 1 *Algorithm 2 always outputs a dominating set with weight within* $6 + \varepsilon$ *times of the optimum one.*

Proof Our proof mainly has two phases. The first phase analyzes that our Algorithm 1 gives a 6-approximation for disks in $K \times K$ squares. The second phase proves that result from Algorithm 2 is less than $(6 + \varepsilon) \cdot w(OPT)$.

Phase 1: If a disk has radius 2 and our partition has side length $\mu < \frac{\sqrt{2}}{2}$, then a disk may dominate disks from at most 16 squares, which can be shown in Fig. 8. Simply, if a disk in OPT is used to dominate the square it belongs to, then we will remove this disk before calculating MWDS for strips. Therefore, it will be used only once. If a disk is not used to dominate the square containing it, then it may be used 3 times in calculating its 3 horizontal neighbor strips (H_1, H_2, and H_3 as shown in Fig. 8) and another 3 times in calculating its 3 vertical neighbor strips (V_1, V_2, and V_3 in Fig. 8). Therefore, Algorithm 1 is a 6-approximation for each block.

Phase 2: Now we consider the disks in side strips for a block. As discussed above, when calculating MWDS for a strip, we may use disks

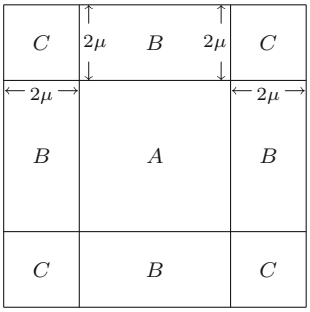
within $(K+2)\times(K+2)$ squares. Therefore, we can divide a block $B^{(xy)}$ into three kinds of squares, just as shown in Fig. 9 ($0 \le x \le \lceil \frac{P(n)}{K} \rceil$, and $0 \le y \le \lceil \frac{Q(n)}{K} \rceil$). If a disk belongs to inner part $\mathcal{A}$ of $B^{(xy)}$, it will be used at most 6 times during calculating process. We name those disks as d_{in}. If a disk belongs to side part $\mathcal{B}$ of $B^{(xy)}$, it may be used at most 5 times for calculating $B^{(xy)}$, but it may used at most 4 times when calculating $B^{(xy)}$'s neighbor block. We name those disks as d_{side}. If a disk belongs to corner squares $\mathcal{C}$ of $B^{(xy)}$, it may be used at most 4 times for calculating $B^{(xy)}$ and at most 8 times for neighbor blocks. We name those disks as d_{corner}. In addition, we know that during shifting process a node can stay at most 4 times in side or corner square. If we name l as the lth shifting, then our final solution will have the following inequality:

$$
\begin{aligned}
&W(\text{Solution}) \\
&= \min_{l} \Big\{ \sum_{\text{Sol}_l} [6w(d_{\text{in}}^{l}) + 9w(d_{\text{side}}^{l}) + 12w(d_{\text{corner}}^{l})] \Big\} \\
&\le \frac{1}{\frac{K}{2}} \sum_{l=0}^{\frac{K}{2}} \big\{ 6w(d_{in}^{l}) + 4 \cdot 12w(d_{\text{side}}^{l} + d_{\text{corner}}^{l}) \big\} \\
&= 6w(OPT) + \frac{42}{\frac{K}{2}} w(OPT) \\
&\le (6+\varepsilon) w(OPT)
\end{aligned}
$$

where $\varepsilon = 42/\frac{K}{2}$ can be arbitrarily small when K is sufficiently large.

Applications

Dominating set problem is widely used in network-related applications. For instance, in mobile and wireless ad hoc networks, it is implemented for communication virtual backbone selection to improve routing efficiency, for sensor coverage problem to extend network lifetime, and for clustering and data gathering problem to avoid flooding and energy waste. In optical network and data center networks, it is used for network management and switch-centric routing protocols. In social network applications, it is used for many cluster-related problems like positive influence, effective leader group, etc. A weighted dominating set is a generalized heterogeneous network model to describe real-world applications, which is more realistic and practical.



Double Partition, Fig. 9 Divide block $B^{(xy)}$ into 3 parts

Open Problems

There are two open problems for the minimum weighted connected dominating set (MWCDS) problem under unit disk graph (UDG). Firstly, there is another grid partition design to construct a constant approximation for domatic partition problem in unit disk graph [10]. Could this new technique result in an improvement in running time or performance ratio? Secondly, does MWCDS problem in UDG have a polynomial-time approximation scheme (PTAS)? Currently, no one has answered the above questions.

Cross-References

▸ Minimum Connected Sensor Cover

Recommended Reading

1. Ambühl C, Erlebach T, Mihalák M, Nunkesser M (2006) Constant-factor approximation for minimum weight (connected) dominating sets in unit disk graphs. In: Proceedings of the 9th international workshop on approximation algorithms for combinatorial optimization problems (APPROX 2006), Barcelona, 28–30 Aug 2006, pp 3–14
2. Gao X, Huang Y, Zhang Z, Wu W (2008) $(6+\epsilon)$-approximation for minimum weight dominating set in unit disk graphs. In: Proceedings of the 14th annual international conference on computing and combinatorics (COCOON), Dalian, 27–29 June 2008, pp 551–557

3. Huang Y, Gao X, Zhang Z, Wu W (2009) A better constant-factor approximation for weighted dominating set in unit disk graph. J Comb Optim 18(2):179–194
4. Dai D, Yu C (2009) A $(5+\epsilon)$-approximation algorithm for minimum weight dominating set in unit disk graph. Theor Comput Sci 410(8–10):756–765
5. Erlebach T, Mihalák M (2009) A $(4+\epsilon)$-approximation for the minimum weight dominating set problem in unit disk graph. In: Proceedings of the 7th workshop on approximation and online algorithms (WAOA), Copenhagen, 10–11 Sept 2009, pp 135–146
6. Zou F, Wang Y, Xu X, Du H, Li X, Wan P, Wu W (2011) New approximation for weighted dominating sets and connected dominating sets in unit disk graphs. Theor Comput Sci 412(3):198–208
7. Willson J, Wu W, Wu L, Ding L, Du D-Z (2014) New approximations for maximum lifetime coverage. Optim J Math Program oper Res. doi:10.1080/02331934.2014.883507
8. Zou F, Li X, Gao S, Wu W (2009) Node-weighted Steiner tree approximation in unit disk graphs. J Comb Optim 18(4):342–349
9. Byrka J, Grandoni F, Rothvoss T, Sanita L (2010) An improved LP-based approximation for Steiner tree. In: Proceedings of the 42th ACM symposium on theory of computing (STOC 2010), Cambridge, 6–8 June 2010, pp 583–592
10. Pandit S, Pemmaraju S, Varadarajan K (2009) Approximation algorithms for domatic partitions of unit disk graphs. In: Proceedings of the 12th international workshop on approximation algorithms for combinatorial optimization problems (APPROX 2009), Berkeley, 21–23 Aug 2009, pp 312–325

Dynamic Approximate All-Pairs Shortest Paths: Breaking the *O(mn)* Barrier and Derandomization

Monika Henzinger[1], Sebastian Krinninger[2], and Danupon Nanongkai[3]
[1]University of Vienna, Vienna, Austria
[2]Faculty of Computer Science, University of Vienna, Vienna, Austria
[3]School of Computer Science and Communication, KTH Royal Institute of Technology, Stockholm, Sweden

Keywords

Approximation algorithms; Data structures; Derandomization; Dynamic graph algorithms

Years and Authors of Summarized Original Work

2013; Henzinger, Krinninger, Nanongkai

Problem Definition

Given an undirected, unweighted graph with n nodes and m edges that is modified by a sequence of edge insertions and deletions, the problem is to maintain a data structure that quickly answers queries that ask for the length $d(u, v)$ of the shortest path between two arbitrary nodes u and v in the graph, called the *distance* of u and v. The fastest *exact* algorithm for this problem is randomized and takes amortized $O\left(n^2\left(\log n + \log^2\left((m+n)/n\right)\right)\right)$ time per update and constant query time [6, 11]. In the *decremental* case, i.e., if only edge deletions are allowed, there exists a *deterministic* algorithm with amortized time $O(n^2)$ per deletion [7]. More precisely, its *total update time* for a sequence of up to m deletions is $O(mn^2)$. Additionally, there is a randomized algorithm with $O(n^3 \log^2 n)$ total update time and constant query time [1]. However, in the decremental case, when only α*-approximate* answers are required, i.e., when it suffices to output an estimate $\delta(u, v)$ such that $d(u, v) \leq \delta(u, v) \leq \alpha d(u, v)$ for all nodes u and v, the total update time can be significantly improved: Let $\epsilon > 0$ be a small constant. The fastest prior work was a class of randomized algorithms with total update time $\tilde{O}(mn)$ for $\alpha = 1+\epsilon$ [10], $\tilde{O}\left(n^{5/2+O(1/\sqrt{\log n})}\right)$ for $\alpha = 3+\epsilon$, and $\tilde{O}\left(n^{2+1/k+O(1/\sqrt{\log n})}\right)$ for $\alpha = 2k-1+\epsilon$ [4].

This leads to the question whether for $\alpha = 1+\epsilon$ (a), a total update time of $o(nm)$ is possible and (b) a deterministic algorithm with total update time $\tilde{O}(nm)$ exists.

As pointed out in [3] and several other places, a *deterministic* algorithm is interesting due to the fact that deterministic algorithms can deal with an *adaptive offline adversary* (the strongest adversary model in online computation [2, 5]), while the randomized algorithms developed so

far assume an *oblivious adversary* (the weakest adversary model) where the order of edge deletions must be fixed before an algorithm makes random choices.

Key Results

The paper of Henzinger, Krinninger, and Nanongkai [8] presents two algorithms for $\alpha = 1 + \epsilon$. The first one is a deterministic algorithm with total update time $\tilde{O}(mn)$. The second one studies a slightly relaxed version of the problem: Given a constant β, let $\delta(u, v)$ be an (α, β)-*approximation* if $d(u, v) \leq \delta(u, v) \leq \alpha d(u, v) + \beta$ for all nodes u and v. The second algorithm is a randomized algorithm with total update time $\tilde{O}(n^{5/2})$ that can guarantee both a $(1 + \epsilon, 2)$ and a $(2 + \epsilon, 0)$ approximation.

The results build on two prior techniques, namely, an exact decremental single-source shortest path data structure [7], called *ES-tree*, and the $(1 + \epsilon, 0)$-approximation algorithm of [10], called *RZ-algorithm*. The RZ-algorithm chooses for all integer i with $1 \leq i \leq \log n$, $\tilde{O}(n/(\epsilon 2^i))$ random nodes as *centers*, and maintains an ES-tree up to distance 2^{i+2} for each center. For correctness, it exploits the fact that the random choice of centers guarantees the following *invariant (I)*: For every pair of nodes u and v with distance $d(u, v)$, there exists with high probability a center c such that $d(u, c) \leq \epsilon d(u, v)$ and $d(c, v) \leq d(u, v)$. The total update time per center is $O(m2^i)$ resulting in a total update time of $\tilde{O}(mn)$. The deterministic algorithm of [8] derandomizes this algorithm by initially choosing centers fulfilling invariant (I) and after each update (a) greedily generating new centers to guarantee that (I) continues to hold and (b) moving the root of the existing ES-trees. To achieve a running time of $\tilde{O}(mn)$, the algorithm is not allowed to create more than $\tilde{O}(n/(\epsilon 2^i))$ many centers for each i. This condition is fulfilled by dynamically assigning each center a set of $\Omega(2^i)$ vertices such that no vertex is assigned to two centers.

The improved randomized algorithm uses the idea of an *emulator*, a sparser *weighted* graph that approximates the distances of the original graph. Emulators were used for dynamic shortest-path algorithms before [4]. The challenge when using an emulator is that edge deletions in the original graph might lead to edge deletions, edge insertions, or weight increases in the emulator, requiring in principle the use of a fully dynamic shortest-path algorithm on the emulator. Bernstein and Roditty [4] deal with this challenge by using an emulator where the number of *distance changes* between any two nodes can be bounded. However, the RZ-algorithm requires that the number of times that the distance between any two nodes changes is at most R before that distance exceeds R for any integer R with $1 \leq R \leq n$. As the emulator used by Bernstein and Roditty does not fulfill this property, they cannot run the RZ-algorithm on it. The new algorithm does not construct such an emulator either. Instead, it builds an emulator where the error introduced by edge insertions is limited and runs the RZ-algorithm with modified ES-trees, called *monotone ES-trees*, on this emulator. The analysis exploits the fact that the distance between any two nodes in the original graph can only *increase* after an edge deletion. Thus, even if an edge deletion leads to changes in the emulator that *decrease* their distance in the emulator, the corresponding ES-trees do not have to be updated, i.e., the distance of a vertex to its root in the ES-tree *never* decreases. The analysis shows that the error introduced through the use of monotone ES-trees in the RZ-algorithm is small so that the claimed approximation ratio is achieved. However, since the ES-trees are run on the sparse emulator the overall running time is $o(mn)$.

Open Problems

The main open problem is to find a similarly efficient algorithm in the *fully dynamic setting*, where both edge insertions and deletions are allowed. A further open problem is to extend the derandomization technique to the *exact* algorithm of [1].

Another challenge is to obtain similar results for *weighted, directed* graphs. We recently extended some of the above techniques to weighted, directed graphs and presented a randomized algorithm with $\tilde{O}\left(mn^{0.986}\right)$ total update time for $(1+\epsilon)$-approximate *single-source* shortest paths [9].

Cross-References

- ▸ Decremental All-Pairs Shortest Paths
- ▸ Fully Dynamic All Pairs Shortest Paths

Recommended Reading

1. Baswana S, Hariharan R, Sen S (2007) Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. J Algorithms 62(2):74–92. Announced at STOC, 2002
2. Ben-David S, Borodin A, Karp RM, Tardos G, Wigderson A (1994) On the power of randomization in on-line algorithms. Algorithmica 11(1):2–14. Announced at STOC, 1990
3. Bernstein A (2013) Maintaining shortest paths under deletions in weighted directed graphs. In: STOC, Palo Alto, pp 725–734
4. Bernstein A, Roditty L (2011) Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In: SODA, San Francisco, pp 1355–1365
5. Borodin A, El-Yaniv R (1998) Online computation and competitive analysis. Cambridge University Press, Cambridge
6. Demetrescu C, Italiano GF (2004) A new approach to dynamic all pairs shortest paths. J ACM 51(6):968–992. Announced at STOC, 2003
7. Even S, Shiloach Y (1981) An on-line edge-deletion problem. J ACM 28(1):1–4
8. Henzinger M, Krinninger S, Nanongkai D (2013) Dynamic approximate all-pairs shortest paths: breaking the $O(mn)$ barrier and derandomization. In: FOCS, Berkeley
9. Henzinger M, Krinninger S, Nanongkai D (2014) Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In: STOC, New York
10. Roditty L, Zwick U (2012) Dynamic approximate all-pairs shortest paths in undirected graphs. SIAM J Comput 41(3):670–683. Announced at FOCS, 2004
11. Thorup M (2004) Fully-dynamic all-pairs shortest paths: faster and allowing negative cycles. In: SWAT, Humlebæk, pp 384–396

Dynamic Approximate-APSP

Aaron Bernstein
Department of Computer Science, Columbia University, New York, NY, USA

Keywords

Dynamic graph algorithms; Hop distances; Shortest paths

Years and Authors of Summarized Original Work

2009; Bernstein

Problem Definition

A dynamic graph algorithm maintains information about a graph that is changing over time. Given a property $\mathcal{P}$ of the graph (e.g., maximum matching), the algorithm must support an online sequence of query and update operations, where an update operation changes the underlying graph, while a query operation asks for the state of $\mathcal{P}$ in the current graph. In the typical model studied, each update affects a single edge, in which case the most general setting is the *fully dynamic* one, where an update can either insert an edge, delete an edge, or change the weight of an edge. Common restrictions of this include the *decremental* setting, where an update can only delete an edge or increase a weight, and the *incremental* setting where an update can insert an edge or decrease a weight.

This entry addresses the problem of maintaining α-approximate all-pairs shortest paths (APSP) in the fully dynamic setting in a weighted, undirected graph (the approximation factor α depends on the algorithm); the goal is to maintain an undirected graph G with real-valued nonnegative edge weights under an online intermixed sequence of the following operations:

- **delete**(u, v) (update): remove edge (u, v) from G.

- **insert**(u, v) (update): insert an edge (u, v) into G.
- **change weight**(u, v, w) (update): change the weight of edge (u, v) to w.
- **distance**(u, v) (query): return an α-approximation to the shortest $u - v$ distance in G.
- **path**(u, v) (query): return an α-approximate shortest path from u to v.

Approaches

The naive approach to the fully dynamic APSP problem is to recompute shortest paths from scratch after every update, allowing queries to be answered in optimal time. Letting n be the number of vertices and m the number of edges, computing APSP requires $O(mn + n^2 \text{loglog}(n))$ time in sparse graphs [8] or slightly less than n^3 in dense graphs [9, 13]. If we allow approximation, a slightly better approach would be to construct an *approximate distance oracle* after each update, i.e., a static data structure for answering approximate distance queries quickly; an oracle for returning k-approximate distances ($k \geq 3$) can be constructed in time $O(\min\{n^2 \log(n), kmn^{1/k}\})$ [1, 11]. Another simple-minded approach would be to not perform any work during the updates and to simply compute the shortest $u-v$ path from scratch when a query arrived; using Dijkstra's algorithm with Fibonacci heaps [7], this would lead to a constant update time and a query time of $O(m+n \log(n))$.

The goal of a dynamic algorithm is to improve upon the above approaches by taking advantage of the fact that each update only affects a single edge, so one can reuse information between updates and thus avoid recomputing from scratch. In a breakthrough result, Demetrescu and Italiano showed that in the most general case of a directed graph with arbitrary real weights, one can answer updates in amortized time $O(n^2 \log^3(n))$ while maintaining optimal $O(1)$ time for distance queries [6]; Thorup improved the update time slightly to $O(n^2(\log(n)+\log^2((m+n)/n)))$ [10]. This entry addresses a recent result of Bernstein [3] which shows that in undirected graphs, one can significantly improve upon this n^2 update time by settling for approximate distances.

Key Results

Bernstein's paper starts by showing that assuming integral weights, there is an algorithm for maintaining $(2 + \epsilon)$-approximate APSP *up to a bounded distance* d with amortized update time $O(md)$. This is efficient for small distances, but in the general case d can be very large, especially in weighted graphs. Bernstein overcomes this by introducing a high-level framework for extending shortest path-related algorithms that are efficient for small distances to ones that are efficient for general graphs; he later applied this approach to two other results [4, 5].

The first step of the approach is to show that with simple scaling techniques, an algorithm that is efficient for small (weighted) distances can be extended to an algorithm that is efficient for shortest paths with few edges (regardless of weight). Applying this technique to the above algorithm yields the following result:

Definition 1 Let the *hop distance* of a path be the number of edges it contains. A graph G is said to have *approximate hop diameter* h if for every pair of vertices (x, y), there is a $(1 + \epsilon)$-approximate shortest $x-y$ path with hop distance $\leq h$.

Theorem 1 ([3]) *Let G be an undirected graph with nonnegative real edge weights, and let R be the ratio of the heaviest to the lightest nonzero weight. One can maintain $(2 + \epsilon)$-APSP in the fully dynamic setting with amortized update time $O(mh \log(nR))$, where h is the approximate hop diameter of the graph.*

Shortcut Edges

Theorem 1 provides an efficient algorithm for small h, but on its own a result that is efficient for small hop diameter is not particularly powerful as even in unweighted graphs h can be $\Omega(n)$. The second step of Bernstein's approach is to show that regardless of whether the original graph is weighted, one can add weighted edges to reduce the hop diameter. A *shortcut edge* (x, y) is a new edge constructed by the algorithm that has weight $w(x, y)$ with $\delta(x, y) \leq w(x, y) \leq (1+\epsilon)\delta(x, y)$,

where $\delta(x, y)$ is the shortest $x - y$ distance. It is clear that because shortcut weights are tethered to shortest distances, they do not change (weighted) distances in the graph. But a shortcut edge can greatly reduce hop distances; for example, in an unweighted graph where $\delta(x, y) = 1000$, adding a single shortcut edge (x, y) of weight 1000 decreases the $x - y$ hop distance to 1 while also decreasing the hop distance of paths that go through x and y. Bernstein adapts techniques from spanner and emulator theory (see in particular Thorup and Zwick's result on graph sparsification [12]) to show that in fact a small number of shortcut edges suffice to greatly reduce the hop diameter of a graph.

Theorem 2 ([3]) *Let G be an undirected graph with nonnegative real edge weights, and let R be the ratio of the heaviest to the lightest edge weight in the graph. There exists an algorithm that in time $O(m \cdot n^{O(1/\sqrt{\log(n)})} \cdot \log(nR))$ constructs a set S of $O(n^{1+O(1/\sqrt{\log(n)})} \cdot \log(nR))$ shortcut edges such that adding S to the edges of the graph reduces the approximate hop diameter to $n^{O(1/\sqrt{\log(n)})}$.*

Theorems 1 and 2 combined encapsulate the approach of Bernstein's algorithm: take an algorithm that works well for small distances, use scaling to transform it into an algorithm that is efficient for graphs of small hop diameter, and then add shortcut edges to the original graph to ensure a small hop diameter. For dynamic APSP, there are additional complications that arise from edges being inserted and deleted over time, but the basic approach remains the same.

Theorem 3 ([3]) *Let G be an undirected graph with real nonnegative edge weights, and let R be the ratio of the maximum edge weight appearing in the graph during any point in the update sequence to the minimum nonzero edge weight. There is an algorithm that maintains fully dynamic $(2+\epsilon)$-approximate APSP in amortized update time $O(m \cdot n^{O(1/\sqrt{\log(n)})} \cdot \log(nR))$ and can answer distance queries in worst-case time $O(\log\log\log(n))$.*

Open Problems

The main open problem for fully dynamic approximate APSP is to develop an efficient algorithm for maintaining $(1 + \epsilon)$ approximate distances, possibly with a small additive error in the unweighted case. Another interesting problem would be to achieve $o(n^2)$ update times for dense graphs – this can already be done to some extent by combining the result of Bernstein discussed here with the fully dynamic spanner of Baswana et al. [2], but only for unweighted graphs and at the cost of a much worse approximation ratio. Other open problems include removing the dependence on $\log(R)$ and developing an efficient *deterministic* algorithm for the problem.

Cross-References

- Decremental All-Pairs Shortest Paths
- Decremental Approximate-APSP in Directed Graphs
- Dynamic Approximate All-Pairs Shortest Paths: Breaking the $O(mn)$ Barrier and Derandomization
- Trade-Offs for Dynamic Graph Problems

Recommended Reading

1. Baswana S, Sen S (2006) Approximate distance oracles for unweighted graphs in expected $o(n^2)$ time. ACM Trans Algorithms 2(4):557–577
2. Baswana S, Khurana S, Sarkar S (2012) Fully dynamic randomized algorithms for graph spanners. ACM Trans Algorithms 8(4):35
3. Bernstein A (2009) Fully dynamic approximate all-pairs shortest paths with query and close to linear update time. In: Proceedings of the 50th FOCS, Atlanta, pp 50–60
4. Bernstein A (2012) Near linear time $(1 + \epsilon)$-approximation for restricted shortest paths in undirected graphs. In: Proceedings of the 23rd SODA, San Francisco, pp 189–201
5. Bernstein A (2013) Maintaining shortest paths under deletions in weighted directed graphs: [extended abstract]. In: STOC, Palo Alto, pp 725–734
6. Demetrescu C, Italiano GF (2004) A new approach to dynamic all pairs shortest paths. J ACM 51(6):968–992. doi:http://doi.acm.org/10.1145/1039488.1039492

7. Fredman ML, Tarjan RE (1987) Fibonacci heaps and their uses in improved network optimization algorithms. J ACM 34(3):596–615
8. Pettie S (2004) A new approach to all-pairs shortest paths on real-weighted graphs. Theor Comput Sci 312(1):47–74. doi:10.1016/S0304-3975(03)00402-X
9. Takaoka T (1992) A new upper bound on the complexity of the all pairs shortest path problem. Inf Process Lett 43(4):195–199. doi:10.1016/0020-0190(92)90200-F
10. Thorup M (2004) Fully-dynamic all-pairs shortest paths: faster and allowing negative cycles. In: Proceedings of the 9th SWAT, Humlebaek, pp 384–396
11. Thorup M, Zwick U (2005) Approximate distance oracles. J ACM 52(1):1–24
12. Thorup M, Zwick U (2006) Spanners and emulators with sublinear distance errors. In: Proceedings of the 17th SODA, Miami, pp 802–809
13. Williams R (2014) Faster all-pairs shortest paths via circuit complexity. In: STOC, New York, pp 664–673

Dynamic Trees

Renato F. Werneck
Microsoft Research Silicon Valley, La Avenida, CA, USA

Keywords

Link-cut trees

Years and Authors of Summarized Original Work

2005; Tarjan, Werneck

Problem Definition

The *dynamic tree problem* is that of maintaining an arbitrary n-vertex forest that changes over time through edge insertions (*links*) and deletions (*cuts*). Depending on the application, one associates information with vertices, edges, or both. Queries and updates can deal with individual vertices or edges, but more commonly they refer to entire paths or trees. Typical operations include finding the minimum-cost edge along a path, determining the minimum-cost vertex in a tree, or adding a constant value to the cost of each edge on a path (or of each vertex of a tree). Each of these operations, as well as *links* and *cuts*, can be performed in $O(\log n)$ time with appropriate data structures.

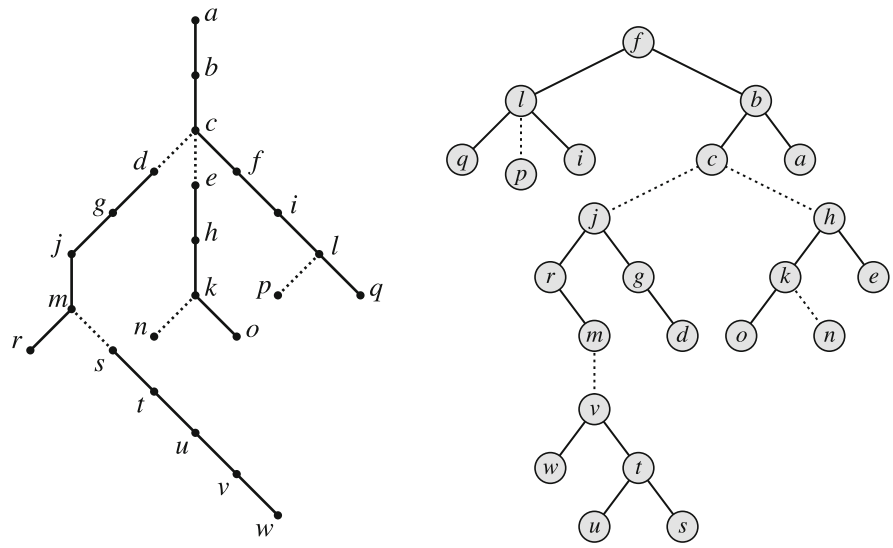
Key Results

The obvious solution to the dynamic tree problem is to represent the forest explicitly. This, however, is inefficient for queries dealing with entire paths or trees, since it would require actually traversing them. Achieving $O(\log n)$ time per operation requires mapping each (possibly unbalanced) input tree into a balanced tree, which is better suited to maintaining information about paths or trees implicitly. There are three main approaches to perform the mapping: path decomposition, tree contraction, and linearization.

Path Decomposition

The first efficient dynamic tree data structure was Sleator and Tarjan's *ST-trees* [13, 14], also known as *link-cut trees* or simply *dynamic trees*. They are meant to represent rooted trees, but the user can change the root with the *evert* operation. The data structure partitions each input tree into vertex-disjoint paths, and each path is represented as a binary search tree in which vertices appear in symmetric order. The binary trees are then connected according to how the paths are related in the forest. More precisely, the root of a binary tree becomes a *middle child* (in the data structure) of the parent (in the forest) of the topmost vertex of the corresponding path. Although a node has no more than two children (left and right) within its own binary tree, it may have arbitrarily many middle children. See Fig. 1. The path containing the root (*qlifcba* in the example) is said to be *exposed*, and is represented as the topmost binary tree. All path-related queries will refer to this path. The *expose* operation can be used to make any vertex part of the exposed path.

With standard balanced binary search trees (such as red-black trees), ST-trees support each dynamic tree operation in $O(\log^2 n)$ amortized



Dynamic Trees, Fig. 1 An ST-tree (Adapted from [14]). On the *left*, the original tree, rooted at *a* and already partitioned into paths; on the *right*, the actual data structure. *Solid edges* connect nodes on the same path; *dashed edges* connect different paths

time. This bound can be improved to $O(\log n)$ amortized with locally biased search trees, and to $O(\log n)$ in the worst case with globally biased search trees. Biased search trees (described in [5]), however, are notoriously complicated. A more practical implementation of ST-trees uses *splay trees*, a self-adjusting type of binary search trees, to support all dynamic tree operations in $O(\log n)$ amortized time [14].

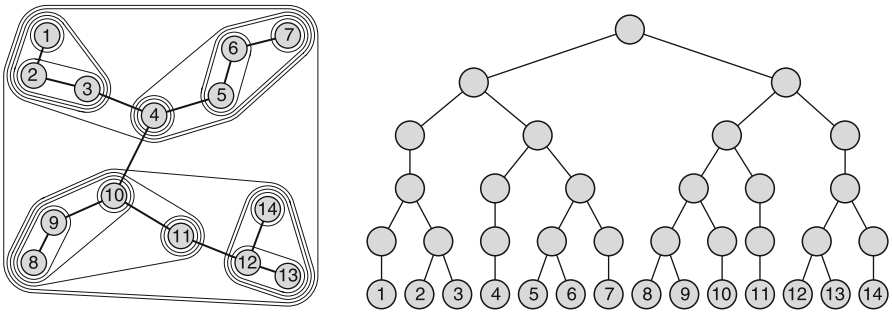
Tree Contraction

Unlike ST-trees, which represent the input trees directly, Frederickson's *topology trees* [6, 7, 8] represent a *contraction* of each tree. The original vertices constitute level 0 of the contraction. Level 1 represents a partition of these vertices into *clusters*: a degree-one vertex can be combined with its only neighbor; vertices of degree two that are adjacent to each other can be clustered together; other vertices are kept as singletons. The end result will be a smaller tree, whose own partition into clusters yields level 2. The process is repeated until a single cluster remains. The topology tree is a representation of the contraction, with each cluster having as children its constituent clusters on the level below. See Fig. 2.

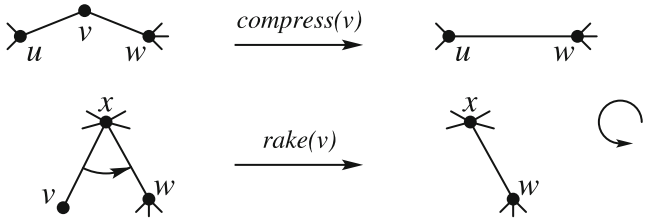
With appropriate pieces of information stored in each cluster, the data structure can be used to answer queries about the entire tree or individual paths. After a *link* or *cut*, the affected topology trees can be rebuilt in $O(\log n)$ time.

The notion of tree contraction was developed independently by Miller and Reif [11] in the context of parallel algorithms. They propose two basic operations, *rake* (which eliminates vertices of degree one) and *compress* (which eliminates vertices of degree two). They show that $O(\log n)$ rounds of these operations are sufficient to contract any tree to a single cluster. Acar et al. translated a variant of their algorithm into a dynamic tree data structure, *RC-trees* [1], which can also be seen as a randomized (and simpler) version of topology trees.

A drawback of topology trees and RC-trees is that they require the underlying forest to have vertices with bounded (constant) degree in order to ensure $O(\log n)$ time per operation. Similarly, although ST-trees do not have this limitation when aggregating information over paths, they require bounded degrees to aggregate over trees. Degree restrictions can be addressed by "ternarizing" the input forest (replacing high-degree vertices



Dynamic Trees, Fig. 2 A topology tree (Adapted from [7]). On the *left*, the original tree and its multilevel partition; on the *right*, a corresponding topology tree



Dynamic Trees, Fig. 3 The *rake* and *compress* operations, as used by top trees (From [16]))

with a series of low-degree ones [9]), but this introduces a host of special cases.

Alstrup et al.'s *top trees* [3, 4] have no such limitation, which makes them more generic than all data structures previously discussed. Although also based on tree contraction, their clusters behave not like vertices, but like *edges*. A *compress* cluster combines two edges that share a degree-two vertex, while a *rake* cluster combines an edge with a degree-one endpoint with a second edge adjacent to its other endpoint. See Fig. 3.

Top trees are designed so as to completely hide from the user the inner workings of the data structure. The user only specifies what pieces of information to store in each cluster, and (through call-back functions) how to update them after a cluster is created or destroyed when the tree changes. As long as the operations are properly defined, applications that use top trees are completely independent of how the data structure is actually implemented, i.e., of the order in which *rakes* and *compresses* are performed.

In fact, top trees were not even proposed as stand-alone data structures, but rather as an interface on top of topology trees. For efficiency reasons, however, one would rather have a more direct implementation. Holm, Tarjan, Thorup and Werneck have presented a conceptually simple stand-alone algorithm to update a top tree after a *link* or *cut* in $O(\log n)$ time in the worst case [17]. Tarjan and Werneck [16] have also introduced *self-adjusting top trees*, a more efficient implementation of top trees based on path decomposition: it partitions the input forest into edge-disjoint paths, represents these paths as splay trees, and connects these trees appropriately. Internally, the data structure is very similar to ST-trees, but the paths are edge-disjoint (instead of vertex-disjoint) and the ternarization step is incorporated into the data structure itself. All the user sees, however, are the *rakes* and *compresses* that characterize tree contraction.

Linearization

ET-trees, originally proposed by Henzinger and King [10] and later slightly simplified by Tarjan [15], use yet another approach to represent dynamic trees: *linearization*. It maintains an *Euler tour* of the each input tree, i.e., a closed path that traverses each edge twice–once in each direction. The tour induces a linear order among the vertices and arcs, and therefore can be represented as a balanced binary search tree. Linking

and cutting edges from the forest corresponds to joining and splitting the affected binary trees, which can be done in $O(\log n)$ time. While linearization is arguably the simplest of the three approaches, it has a crucial drawback: because each edge appears twice, the data structure can only aggregate information over trees, not paths.

Lower Bounds

Dynamic tree data structures are capable of solving the *dynamic connectivity* problem on acyclic graphs: given two vertices v and w, decide whether they belong to the same tree or not. P˘atraşcu and Demaine [12] have proven a lower bound of $\Omega(\log n)$ for this problem, which is matched by the data structures presented here.

Applications

Sleator and Tarjan's original application for dynamic trees was Dinic's blocking flow algorithm [13]. Dynamic trees are used to maintain a forest of arcs with positive residual capacity. As soon as the source s and the sink t become part of the same tree, the algorithm sends as much flow as possible along the s–t path; this reduces to zero the residual capacity of at least one arc, which is then *cut* from the tree. Several maximum flow and minimum-cost flow algorithms incorporating dynamic trees have been proposed ever since (some examples are [9, 15]). Dynamic tree data structures, especially those based on tree contraction, are also commonly used within dynamic graph algorithms, such as the dynamic versions of minimum spanning trees [6, 10], connectivity [10], biconnectivity [6], and bipartiteness [10]. Other applications include the evaluation of dynamic expression trees [8] and standard graph algorithms [13].

Experimental Results

Several studies have compared the performance of different dynamic-tree data structures; in most cases, ST-trees implemented with splay trees are the fastest alternative. Frederickson, for example, found that topology trees take almost 50 % more time than splay-based ST-trees when executing dynamic tree operations within a maximum flow algorithm [8]. Acar et al. [2] have shown that RC-trees are significantly slower than splay-based ST-trees when most operations are *links* and *cuts* (such as in network flow algorithms), but faster when queries and value updates are dominant. The reason is that splaying changes the structure of ST-trees even during queries, while RC-trees remain unchanged.

Tarjan and Werneck [17] have presented an experimental comparison of several dynamic tree data structures. For random sequences of *links* and *cuts*, splay-based ST-trees are the fastest alternative, followed by splay-based ET-trees, self-adjusting top trees, worst-case top trees, and RC-trees. Similar relative performance was observed in more realistic sequences of operations, except when queries far outnumber structural operations; in this case, the self-adjusting data structures are slower than RC-trees and worst-case top trees. The same experimental study also considered the "obvious" implementation of ST-trees, which represents the forest explicitly and require linear time per operation in the worst case. Its simplicity makes it significantly faster than the $O(\log n)$-time data structures for path-related queries and updates, unless paths are hundred nodes long. The sophisticated solutions are more useful when the underlying forest has high diameter or there is a need to aggregate information over trees (and not only paths).

Cross-References

- ▸ Fully Dynamic Connectivity
- ▸ Fully Dynamic Connectivity: Upper and Lower Bounds
- ▸ Fully Dynamic Higher Connectivity
- ▸ Fully Dynamic Higher Connectivity for Planar Graphs
- ▸ Fully Dynamic Minimum Spanning Trees
- ▸ Fully Dynamic Planarity Testing
- ▸ Lower Bounds for Dynamic Connectivity
- ▸ Routing

Recommended Reading

1. Acar UA, Blelloch GE, Harper R, Vittes JL, Woo SLM (2004) Dynamizing static algorithms, with applications to dynamic trees and history independence. In: Proceedings of the 15th annual ACM-SIAM symposium on discrete algorithms (SODA). SIAM, pp 524–533
2. Acar UA, Blelloch GE, Vittes JL (2005) An experimental analysis of change propagation in dynamic trees. In: Proceedings of the 7th workshop on algorithm engineering and experiments (ALENEX). pp 41–54
3. Alstrup S, Holm J, de Lichtenberg K, Thorup M (1997) Minimizing diameters of dynamic trees. In: Proceedings of the 24th international colloquium on automata, languages and programming (ICALP), Bologna, 7–11 July 1997. Lecture notes in computer science, vol 1256. Springer, pp 270–280
4. Alstrup S, Holm J, Thorup M, de Lichtenberg K (2005) Maintaining information in fully dynamic trees with top trees. ACM Trans Algorithm 1(2):243–264
5. Bent SW, Sleator DD, Tarjan RE (1985) Biased search trees. SIAM J Comput 14(3):545–568
6. Frederickson GN (1985) Data structures for on-line update of minimum spanning trees, with applications. SIAM J Comput 14(4):781–798
7. Frederickson GN (1997) Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. SIAM J Comput 26(2):484–538
8. Frederickson GN (1997) A data structure for dynamically maintaining rooted trees. J Algorithms 24(1):37–65
9. Goldberg AV, Grigoriadis MD, Tarjan RE (1991) Use of dynamic trees in a network simplex algorithm for the maximum flow problem. Math Program 50:277–290
10. Henzinger MR, King V (1997) Randomized fully dynamic graph algorithms with polylogarihmic time per operation. In: Proceedings of the 27th annual ACM symposium on theory of computing (STOC), pp 519–527
11. Miller GL, Reif JH (1985) Parallel tree contraction and its applications. In: Proceedings of the 26th annual IEEE symposium on foundations of computer science (FOCS), pp 478–489
12. Pătraşcu M, Demaine ED (2004) Lower bounds for dynamic connectivity. In: Proceedings of the 36th annual ACM symposium on theory of computing (STOC), pp 546–553
13. Sleator DD, Tarjan RE (1983) A data structure for dynamic trees. J Comput Syst Sci 26(3):362–391
14. Sleator DD, Tarjan RE (1985) Self-adjusting binary search trees. J ACM 32(3):652–686
15. Tarjan RE (1997) Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. Math Program 78:169–177
16. Tarjan RE, Werneck RF (2005) Self-adjusting top trees. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms (SODA), pp 813–822
17. Tarjan RE, Werneck RF (2007) Dynamic trees in practice. In: Proceedings of the 6th workshop on experimental algorithms (WEA). Lecture notes in computer science, vol 4525, pp 80–93
18. Werneck RF (2006) Design and analysis of data structures for dynamic trees. PhD thesis, Princeton University

E

Edit Distance Under Block Operations

S. Cenk Sahinalp
Laboratory for Computational Biology, Simon Fraser University, Burnaby, BC, USA

Keywords

Block edit distance

Years and Authors of Summarized Original Work

2000; Cormode, Paterson, Sahinalp, Vishkin
2000; Muthukrishnan, Sahinalp

Problem Definition

Given two strings $S = s_1 s_2 \ldots s_n$ and $R = r_1 r_2 \ldots r_m$ (wlog let $n \geq m$) over an alphabet $\sigma = \{\sigma_1, \sigma_2, \ldots \sigma_\ell\}$, the *standard edit distance* between S and R, denoted $ED(S, R)$ is the minimum number of *single character edits*, specifically *insertions*, *deletions* and *replacements*, to transform S into R (equivalently R into S).

If the input strings S and R are permutations of the alphabet σ (so that $|S| = |R| = |\sigma|$) then an analogous *permutation edit distance* between S and R, denoted $PED(S, R)$ can be defined as the minimum number of single character *moves*, to transform S into R (or vice versa).

A generalization of the standard edit distance is *edit distance with moves*, which, for input strings S and R is denoted $EDM(S, R)$, and is defined as the minimum number of character edits and *substring (block) moves* to transform one of the strings into the other. A move of block $s[j, k]$ to position h transforms $S = s_1 s_2 \ldots s_n$ into $S' = s_1 \ldots s_{j-1} s_{k+1} s_{k+2} \ldots s_{h-1} s_j \ldots s_k s_h \ldots s_n$ [4].

If the input strings S and R are permutations of the alphabet σ (so that $|S| = |R| = |\sigma|$) then $EDM(S, R)$ is also called as the transposition distance and is denoted $TED(S, R)$ [1].

Perhaps the most general form of the standard edit distance that involves edit operations on blocks/substrings is the *block edit distance*, denoted $BED(S, R)$. It is defined as the minimum number of single character edits, block moves, as well as *block copies and block uncopies* to transform one of the strings into the other. Copying of a block $s[j, k]$ to position h transforms $S = s_1 s_2 \ldots s_n$ into $S' = s_1 \ldots s_j s_{j+1} \ldots s_k \ldots s_{h-1} s_j \ldots s_k s_h \ldots s_n$. A block uncopy is the inverse of a block copy: it deletes a block $s[j, k]$ provided there exists $s[j', k'] = s[j, k]$ which does not overlap with $s[j, k]$ and transforms S into $S' = s_1 \ldots s_{j-1} s_{k+1} \ldots s_n$.

Throughout this discussion all edit operations have unit cost and they may overlap; i.e., a character can be edited on multiple times.

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

Key Results

There are exact and approximate solutions to computing the edit distances described above with varying performance guarantees. As can be expected, the best available running times as well as the approximation factors for computing these edit distances vary considerably with the edit operations allowed.

Exact Computation of the Standard and Permutation Edit Distance

The fastest algorithms for exactly computing the standard edit distance have been available for more than 25 years.

Theorem 1 (Levenshtein [9]) *The standard edit distance ED(S, R) can be computed exactly in time* $O(n \cdot m)$ *via dynamic programming.*

Theorem 2 (Masek-Paterson [11]) *The standard edit distance ED(S, R) can be computed exactly in time* $O(n + n \cdot m/\log^2_{|\sigma|} n)$ *via the "four-Russians trick".*

Theorem 3 (Landau-Vishkin [8]) *It is possible to compute ED(S, R) in time* $O(n \cdot ED(S, R))$.

Finally, note that if S and R are permutations of the alphabet σ, $PED(S, R)$ can be computed much faster than the standard edit distance for general strings: Observe that $PED(S, R) = n - LCS(S, R)$ where $LCS(S, R)$ represents the longest common subsequence of S and R. For permutations S, R, $LCS(S, R)$ can be computed in time $O(n \cdot \log \log n)$ [3].

Approximate Computation of the Standard Edit Distance

If some approximation can be tolerated, it is possible to considerably improve the $\tilde{O}(n \cdot m)$ time ($\tilde{O}$ notation hides polylogarithmic factors) available by the techniques above. The fastest algorithm that *approximately* computes the standard edit distance works by *embedding* strings S and R from alphabet σ into shorter strings S' and R' from a larger alphabet σ' [2]. The embedding is achieved by applying a general version of the *Locally Consistent Parsing* [13, 14] to partition the strings R and S into *consistent blocks* of size c to $2c - 1$; the partitioning is consistent in the sense that identical (long) substrings are partitioned identically. Each block is then replaced with a label such that identical blocks are identically labeled. The resulting strings S' and R' preserve the edit distance between S and R approximately as stated below.

Theorem 4 (Batu-Ergun-Sahinalp [2]) *ED(S, R) can be computed in time* $\tilde{O}(n^{1+\epsilon})$ *within an approximation factor of* $\min\{n^{\frac{1-\epsilon}{3}+o(1)}, (ED(S, R)/n^{\epsilon})^{\frac{1}{2}+o(1)}\}$.

For the case of $\epsilon = 0$, the above result provides an $\tilde{O}(n)$ time algorithm for approximating $ED(S, R)$ within a factor of $\min\{n^{\frac{1}{3}+o(1)}, ED(S, R)^{\frac{1}{2}+o(1)}\}$.

Approximate Computation of Edit Distances Involving Block Edits

For all edit distance variants described above which involve blocks, there are no known polynomial time algorithms; in fact it is NP-hard to compute $TED(S, R)$ [1], $EDM(S, R)$ and $BED(S, R)$ [10]. However, in case S and R are permutations of σ, there are polynomial time algorithms that approximate transposition distance within a constant factor:

Theorem 5 (Bafna-Pevzner [1]) *TED(S, R) can be approximated within a factor of 1.5 in* $O(n^2)$ *time.*

Furthermore, even if S and R are arbitrary strings from σ, it is possible to approximately compute both $BED(S, R)$ and $EDM(S, R)$ in near linear time. More specifically obtain an embedding of S and R to binary vectors $f(S)$ and $f(R)$ such that:

Theorem 6 (Muthukrishnan-Sahinalp [12]) $\frac{||f(S)-f(R)||_1}{\log^* n} \leq BED(S, R) \leq ||f(S) - f(R)||_1 \cdot \log n$.

In other words, the Hamming distance between $f(S)$ and $f(R)$ approximates $BED(S, R)$ within a factor of $\log n \cdot \log^* n$. Similarly for $EDM(S, R)$, it is possible to embed S and R to integer valued vectors $F(S)$ and $F(R)$ such that:

Theorem 7 (Cormode-Muthukrishnan [4]) $\frac{||F(S)-F(R)||_1}{\log^* n} \leq EDM(S,R) \leq ||F(S) - F(R)||_1 \cdot \log n$.

In other words, the L_1 distance between $F(S)$ and $F(R)$ approximates $EDM(S, R)$ within a factor of $\log n \cdot \log^* n$.

The embedding of strings S and R into binary vectors $f(S)$ and $f(R)$ is introduced in [5] and is based on the Locally Consistent Parsing described above. To obtain the embedding, one needs to hierarchically partition S and R into growing size *core* blocks. Given an alphabet σ, Locally Consistent Parsing can identify only a limited number of substrings as core blocks. Consider the lexicographic ordering of these core blocks. Each dimension i of the embedding $f(S)$ simply indicates (by setting $f(S)[i] = 1$) whether S includes the ith core block corresponding to the alphabet σ as a substring. Note that if a core block exists in S as a substring, Locally Consistent Parsing will identify it.

Although the embedding above is exponential in size, the resulting binary vector $f(S)$ is very sparse. A simple representation of $f(S)$ and $f(R)$, exploiting their sparseness can be computed in time $O(n \log^* n)$ and the Hamming distance between $f(S)$ and $f(R)$ can be computed in linear time by the use of this representation [12].

The embedding of S and R into integer valued vectors $F(S)$ and $F(R)$ are based on similar techniques. Again, the total time needed to approximate $EDM(S, R)$ within a factor of $\log n \cdot \log^* n$ is $O(n \log^* n)$.

Applications

Edit distances have important uses in computational evolutionary biology, in estimating the evolutionary distance between pairs of genome sequences under various edit operations. There are also several applications to the *document exchange problem* or *document reconciliation problem* where two copies of a text string S have been subject to edit operations (both single character and block edits) by two parties resulting in two versions S_1 and S_2, and the parties communicate to reconcile the differences between the two versions. An information theoretic lower bound on the number of bits to communicate between the two parties is then $\Omega(BED(S, R)) \cdot \log n$. The embedding of S and R to binary strings $f(S)$ and $f(R)$ provides a simple protocol [5] which gives a near-optimal tradeoff between the number of rounds of communication and the total number of bits exchanged and works with high probability.

Another important application is to the Sequence Nearest Neighbors (SNN) problem, which asks to preprocess a set of strings $S_1, \ldots, S_k$ so that given an on-line query string R, the string S_i which has the lowest distance of choice to R can be computed in time polynomial with $|R|$ and polylogarithmic with $\sum_{j=1}^{k} |S_j|$. There are no known exact solutions for the SNN problem under any edit distance considered here. However, in [12], the embedding of strings S_i into binary vectors $f(S_i)$, combined with the Approximate Nearest Neighbors results given in [6] for Hamming Distance, provides an approximate solution to the SNN problem under block edit distance as follows.

Theorem 8 (Muthukrishnan-Sahinalp [12]) *It is possible to preprocess a set of strings $S_1, \ldots, S_k$ from a given alphabet σ in $O(poly(\sum_{j=1}^{k} |S_j|))$ time such that for any on-line query string R from σ one can compute a string S_i in time $O(polylog(\sum_{j=1}^{k} |S_j|) \cdot poly(|R|))$ which guarantees that for all $h \in [1, k]$, $BED(S_i, R) \leq BED(S_h, R) \cdot \log(\max_j |S_j|) \cdot \log^*(\max_j |S_j|)$.*

Open Problems

It is interesting to note that when dealing with permutations of the alphabet σ the problem of computing both character edit distances and block edit distances become much easier; one can compute $PED(S, R)$ exactly and $TED(S, R)$ within an approximation factor of 1.5 in $\tilde{O}(n)$ time. For arbitrary strings, it is an open question whether one can approximate $TED(S, R)$ or $BED(S, R)$ within a factor of $o(\log n)$ in polynomial time.

One recent result in this direction shows that it is not possible to obtain a polylogarithmic approximation to $TED(S, R)$ via a greedy strategy [7]. Furthermore, although there is a lower bound of $\Omega(n^{\frac{1}{3}})$ on the approximation factor that can be achieved for computing the standard edit distance in $\tilde{O}(n)$ time by the use of string embeddings, there is no general lower bound on how closely one can approximate $ED(S, R)$ in near linear time.

Cross-References

▸ Approximate String Matching

Recommended Reading

1. Bafna V, Pevzner PA (1998) Sorting by transpositions. SIAM J Discret Math 11(2):224–240
2. Batu T, Ergün F, Sahinalp SC (2006) Oblivious string embeddings and edit distance approximations. In: Proceedings of the ACM-SIAM SODA, pp 792–801
3. Besmaphyatnikh S, Segal M (2000) Enumerating longest increasing subsequences and patience sorting. Inf Process Lett 76(1–2):7–11
4. Cormode G, Muthukrishnan S (2002) The string edit distance matching problem with moves. In: Proceedings of the ACM-SIAM SODA, pp 667–676
5. Cormode G, Paterson M, Sahinalp SC, Vishkin U (2000) Communication complexity of document exchange. In: Proceedings of the ACM-SIAM SODA, pp 197–206
6. Indyk P, Motwani R (1998) Approximate nearest neighbors: towards removing the curse of fimensionality. In: Proceedings of the ACM STOC, pp 604–613
7. Kaplan H, Shafrir N (2005) The greedy algorithm for shortest superstrings. Info Process Lett 93(1):13–17
8. Landau G, Vishkin U (1989) Fast parallel and serial approximate string matching. J Algorithms 10:157–169
9. Levenshtein VI (1965) Binary codes capable of correcting deletions, insertions, and reversals. Dokl Akad Nauk SSSR 163(4):845–848 (Russian). (1966) Sov Phys Dokl 10(8):707–710 (English translation)
10. Lopresti DP, Tomkins A (1997) Block edit models for approximate string matching. Theor Comput Sci 181(1):159–179
11. Masek W, Paterson M (1980) A faster algorithm for computing string edit distances. J Comput Syst Sci 20:18–31
12. Muthukrishnan S, Sahinalp SC (2000) Approximate nearest neighbors and sequence comparison with block operations. In: Proceedings of the ACM STOC, pp 416–424
13. Sahinalp SC, Vishkin U (1994) Symmetry breaking for suffix tree construction. In: ACM STOC, pp 300–309
14. Sahinalp SC, Vishkin U (1996) Efficient approximate and dynamic matching of patterns using a labeling paradigm. In: Proceeding of the IEEE FOCS, pp 320–328

Efficient Decodable Group Testing

Hung Q. Ngo[1] and Atri Rudra[2]
[1]Computer Science and Engineering, The State University of New York, Buffalo, NY, USA
[2]Department of Computer Science and Engineering, State University of New York, Buffalo, NY, USA

Keywords

Coding theory; Nonadaptive group testing; Sublinear-time decoding

Years and Authors of Summarized Original Work

2011; Ngo, Porat, Rudra

Problem Definition

The basic group testing problem is to identify the unknown set of *positive items* from a large population of *items* using as few *tests* as possible. A test is a subset of items. A test returns positive if there is a positive item in the subset. The semantics of "positives," "items," and "tests" depend on the application.

In the original context [3], group testing was invented to solve the problem of identifying syphilis-infected blood samples from a large collection of WWII draftees' blood samples. In this case, items are blood samples, which are positive if they are infected. A test is a *pool* (group) of blood samples. Testing a group of samples at a time will save resources if the test outcome is negative. On the other

hand, if the test outcome is positive, then all we know is that at least one sample in the pool is positive, but we do not know which one(s).

In *nonadaptive combinatorial group testing* (NACGT), we assume that the number of positives is at most d for some fixed integer d and that all tests have to be specified in advance before any test outcome is known. The NACGT paradigm has found numerous applications in many areas of mathematics, computer science, and computational biology [4, 9, 10].

A NACGT strategy with t tests on a universe of N items is represented by a $t \times N$ binary matrix $\mathbf{M} = (m_{ij})$, where $m_{ij} = 1$ iff item j belongs to test i. Let $\mathbf{M}_i$ and $\mathbf{M}^j$ denote row i and column j of $\mathbf{M}$, respectively. Abusing notation, we will also use $\mathbf{M}_i$ (respectively, $\mathbf{M}^j$) to denote the set of rows (respectively, columns) corresponding to the 1-entries of row i (respectively, column j). In other words, $\mathbf{M}_i$ is the ith pool, and $\mathbf{M}^j$ is the set of pools that item j belongs to.

Let $D \subset [N]$ be the unknown subset of positive items, where $|D| \leq d$. Let $\boldsymbol{y} = (y_i)_{i=1}^t \in \{0,1\}^t$ denote the test outcome vector, i.e., $y_i = 1$ iff the ith test is positive. Then, the test outcome vector is precisely the (Boolean) union of the positive columns: $\boldsymbol{y} = \bigcup_{j \in D} \mathbf{M}^j$. The task of identifying the unknown subset D from the test outcome vector $\boldsymbol{y}$ is called *decoding*.

The main problem In many modern applications of NACGT, there are two key requirements for an NACGT scheme:

1. *Small number of tests*. "Tests" are computationally expensive in many applications.
2. *Efficient decoding*. As the item universe size N can be extremely large, it would be ideal for the decoding algorithm to run in time sublinear in N and more precisely in $\text{poly}(d, \log N)$ time.

Key Results

To be able to uniquely identify an arbitrary subset D of at most d positives, it is necessary and sufficient for the test outcome vectors $\boldsymbol{y}$ to be different for distinct subsets D of at most d positives. An NACGT matrix with the above property is called *d-separable*. However, in general such matrices only admit the brute force $\Omega(N^d)$-time decoding algorithm. A very natural decoding algorithm called the *naïve decoding algorithm* runs much faster, in time $O(tN)$.

Definition 1 (Naïve decoding algorithm) Eliminate all items that participate in negative tests; return the remaining items.

This algorithm does not work for arbitrary d-separable matrices. However, if the test matrix $\mathbf{M}$ satisfies a slightly stronger property called *d-disjunct*, then the naïve decoding algorithm is guaranteed to work correctly.

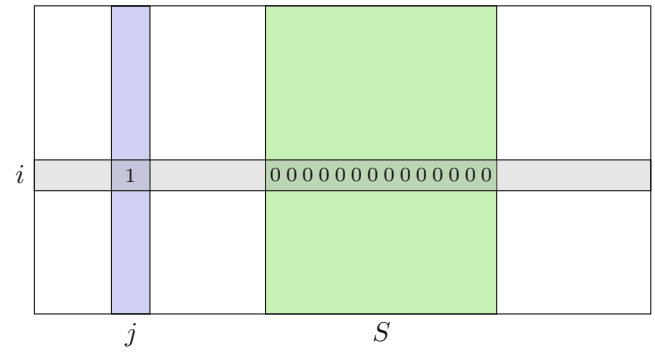
Definition 2 (Disjunct matrix) A $t \times N$ binary matrix $\mathbf{M}$ is said to be d-disjunct iff $\mathbf{M}^j \setminus \bigcup_{k \in S} \mathbf{M}^k \neq \emptyset$ for any set S of d columns and any $j \notin S$. (See Fig. 1.)

Minimize Number of Tests

It is remarkable that d-disjunct matrices not only allow for linear time decoding, which is a vast improvement over the brute-force algorithm for separable matrices, but also have asymptotically the same number of tests as d-separable matrices [4]. Let $t(d, N)$ denote the minimum number of rows of an N-column d-disjunct matrix. It has been known for about 40 years [5] that $t(\Omega(\sqrt{N}), N) = \Theta(N)$, and for $d = O(\sqrt{N})$ we have

$$\Omega\left(\frac{d^2}{\log d} \log N\right) \leq t(d, N) \leq O(d^2 \log N). \tag{1}$$

A $t \times N$ d-disjunct matrix with $t = O(d^2 \log N)$, rows can be constructed randomly or even deterministically (see [11]). However, the decoding time $O(tN)$ of the naïve decoding algorithm is still too slow for modern applications, where in most cases $d \ll N$ and thus $t \ll N$.



Efficient Decodable Group Testing, Fig. 1 A d-disjunct matrix has the following property: for any subset S of d (not necessarily contiguous) columns, and any column j that is not present in S, there exists a row i that has a 1 in column j and all zeros in S

Efficient Decoding

An ideal decoding time would be in the order of $\text{poly}(d, \log N)$, which is sublinear in N for practical ranges of d. Ngo, Porat, and Rudra [10] showed how to achieve this goal using a couple of ideas: (a) two-layer test matrix construction and (b) code concatenation using a list recoverable code.

(a) Two-layer test matrix construction The idea is to construct $\mathbf{M}$ by stacking on top of one another two matrices: a "filtering" matrix $\mathbf{F}$ and an "identification" matrix $\mathbf{D}$. (See Fig. 2.) The filtering matrix is used to quickly identify a "small" set of L candidate items including *all* the positives. Then, the identification matrix is used to pinpoint precisely the positives. For example, let $\mathbf{D}$ be any d-disjunct matrix, and that from the tests corresponding to the rows of $\mathbf{F}$, we can produce a set S of $L = \text{poly}(d, \log N)$ candidate items in time $\text{poly}(d, \log N)$. Then, by running the naïve decoding algorithm on S using test results corresponding to the rows of $\mathbf{D}$, we can identify all the positives in time $\text{poly}(d, \log N)$. To formalize the notion of "filtering matrix," we borrow a concept from coding theory, where producing a small list of candidate codewords is the *list decoding problem* [6].

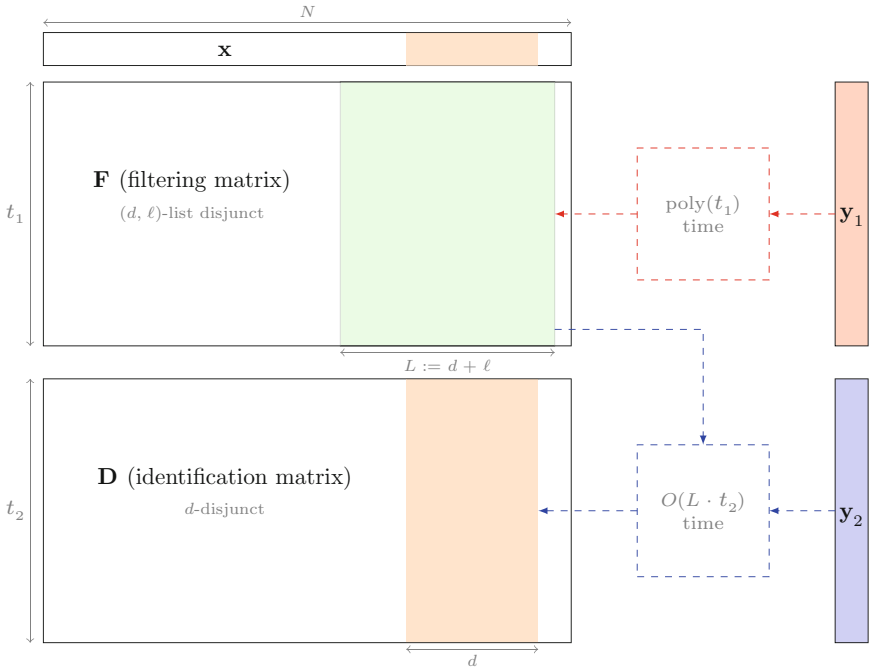
Definition 3 (List-disjunct matrix) Let $d + \ell \leq N$ be positive integers. A matrix $\mathbf{F}$ is (d, ℓ)-list disjunct if and only if $\bigcup_{j \in T} \mathbf{M}^j \setminus \bigcup_{k \in S} \mathbf{M}^k \neq \emptyset$ for any two disjoint sets S and T of columns of $\mathbf{F}$ with $|S| = d$ and $|T| = \ell$. (See Fig. 3.)

Note that a matrix is d-disjunct matrix iff it is $(d, 1)$-list disjunct. However, the relaxation to $\ell = \Theta(d)$ allows the existence (and construction) of $(d, O(d))$-list-disjunct matrices with $\Theta(d \log(N/d))$ rows. The existence of such small list-disjunct matrices is crucially used in the second idea below.

(b) Code Concatenation with list recoverable codes A $t \times N$ (d, ℓ)-list-disjunct matrix admits $O(tN)$-decoding time using the naïve decoding algorithm. However, to achieve $\text{poly}(d, \log N)$ decoding time overall, we will need to construct list-disjunct matrices that allow for a $\text{poly}(d, \log N)$ decoding time. In particular, to use such a matrix as a filtering matrix, it is necessary that $\ell = \text{poly}(d)$. To construct efficiently decodable list-disjunct matrices, we need other ideas. Ngo, Porat, and Rudra [10] used a connection to list recoverable codes [6] to construct such matrices. This connection was used to construct $(d, O(d^{3/2}))$-list-disjunct matrices with $t = o(d^2 \log_d N)$ rows that can be decoded in $\text{poly}(t)$ time. This along with the construction in Fig. 2 implies the following result:

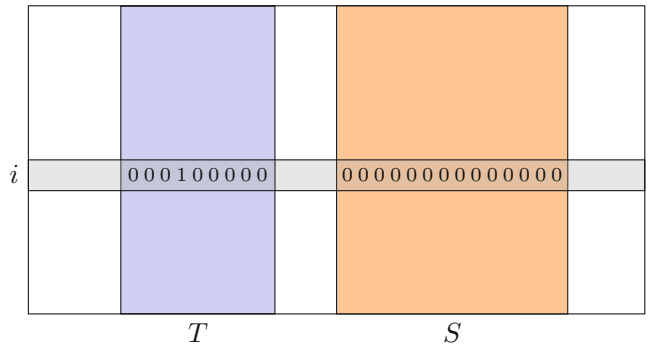
Theorem 1 ([10]) *Given any d-disjunct matrix, it can be converted into another matrix with $1 + o(1)$ times as many rows that is also efficiently decodable (even if the original matrix was not).*

Other constructions of list-disjunct matrices with worse parameters were obtained earlier by Indyk, Ngo and Rudra [7], and Cheraghchi [1] using connections to expanders and randomness extractors.



Efficient Decodable Group Testing, Fig. 2 The vector $\boldsymbol{x}$ denotes the characteristic vector of the d positives (illustrated by the *orange box*). The final matrix is the stacking of $\mathbf{F}$, which is a (d, ℓ)-list-disjunct matrix, and $\mathbf{D}$, which is a d-disjunct matrix. The result vector is naturally divided into $\boldsymbol{y}_1$ (the part corresponding to $\mathbf{F}$ and denoted by the *red vector*) and $\boldsymbol{y}_2$ (the part corresponding to $\mathbf{D}$ and denoted by the *blue vector*). The decoder first uses $\boldsymbol{y}_1$ to compute a superset of the set of positives (denoted by *green box*), which is then used with $\boldsymbol{y}_2$ to compute the final set of positives. The first step of the decoding is represented by the *red-dotted box*, while the second step (naïve decoder) is denoted by the *blue-dotted box*

Efficient Decodable Group Testing, Fig. 3 A (d, ℓ)-list-disjunct matrix satisfies the following property: for any subset S of size d and any disjoint subset T of size ℓ, there exists a row i that has a 1 in at least one column in T and all zeros in S



Applications

Heavy hitter is one of the most fundamental problems in data streaming [8]. Cormode and Muthukrishnan [2] showed that an NACGT scheme that is efficiently decodable and is also *explicit* solves a natural version of the heavy hitter problem. An explicit construction means one needs an algorithm that outputs a column or a specific entry of $\mathbf{M}$ instead of storing the entire matrix $\mathbf{M}$ which can be extremely space consuming. This is possible with Theorem 1 by picking the filtering and decoding matrices to be explicit.

Another important generalization of NACGT matrices are those that can handle errors in the

test outcomes. Again this is possible with the construction of Fig. 2 if the filtering and decoding matrices are also error tolerant. The list-disjunct matrices constructed by Cheraghchi are also error tolerant [1].

Open Problems

The outstanding open problem in group testing theory is to close the gap (1). An explicit construction of (d, d)-list-disjunct matrices is not known; solving this problem will lead to a scheme that is (near-)optimal in all desired objectives.

Recommended Reading

1. Cheraghchi M (2013) Noise-resilient group testing: Limitations and constructions. Discret Appl Math 161(1–2):81–95
2. Cormode G, Muthukrishnan S (2005) What's hot and what's not: tracking most frequent items dynamically. ACM Trans Database Syst 30(1):249–278
3. Dorfman R (1943) The detection of defective members of large populations. Ann Math Stat 14(4):436–440
4. Du DZ, Hwang FK (2000) Combinatorial group testing and its applications. Series on applied mathematics, vol 12, 2nd edn. World Scientific, River Edge
5. D'yachkov AG, Rykov VV (1982) Bounds on the length of disjunctive codes. Problemy Peredachi Informatsii 18(3):7–13
6. Guruswami V (2004) List decoding of error-correcting codes (Winning thesis of the 2002 ACM doctoral dissertation competition). Lecture notes in computer science, vol 3282. Springer
7. Indyk P, Ngo HQ, Rudra A (2010) Efficiently decodable non-adaptive group testing. In: Proceedings of the twenty first annual ACM-SIAM symposium on discrete algorithms (SODA'2010). ACM, New York, pp 1126–1142
8. Muthukrishnan S (2005) Data streams: algorithms and applications. Found Trends Theor Comput Sci 1(2)
9. Ngo HQ, Du DZ (2000) A survey on combinatorial group testing algorithms with applications to DNA library screening. In: Discrete mathematical problems with medical applications (New Brunswick, 1999). DIMACS series in discrete mathematics and theoretical computer science, vol 55. American Mathematical Society, Providence, pp 171–182
10. Ngo HQ, Porat E, Rudra A (2011) Efficiently decodable error-correcting list disjunct matrices and applications – (extended abstract). In: ICALP (1), pp 557–568
11. Porat E, Rothschild A (2011) Explicit nonadaptive combinatorial group testing schemes. IEEE Trans Inf Theory 57(12):7982–7989

Efficient Dominating and Edge Dominating Sets for Graphs and Hypergraphs

Andreas Brandstädt[1,2] and Ragnar Nevries[1]
[1]Computer Science Department, University of Rostock, Rostock, Germany
[2]Department of Informatics, University of Rostock, Rostock, Germany

Keywords

Efficient domination; Efficient edge domination; Exact cover

Years and Authors of Summarized Original Work

2010; Brandstädt, Hundt, Nevries
2011; Brandstädt, Mosca
2012; Brandstädt, Leitert, Rautenbach
2013; Brandstädt, Milanič, Nevries
2014; Brandstädt, Giakoumakis
2014; Nevries

Problem Definition

For a hypergraph $H = (V, \mathcal{E})$, a subset of edges $\mathcal{E}' \subseteq \mathcal{E}$ is an *exact cover* of H, if every vertex of V is contained in exactly one hyperedge of $\mathcal{E}'$, that is, for all $e, f \in \mathcal{E}'$ with $e \neq f$, $e \cap f = \emptyset$ and $\bigcup \mathcal{E}' = V$. The Exact Cover (XC) problem asks for the existence of an exact cover in a given hypergraph H. Exact Cover is in Karp's famous list of 21 NP-complete problems; it is NP-complete even for 3-element hyperedges (problem X3C [SP2] in [14]).

Let G be a finite simple undirected graph with vertex set V and edge set E. A vertex *dominates* itself and all its neighbors, i.e., every vertex $v \in V$ dominates its closed neighborhood $N[v] = \{u \mid u = v \text{ or } uv \in E\}$. A vertex subset D of G is an *efficient dominating* (*e.d.*) set, if, for every vertex $v \in V$, there is exactly one $d \in D$ dominating v [1, 2]. An edge subset M of G is an *efficient edge dominating* (*e.e.d.*) set, if it is an efficient dominating set in the line graph $L(G)$ of G [15]. Efficient dominating sets are sometimes also called *independent perfect dominating sets*, and efficient edge dominating sets are also known as *dominating induced matchings*.

The EFFICIENT DOMINATION (ED) problem for a graph G asks for the existence of an e.d. set in G. The EFFICIENT EDGE DOMINATION (EED) problem asks for the existence of an e.d. set in the line graph $L(G)$.

For a graph G, let $\mathcal{N}(G)$ denote its closed neighborhood hypergraph, that is, for every vertex $v \in V$, the closed neighborhood $N[v]$ is a hyperedge in $\mathcal{N}(G)$; note that this is a multiset since distinct vertices may have the same closed neighborhood. For a graph G, the *square* G^2 has the same vertex set as G and two vertices, x and y, are adjacent in G^2, if and only if their distance in G is at most 2. Note that G^2 is isomorphic to $L(\mathcal{N}(G))$.

By definition, the ED problem on a graph G is the same as the Exact Cover problem on its closed neighborhood hypergraph $\mathcal{N}(G)$, and the EED problem is the same as the Exact Cover problem on $L(\mathcal{N}(G))$.

Key Results

ED and EED are NP-complete; their complexity on special graph classes was studied in various papers – see, e.g., [2, 3, 12, 16–18, 20, 22, 24, 25] for ED and [5, 7, 11, 15, 19, 21] for EED. In particular, ED remains NP-complete for chordal graphs as well as for (very restricted) bipartite graphs such as chordal bipartite graphs, and EED is NP-complete for bipartite graphs but solvable in linear time for chordal graphs.

ED for Graphs

A key tool in [8] is a reduction of ED for G to the maximum-weight independent set problem for G^2, which is based on the following observation:

For a hypergraph $H = (V, \mathcal{E})$ and $e \in \mathcal{E}$, let $\omega(e) := |e|$ be an edge weight function. For the line graph $L(H)$, let $\alpha_\omega(L(H))$ denote the maximum weight of an independent vertex set in $L(H)$. The weight of any independent vertex set in $L(H)$ is at most $|V|$, and H has an exact cover, if and only if $\alpha_\omega(L(H)) = |V|$. Using the fact that G^2 is isomorphic to $L(\mathcal{N}(G))$ and ED on G corresponds to Exact Cover on $\mathcal{N}(G)$, this means that ED on G can be reduced to the maximum weight of an independent vertex set in G^2, similarly for EED. This unified approach helps to answer some open questions on ED and EED for graph classes; one example is ED for strongly chordal graphs: Since for a dually chordal graph G, its square G^2 is chordal, ED is solvable in polynomial time for dually chordal graphs and thus for strongly chordal graphs [8] (recall that ED is NP-complete for chordal graphs). Similar properties of powers lead to polynomial time for ED on AT-free graphs using known results [8]. For P_5-free graphs having an e.d., G^2 is P_4-free [9].

ED is NP-complete for planar bipartite graphs of maximum degree 3 [9]. In [23], this is sharpened by adding a girth condition: ED is NP-complete for planar bipartite graphs of maximum degree 3 and girth at least g, for every fixed g.

From the known results, it follows that ED is NP-complete for F-free graphs whenever F contains a cycle or a claw. Thus, F can be assumed to be cycle- and claw-free (see, e.g., [9]); such graphs F are called *linear forests*. For $(P_3 + P_3)$-free graphs and thus for P_7-free graphs, ED is NP-complete. ED is robustly solvable in time $O(nm)$ for P_5-free graphs and for $(P_4 + P_2)$-free graphs [9, 23]. For every fixed $k \geq 1$, ED is solvable in polynomial time for $(P_5 + kP_2)$-free graphs [4]. For P_6-free graphs, the complexity of ED is an open problem, and correspondingly for $(P_6 + kP_2)$-free graphs; these are the only open cases for F-free graphs.

EED for Graphs

The fact that graphs having an e.e.d. are K_4-free leads to a simple linear time algorithm for EED on chordal graphs. More generally, EED is solvable in polynomial time for hole-free graphs and thus for weakly chordal graphs and for chordal bipartite graphs [7]. This also follows from the fact that, for a weakly chordal graph G, $L(G)^2$ is weakly chordal [10] and from the reduction of EED for G to the maximum-weight independent set problem for $L(G)^2$. In [23], this is improved to a robust $O(nm)$ time algorithm for EED on hole-free graphs. In [8], we show that EED is solvable in linear time for dually chordal graphs.

One of the open problems for EED was its complexity on P_k-free graphs. In [5], we show that EED is solvable in linear time for P_7-free graphs. The complexity of EED remains open for P_k-free graphs, $k \geq 8$. In [11], EED is solved in polynomial time on claw-free graphs. EED is NP-complete for planar bipartite graphs of maximum degree 3 [7]. In [23], it is shown that EED is NP-complete for planar bipartite graphs of maximum degree 3 and girth at least g, for every fixed g.

XC, ED, and EED for Hypergraphs

The notion of α-*acyclicity* [13] is one of the most important and most frequently studied hypergraph notions. Among the many equivalent conditions describing α-acyclic hypergraphs, we take the following: For a hypergraph $H = (V, \mathcal{E})$, a tree T with node set $\mathcal{E}$ and edge set E_T is a *join tree* of H, if, for all vertices $v \in V$, the set of hyperedges $\mathcal{E}_v := \{e \in \mathcal{E} \mid v \in e\}$ containing v induces a subtree of T. H is α-*acyclic*, if it has a join tree. Let $H^* := (\mathcal{E}, \{\mathcal{E}_v \mid v \in V\})$ be the *dual hypergraph* of H. The hypergraph $H = (V, \mathcal{E})$ is a *hypertree*, if there is a tree T with vertex set V such that, for all $e \in \mathcal{E}$, $T[e]$ is connected. Obviously, H is α-acyclic, if and only if its dual H^* is a hypertree.

By a result of Duchet, Flament, and Slater (see, e.g., [6]), it is known that H is a hypertree, if and only if H has the Helly property and its line graph $L(H)$ is chordal. In its dual version, it says that H is α-acyclic, if and only if H is conformal and its 2-section graph is chordal. In [8], we show:

(i) ED and XC are NP-complete for α-acyclic hypergraphs but solvable in polynomial time for hypertrees.
(ii) EED is NP-complete for hypertrees but solvable in polynomial time for α-acyclic hypergraphs.

Recommended Reading

1. Bange DW, Barkauskas AE, Slater PJ (1988) Efficient dominating sets in graphs. In: Ringeisen RD, Roberts FS (eds) Applications of discrete mathematics. SIAM, Philadelphia, pp 189–199
2. Bange DW, Barkauskas AE, Host LH, Slater PJ (1996) Generalized domination and efficient domination in graphs. Discret Math 159:1–11
3. Biggs N (1973) Perfect codes in graphs. J Comb Theory (B) 15:289–296
4. Brandstädt A, Giakoumakis V (2014) Efficient domination for $(P_5 + kP_2)$-free graphs. Manuscript, arXiv:1407.4593v1
5. Brandstädt A, Mosca R (2011) Dominating induced matchings for P_7-free graphs in linear time. Technical report CoRR, arXiv:1106.2772v1; Extended abstract in: Asano T, Nakano S-I, Okamoto Y, Watanabe O (eds) Algorithms and computation. LNCS, vol 7074. Springer, pp 100–109
6. Brandstädt A, Le VB, Spinrad JP (1999) Graph classes: a survey. SIAM monographs on discrete mathematics and applications, vol 3. SIAM, Philadelphia
7. Brandstädt A, Hundt C, Nevries R (2010) Efficient edge domination on hole-free graphs in polynomial time. In: Lópes-Ortiz A (ed) LATIN 2010: theoretical informatics, Oaxaca. LNCS, vol 6034. Springer, pp 650–661
8. Brandstädt A, Leitert A, Rautenbach D (2012) Efficient dominating and edge dominating sets for graphs and hypergraphs. Technical report CoRR, arXiv:1207.0953v2 and in: Choa K-M, Hsu T-S, Lee D-T (eds) Algorithms and computation. LNCS, vol 7676. Springer, pp 267–277
9. Brandstädt A, Milanič M, Nevries R (2013) New polynomial cases of the weighted efficient domination problem. Technical report CoRR, arXiv:1304.6255 and in: Chatterjee K, Sgall J (eds) Mathematical foundations of computer science 2013. LNCS, vol 8087. Springer, pp 195–206

10. Cameron K, Sritharan R, Tang Y (2003) Finding a maximum induced matching in weakly chordal graphs. Discret Math 266:133–142
11. Cardoso DM, Korpelainen N, Lozin VV (2011) On the complexity of the dominating induced matching problem in hereditary classes of graphs. Discret Appl Math 159:521–531
12. Chang GJ, Pandu Rangan C, Coorg SR (1995) Weighted independent perfect domination on co-comparability graphs. Discret Appl Math 63:215–222
13. Fagin R (1983) Degrees of acyclicity for hypergraphs and relational database schemes. J ACM 30:514–550
14. Garey MR, Johnson DS (1979) Computers and intractability – a guide to the theory of NP-completeness. Freeman, San Francisco
15. Grinstead DL, Slater PL, Sherwani NA, Holmes ND (1993) Efficient edge domination problems in graphs. Inf Process Lett 48:221–228
16. Kratochvíl J (1991) Perfect codes in general graphs, Rozpravy Československé Akad. Věd Řada Mat. Přírod Vd' 7. Akademia, Praha
17. Liang YD, Lu CL, Tang CY (1997) Efficient domination on permutation graphs and trapezoid graphs. In: Jiang T, Lee DT (eds) Computing and combinatorics, Shanghai. LNCS, vol 1276. Springer, pp 232–241
18. Lin Y-L (1998) Fast algorithms for independent domination and efficient domination in trapezoid graphs. In: Chwa K-Y, Ibarra OH (eds) Algorithms and computation, Taejon. LNCS, vol 1533. Springer, pp 267–275
19. Lu CL, Tang CY (1998) Solving the weighted efficient edge domination problem on bipartite permutation graphs. Discret Appl Math 87:203–211
20. Lu CL, Tang CY (2002) Weighted efficient domination problem on some perfect graphs. Discret Appl Math 117:163–182
21. Lu CL, Ko M-T, Tang CY (2002) Perfect edge domination and efficient edge domination in graphs. Discret Appl Math 119:227–250
22. Milanič M (2011) A hereditary view on efficient domination. Extended abstract In: Adacher L, Flamini M, Leo G, Nicosia G, Pacifici A, Piccialli V (eds) Proceedings of the 10th cologne-twente workshop on graphs and combinatorial optimization, Frascati, pp 203–206. Full version: Hereditary efficiently dominatable graphs. J Graph Theory 73:400–424
23. Nevries R (2014) Efficient domination and polarity. Ph.D. thesis, University of Rostock, Germany
24. Yen C-C (1992) Algorithmic aspects of perfect domination. Ph.D. thesis, National Tsing Hua University, Taiwan
25. Yen C-C, Lee RCT (1996) The weighted perfect domination problem and its variants. Discret Appl Math 66:147–160

Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds

S.M. Yiu and Francis Y.L. Chin
Department of Computer Science, University of Hong Kong, Hong Kong, China

Keywords

Multiple global alignment; Multiple string alignment

Years and Authors of Summarized Original Work

1993; Gusfield

Problem Definition

Multiple sequence alignment is an important problem in computational biology. Applications include finding highly conserved subregions in a given set of biological sequences and inferring the evolutionary history of a set of taxa from their associated biological sequences (e.g., see [9]). There are a number of measures proposed for evaluating the goodness of a multiple alignment, but prior to this work, no efficient methods are known for computing the optimal alignment for any of these measures. The work of Gusfield [7] gives two computationally efficient multiple alignment approximation algorithms for two of the measures with approximation ratio of less than 2. For one of the measures, they also derived a randomized algorithm, which is much faster and with high probability and reports a multiple alignment with small error bounds. To the best knowledge of the entry authors, this work is the first to provide approximation algorithms (with guarantee error bounds) for this problem.

Notations and Definitions

Let X and Y be two strings of alphabet Σ. The pairwise alignment of X and Y maps X and Y into strings X' and Y' that may contain spaces, denoted by '_', where (1) $|X'| = |Y'| = \ell$ and (2) removing spaces from X' and Y' returns X and Y, respectively. The score of the alignment is defined as $d(X', Y') = \sum_{i=1}^{\ell} s(X'(i), Y'(i))$ where $X'(i)$ (and $Y'(i)$) denotes the ith character in X' (and Y') and $s(a, b)$ with $a, b \in \Sigma \cup$ '_' is the distance-based scoring scheme that satisfies the following assumptions:

1. $s(\text{'\_'}, \text{'\_'}) = 0$;
2. Triangular inequality: for any three characters, $x, y, z, s(x, z) \leq s(x, y) + s(y, z)$.

Let $\chi = X_1, X_2, \ldots, X_k$ be a set of $k > 2$ strings of alphabet Σ. A multiple alignment A of these k strings maps $X_1, X_2, \ldots, X_k$ to $X_1', X_2', \ldots, X_k'$ that may contain spaces such that (1) $|X_1'| = |X_2'| = \cdots = |X_k'| = \ell$ and (2) removing spaces from X_i' returns X_i for all $1 \leq i \leq k$. The multiple alignment A can be represented as a $k \times \ell$ matrix.

The Sum of Pairs (SP) Measure

The score of a multiple alignment A, denoted by SP(A), is defined as the sum of the scores of pairwise alignments induced by A, that is, $\sum_{i<j} d(X_i', X_j') = \sum_{i<j} \sum_{p=1}^{\ell} s(X_i'[p], X_j'[p])$ where $1 \leq i < j \leq k$.

Problem 1 (Multiple Sequence Alignment with Minimum SP Score)

INPUT: A set of k strings, a scoring scheme s.
OUTPUT: A multiple alignment A of these k strings with minimum SP(A).

The Tree Alignment (TA) Measure

In this measure, the multiple alignment is derived from an evolutionary tree. For a given set χ of k strings, let $\chi' \supseteq \chi$. An evolutionary tree T_χ' for χ is a tree with at least k nodes, where there is a one-to-one correspondence between the nodes and the strings in χ'. Let $X_u' \in \chi'$ be the string for node u. The score of T_χ', denoted by TA $\left(T_\chi'\right)$, is defined as $\sum_{e=(u,v)} D\left(X_u', X_v'\right)$ where e is an edge in T_χ' and $D\left(X_u', X_v'\right)$ denotes the score of the optimal pairwise alignment for X_u' and X_v'. Analogously, the multiple alignment of χ under the TA measure can also be represented by a $|\chi'| \times \ell$ matrix, where $|\chi'| \geq k$, with a score defined as $\sum_{e=(u,v)} d\left(X_u', X_v'\right)$ (e is an edge in T_χ'), similar to the multiple alignment under the SP measure in which the score is the summation of the alignment scores of all pairs of strings. Under the TA measure, since it is always possible to construct the $|\chi'| \times \ell$ matrix such that $d\left(X_u', X_v'\right) = D\left(X_u', X_v'\right)$ for all $e = (u, v)$ in T_χ' and we are usually interested in finding the multiple alignment with the minimum TA value, so $D\left(X_u', X_v'\right)$ is used instead of $d\left(X_u', X_v'\right)$ in the definition of TA $\left(T_\chi'\right)$.

Problem 2 (Multiple Sequence Alignment with Minimum TA Score)

INPUT: A set of k strings, a scoring scheme s.
OUTPUT: An evolutionary tree T for these k strings with minimum TA(T).

Key Results

Theorem 1 *Let A^* be the optimal multiple alignment of the given k strings with minimum SP score. They provide an approximation algorithm (the center star method) that gives a multiple alignment A such that* $\frac{SP(A)}{SP(A^*)} \leq \frac{2(k-1)}{k} = 2 - \frac{2}{k}$.

The center star method is to derive a multiple alignment which is consistent with the optimal pairwise alignments of a center string with all the other strings. The bound is derived based on the triangular inequality of the score function. The time complexity of this method is $O(k^2\ell^2)$, where ℓ^2 is the time to solve the pairwise alignment by dynamic programming and k^2 is needed to find the center string, X_c, which gives the minimum value of $\sum_{i \neq c} D(X_c, X_i)$.

Theorem 2 *Let A^* be the optimal multiple alignment of the given k strings with minimum SP score. They provide a randomized algorithm that gives a multiple alignment A such that $\frac{SP(A)}{SP(A^*)} \leq 2 + \frac{1}{r-1}$ with probability at least $1 - \left(\frac{r-1}{r}\right)^p$ for any $r > 1$ and $p \geq 1$. Instead of computing $\binom{k}{2}$ optimal pairwise alignments to find the best center string, the randomized algorithm only considers p randomly selected strings to be candidates for the best center string; thus, this method needs to x compute only $(k-1)p$ optimal pairwise alignments in $O(kp\ell^2)$ time where $1 \leq p \leq k$.*

Theorem 3 *Let T^* be the optimal evolutionary tree of the given k strings with minimum TA score. They provide an approximation algorithm that gives an evolutionary tree T such that $\frac{TA(T)}{TA(T^*)} \leq \frac{2(k-1)}{k} = 2 - \frac{2}{k}$.*

In the algorithm, they first compute all the $\binom{k}{2}$ optimal pairwise alignments to construct a graph with every node representing a distinct string X_i and the weight of each edge (X_i, X_j) as $D(X_i X_j)$. This step determines the overall time complexity $O(k^2\ell^2)$. Then, they find a minimum spanning tree from the graph. The multiple alignment has to be consistent with the optimal pairwise alignments represented by the edges of this minimum spanning tree.

Applications

Multiple sequence alignment is a fundamental problem in computational biology. In particular, multiple sequence alignment is useful in identifying those common structures, which may only be weakly reflected in the sequence and not easily revealed by pairwise alignment. These common structures may carry important information for their evolutionary history, critical conserved motifs, and common 3D molecular structure, as well as biological functions.

More recently, multiple sequence alignment is also used in revealing noncoding RNAs (ncRNAs) [2]. In this type of multiple alignment, we are not only align the underlying sequences but also the secondary structures of the RNAs. Researchers believe that ncRNAs that belong to the same family should have common components giving a similar secondary structure. The multiple alignment can help to locate and identify these common components.

Open Problems

A number of open problems related to the work of Gusfield remain open. For the SP measure, the center star method can be extended to the q-star method ($q > 2$) with approximation ratio of $2 - q/k$ [1, 10], sect. 7.5 of [11]). Whether there exists an approximation algorithm with better approximation ratio or with better time complexity is still unknown. For the TA measure, to be the best knowledge of the entry authors, the approximation ratio in Theorem 3 is currently the best result.

Another interesting direction related to this problem is the constrained multiple sequence alignment problem [12] which requires the multiple alignment to contain certain aligned characters with respect to a given constrained sequence. The best known result [6] is an approximation algorithm (also follows the idea of center star method) which gives an alignment with approximation ratio of $2 - 2/k$ for k strings.

For the complexity of the problem, Wang and Jiang [13] were the first to prove the NP-hardness of the problem with SP score under a *nonmetric* distance measure over a 4-symbol alphabet. More recently, in [5], the multiple alignment problem with SP score, star alignment, and TA score have been proved to be NP-hard for all binary or larger alphabets under *any metric*. Developing efficient approximation algorithms with good bounds for any of these measures is desirable.

Experimental Results

Two experiments have been reported in the paper showing that the worst-case error bounds in

Theorems 1 and 2 (for the SP measure) are pessimistic compared to the typical situation arising in practice.

The scoring scheme used in the experiments is $s(a,b) = 0$ if $a = b$; $s(a,b) = 1$ it either a or b is a space; otherwise $s(a,b) = 2$. Since computing the optimal multiple alignment with minimum SP score has been shown to be NP-hard, they evaluate the performance of their algorithms using the lower bound of $\sum_{i<j} D(X_i, X_j)$ (recall that $D(X_i, X_j)$ is the score of the optimal pairwise alignment of X_i and X_j). They have aligned 19 similar amino acid sequences with average length of 60 of homeoboxs from different species. The ratio of the scores of reported alignment by the center star method to the lower bound is only 1.018 which is far from the worst-case error bound given in Theorem 1. They also aligned 10 not-so-similar sequences near the homeoboxes, and the ratio of the reported alignment to the lower bound is 1.162. Results also show that the alignment obtained by the randomized algorithm is usually not far away from the lower bound.

Data Sets

The exact sequences used in the experiments are not provided.

Cross-References

▸ Statistical Multiple Alignment

Recommended Reading

1. Bafna V, Lawler EL, Pevzner PA (1997) Approximation algorithms for multiple sequence alignment. Theor Comput Sci 182:233–244
2. Dalli D, Wilm A, Mainz I, Stegar G (2006) STRAL: progressive alignment of non-coding RNA using base pairing probability vectors in quadratic time. Bioinformatics 22(13):1593–1599
3. Do C, Brudno M, Batzoglou S (2004) ProbCons: probabilistic consistency-based multiple alignment of amino acid sequences. In: Proceedings of the thirteenth national conference on artificial intelligence, pp 703–708, San Jose. AAAI Press
4. Edgar R (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res 32:1792–1797
5. Elias I (2003) Setting the intractability of multiple alignment. In: Proceedings of the 14th annual international symposium on algorithms and computation (ISAAC 2003), Kyoto, pp 352–363
6. Francis YL, Chin NLH, Lam TW, Prudence WHW (2005) Efficient constrained multiple sequence alignment with performance guarantee. J Bioinform Comput Biol 3(1):1–18
7. Gusfield D (1993) Efficient methods for multiple sequence alignment with guaranteed error bounds. Bull Math Biol 55(1):141–154
8. Notredame C, Higgins D, Heringa J (2000) T-coffee: a novel method for multiple sequence alignments. J Mol Biol 302:205–217
9. Pevsner J (2003) Bioinformatics and functional genomics. Wiley, New York
10. Pevzner PA (1992) Multiple alignment, communication cost, and graph matching. SIAM J Appl Math 52:1763–1779
11. Pevzner PA (2000) Computational molecular biology: an algorithmic approach. MIT, Cambridge
12. Tang CY, Lu CL, Chang MDT, Tsai YT, Sun YJ, Chao KM, Chang JM, Chiou YH, Wu CM, Chang HT, Chou WI (2002) Constrained multiple sequence alignment tool development and its application to RNase family alignment. In: Proceedings of the first IEEE computer society bioinformatics conference (CSB 2002), Stanford, pp 127–137
13. Wang L, Jiang T (1994) On the complexity of multiple sequence alignment. J Comput Biol 1:337–48
14. Ye Y, Cheung D, Wang Y, Yiu SM, Qing Z, Lam TW, Ting HF GLProbs: aligning multiple sequences adaptively. IEEE/ACM Trans Comput Biol Bioinformatics. doi:http://doi.ieeecomputersociety.org/10.1109/TCBB.2014.2316820

Efficient Polynomial Time Approximation Scheme for Scheduling Jobs on Uniform Processors

Klaus Jansen
Department of Computer Science, University of Kiel, Kiel, Germany

Keywords

Approximation schemes; Integer linear programming; Scheduling; Sensitivity analysis

Years and Authors of Summarized Original Work

2009; Jansen
2011; Jansen, Robenek
2014; Chen, Jansen, Zhang

Problem Definition

We consider the following fundamental problem in scheduling theory. Suppose that there is a set $\mathcal{J}$ of n independent jobs J_j with processing time p_j and a set $\mathcal{P}$ of m nonidentical processors P_i that run at different speeds s_i. If job J_j is executed on processor P_i, then processor P_i needs p_j/s_i time units to complete the job. The goal is to find an assignment $a : \mathcal{J} \to \mathcal{P}$ for the jobs to the processors that minimizes the total length of the schedule $\max_{i=1,\ldots,m} \sum_{J_j : a(J_j)=P_i} p_j/s_i$. This is the minimum time needed to complete all jobs on the processors. The problem is denoted $Q||C_{\max}$ and it is also called the minimum makespan problem on uniform parallel processors. By simplicity we may assume that the number m of processors is bounded by the number of jobs; otherwise select only the fastest n machines in $O(m)$ time.

Key Results

The scheduling problem on uniform and also identical processors is NP-hard [7] and the existence of a polynomial time algorithm for it would imply $P = NP$. Hochbaum and Shmoys [9, 10] presented a family of polynomial time approximation algorithms $\{A_\epsilon | \epsilon > 0\}$ for both scheduling problems, where each algorithm A_ϵ generates a schedule of length $(1 + \epsilon)\, OPT(I)$ for each instance I and has running time polynomial in the input size $|I|$. Such a family of algorithms is called a polynomial time approximation scheme (PTAS). It is allowed that the running time of each algorithm A_ϵ is exponential in $1/\epsilon$. The running time of the PTAS for uniform processors by Hochbaum and Shmoys [10] is $(n/\epsilon)^{O(1/\epsilon^2)}$.

Two restricted classes of approximation schemes were defined to classify different faster approximation scheme. An efficient polynomial time approximation scheme (EPTAS) is a PTAS with running time $f(1/\epsilon)\, poly(|I|)$ for some function f, while a fully polynomial time approximation scheme (FPTAS) runs in time $poly(1/\epsilon, |I|)$; polynomial in $1/\epsilon$ and the size $|I|$ of the instance. Since the scheduling problem on identical and also uniform processors is NP-hard in the strong sense (it contains bin packing as special case), we cannot hope for an FPTAS. For identical processors, Hochbaum and Shmoys (see [8]) and Alon et al. [1] gave an EPTAS with running time $f(1/\epsilon) + O(n)$, where f is doubly exponential in $1/\epsilon$.

Known Techniques

Hochbaum and Shmoys [9] introduced the dual approximation approach for identical and uniform processors and used the relationship between these scheduling problems and the bin packing problem. This relationship between scheduling on identical processors and bin packing problem had been exploited already by Coffman et al. [3]. Using the dual approximation approach, Hochbaum and Shmoys [9] proposed a PTAS for scheduling on identical processors with running time $(n/\epsilon)^{O(1/\epsilon^2)}$.

The main idea in the approach is to guess the length of the schedule by using binary search and to consider the corresponding bin packing instance with scaled identical bin size equal to 1. Then they distinguish between large items with size $> \epsilon$ and small items with size $\leq \epsilon$. For the large items they use a dynamic programming approach to calculate the minimum number of bins needed to pack them all. Afterward, they pack the remaining small items in a greedy way in enlarged bins of size $1 + \epsilon$ (i.e., they pack into any bin that currently contains items of total size at most 1; and if no such bin exists, then they open a new bin).

Furthermore, Hochbaum and Shmoys (see [8]) and Alon et al. [1] achieved an improvement to linear time by using an integer linear program for the cutting stock formulation of bin packing for the large items and a result on integer linear programming with a fixed number of variables by Lenstra [15]. This gives an EPTAS for identical

processors with running time $f(1/\epsilon) + O(n)$ where f is doubly exponential in $1/\epsilon$.

For uniform processors, the decision problem for the scheduling problem with makespan at most T can be viewed as a bin packing problem with different bin sizes. Using an ϵ-relaxed version of this bin packing problem, Hochbaum and Shmoys [10] were also able to obtain a PTAS for scheduling on uniform processors with running time $(n/\epsilon)^{O(1/\epsilon^2)}$. The main underlying idea in their algorithm is a clever rounding technique and a nontrivial dynamic programming approach over the different bins ordered by their sizes.

New Results

Recently, Jansen [11] proposed an EPTAS for scheduling jobs on uniform machines:

Theorem 1 ([11]) *There is an EPTAS (a family of algorithms $\{A_\epsilon | \epsilon > 0\}$) which, given an instance I of $Q||C_{\max}$ with n jobs and m processors with different speeds and a positive number $\epsilon > 0$, produces a schedule for the jobs of length $A_\epsilon(I) \leq (1+\epsilon)OPT(I)$. The running time of A_ϵ is*

$$2^{O(1/\epsilon^2 \log^3(1/\epsilon))} + poly(n).$$

Interestingly, the running time of the EPTAS is only single exponential in $1/\epsilon$.

Integer Linear Programming and Grouping Techniques

The new algorithm uses the dual approximation method by Hochbaum and Shmoys [10] to transform the scheduling problem into a bin packing problem with different bin sizes. Next, the input is structured by rounding bin sizes and processing times to values of the form $(1+\delta)^i$ and $\delta(1+\delta)^i$ with $i \in \mathbb{Z}$ where δ depends on ϵ. After sorting the bins according to their sizes, $c_1 \geq \ldots \geq c_m$, three groups of bins are built: $\mathcal{B}_1$ with the largest K bins (where K is constant). Let G be the smallest index such that capacity $c_{K+G+1} \leq \gamma c_K$ where $\gamma < 1$ depends on ϵ; such an index G exists for $c_m \leq \gamma c_K$. In this case $\mathcal{B}_2$ is the set of the next G largest bins where the maximum size $c_{\max}(\mathcal{B}_2) = c_{K+1}$ divided by the minimum size $c_{\min}(\mathcal{B}_2) = c_{K+G}$ is bounded by a constant $1/\gamma$ and $\mathcal{B}_3$ is the set with the remaining smaller bins of size smaller than γc_K. This generates a gap of constant size between the capacities of bins in $\mathcal{B}_1$ and $\mathcal{B}_3$. If the rate c_m/c_K, where c_m is the smallest bin size, is larger the constant γ, then a simpler instance is obtained with only two groups $\mathcal{B}_1$ and $\mathcal{B}_2$ of bins.

For $\mathcal{B}_1$ all packings for the very large items are computed (those which only fit there). If there is a feasible packing, then a mixed integer linear program (MILP) or an integer linear program (ILP) in the simpler case is used to place the other items into the bins. The placement of the large items into the second group $\mathcal{B}_2$ is done via integral configuration variables; similar to the ILP formulation for bin packing by Fernandez de la Vega and Lueker [6]. Fractional configuration variables are used for the placement of large items into $\mathcal{B}_3$. Furthermore, additional fractional variables are taken to place small items into $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$. The MILP has only a constant number of integral variables and, therefore, can be solved via the algorithm by Lenstra or Kannan [14, 15].

In order to avoid that the running time is doubly exponential in $1/\epsilon$, a recent result by Eisenbrand and Shmonin [5] about integer cones is used. To apply their result a system of equalities for the integral configuration variables is considered and the corresponding coefficients are rounded. Then each feasible solution of the modified MILP contains at most $O(1/\delta \log^2(1/\delta))$ integral variables with values larger than zero. By choosing the strictly positive integral variables in the MILP, the number of integral configuration variables is reduced from $2^{O(1/\delta \log(1/\delta))}$ to $O(1/\delta \log^2(1/\delta))$. The number of choices is bounded by $2^{O(1/\delta^2 \log^3(1/\delta))}$.

Afterward, the fractional variables in the MILP solution are rounded to integral values using ideas from scheduling job shops [13] and scheduling on unrelated machines [16]. The effect of the rounding is that most of the items can be placed directly into the bins. Only a few of them cannot be placed this way, and here is where the K largest bins and the gap between $\mathcal{B}_1$ and $\mathcal{B}_3$ come into play. It can be proved that these items can be moved to the K largest bins by increasing their sizes only slightly.

Algorithm Avoiding the MILP

Recently an EPTAS for scheduling on uniform machines is presented by Jansen and Robenek [12] that avoids the use of an MILP or ILP solver. In the new approach instead of solving (M)ILPs, an LP-relaxation and structural information about the "closest" ILP solution is used.

In the following the main techniques are described for identical processors. For a given LP-solution x, the distance to the closest ILP solution y in the infinity norm is studied, i.e., $\|x - y\|_\infty$. For the constraint matrix A_δ of the considered LP, this distance is defined by

$$\max\text{-gap}(A_\delta) := \max\{\min\{\|y^\star - x^\star\|_\infty : y^\star \text{ solution of ILP}\} : x^\star \text{solution of LP}\}.$$

Let $C(A_\delta)$ denote an upper bound for $\max\text{-gap}(A_\delta)$. The running time of the algorithm is $2^{O(1/\epsilon \log(1/\epsilon) \log(C(A_\delta)))} + poly(n)$. The algorithm for uniform processors is more complex, but we obtain a similar running time $2^{O(1/\epsilon \log(1/\epsilon) \log(C(\tilde{A}_\delta)))} + poly(n)$, where the constraint matrix $\tilde{A}_\delta$ is slightly different. For the details we refer to [12].

It can be proved using a result by Cook et al. [4] that $C(A_\delta), C(\tilde{A}_\delta) \leq 2^{O(1/\epsilon \log^2(1/\epsilon))}$. Consequently, the algorithm has a running time at most $2^{O(1/\epsilon^2 \log^3(1/\epsilon))} + poly(n)$, the same as in [11]. But, to our best knowledge, no instance is known to take on the value $2^{O(1/\epsilon \log^2(1/\epsilon))}$ for $\max\text{-gap}(A_\delta)$. We conjecture $C(A_\delta) \leq poly(1/\epsilon)$. If that holds, the running time of the algorithm would be $2^{O(1/\epsilon \log^2(1/\epsilon))} + poly(n)$ and thus improve the result in [11].

Lower Bounds

Recently, Chen, Jansen, and Zhang [2] proved the following lower bound on the running time: For scheduling on an arbitrary number of identical machines, denoted by $P||C_{\max}$, a polynomial time approximation scheme (PTAS) of running time $2^{O((1/\epsilon)^{1-\delta})} * poly(n)$ for any $\delta > 0$ would imply that the exponential time hypothesis (ETH) for 3-SAT fails.

Open Problems

The main open question is whether there is an EPTAS for scheduling jobs on identical and uniform machines with a running time $2^{O(1/\epsilon \log^c(1/\epsilon))} * poly(n)$.

Experimental Results

None is reported.

Recommended Reading

1. Alon N, Azar Y, Woeginger GJ, Yadid T (1998) Approximation schemes for scheduling on parallel machines. J Sched 1:55–66
2. Chen L, Jansen K, Zhang G (2014) On the optimality of approximation schemes for the classical scheduling problem. In: Symposium on discrete algorithms (SODA 2014), Portland
3. Coffman EG, Garey MR, Johnson DS (1978) An application of bin packing to multiprocessor scheduling. SIAM J Comput 7:1–17
4. Cook W, Gerards AMH, Schrijver A, Tardos E (1986) Sensitivity theorems in integer linear programming. Math Program 34:251–264
5. Eisenbrand F, Shmonin G (2006) Caratheodory bounds for integer cones. Oper Res Lett 34:564–568
6. Fernandez de la Vega W, Lueker GS (1981) Bin packing can be solved within $1 + \epsilon$ in linear time. Combinatorica 1:349–355
7. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman, San Francisco
8. Hochbaum DS (1997) Various notions of approximations: good, better, best, and more. In: Hochbaum DS (ed) Approximation algorithms for NP-hard problems, chap 9. Prentice Hall, Boston, pp 346–398
9. Hochbaum DS, Shmoys DB (1987) Using dual approximation algorithms for scheduling problems: practical and theoretical results. J ACM 34:144–162
10. Hochbaum DS, Shmoys DB (1988) A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. SIAM J Comput 17:539–551
11. Jansen K (2010) An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of ntegral vasriables. SIAM J Discret Math 24(2):457–485
12. Jansen K, Robenek C (2011) Scheduling jobs on identical and uniform processors revisited. In: Workshop on approximation and online algorithms, WAOA 2011. LNCS, vol 7164, pp 109–122 and Technical report, University of Kiel, Saarbrücken, TR-1109

13. Jansen K, Solis-Oba R, Sviridenko M (2003) Makespan minimization in job shops: a linear time approximation acheme. SIAM J Discret Math 16:288–300
14. Kannan R (1987) Minkowski's convex body theorem and integer programming. Math Oper Res 12:415–440
15. Lenstra HW (1983) Integer programming with a fixed number of variables. Math Oper Res 8:538–548
16. Lenstra JK, Shmoys DB, Tardos E (1990) Approximation algorithms for scheduling unrelated parallel machines. Math Program 24:259–272

Engineering Algorithms for Computational Biology

David A. Bader
College of Computing, Georgia Institute of Technology, Atlanta, GA, USA

Keywords

High-performance computational biology

Years and Authors of Summarized Original Work

2002; Bader, Moret, Warnow

Problem Definition

In the 50 years since the discovery of the structure of DNA, and with new techniques for sequencing the entire genome of organisms, biology is rapidly moving towards a data-intensive, computational science. Many of the newly faced challenges require high-performance computing, either due to the massive-parallelism required by the problem, or the difficult optimization problems that are often combinatoric and NP-hard. Unlike the traditional uses of supercomputers for regular, numerical computing, many problems in biology are irregular in structure, significantly more challenging to parallelize, and integer-based using abstract data structures.

Biologists are in search of biomolecular sequence data, for its comparison with other genomes, and because its structure determines function and leads to the understanding of biochemical pathways, disease prevention and cure, and the mechanisms of life itself. Computational biology has been aided by recent advances in both technology and algorithms; for instance, the ability to sequence short contiguous strings of DNA and from these reconstruct the whole genome and the proliferation of high-speed microarray, gene, and protein chips for the study of gene expression and function determination. These high-throughput techniques have led to an exponential growth of available genomic data.

Algorithms for solving problems from computational biology often require parallel processing techniques due to the data- and compute-intensive nature of the computations. Many problems use polynomial time algorithms (e.g., all-to-all comparisons) but have long running times due to the large number of items in the input; for example, the assembly of an entire genome or the all-to-all comparison of gene sequence data. Other problems are compute-intensive due to their inherent algorithmic complexity, such as protein folding and reconstructing evolutionary histories from molecular data, that are known to be NP-hard (or harder) and often require approximations that are also complex.

Key Results

None

Applications

Phylogeny Reconstruction

A phylogeny is a representation of the evolutionary history of a collection of organisms or genes (known as taxa). The basic assumption of process necessary to phylogenetic reconstruction is repeated divergence within species or genes. A phylogenetic reconstruction is usually depicted as a tree, in which modern taxa are depicted at the leaves and ancestral taxa occupy internal nodes, with the edges of the tree denoting evolutionary relationships among the taxa. Reconstructing phylogenies is a major component of modern

research programs in biology and medicine (as well as linguistics). Naturally, scientists are interested in phylogenies for the sake of knowledge, but such analyses also have many uses in applied research and in the commercial arena. Existing phylogenetic reconstruction techniques suffer from serious problems of running time (or, when fast, of accuracy). The problem is particularly serious for large data sets: even though data sets comprised of sequence from a single gene continue to pose challenges (e.g., some analyses are still running after 2 years of computation on medium-sized clusters), using whole-genome data (such as gene content and gene order) gives rise to even more formidable computational problems, particularly in data sets with large numbers of genes and highly-rearranged genomes.

To date, almost every model of speciation and genomic evolution used in phylogenetic reconstruction has given rise to NP-hard optimization problems. Three major classes of methods are in common use. Heuristics (a natural consequence of the NP-hardness of the problems) run quickly, but may offer no quality guarantees and may not even have a well-defined optimization criterion, such as the popular *neighbor-joining* heuristic [9]. Optimization based on the criterion of *maximum parsimony* (MP) [4] seeks the phylogeny with the least total amount of change needed to explain modern data. Finally, optimization based on the criterion of *maximum likelihood* (ML) [5] seeks the phylogeny that is the most likely to have given rise to the modern data.

Heuristics are fast and often rival the optimization methods in terms of accuracy, at least on datasets of moderate size. Parsimony-based methods may take exponential time, but, at least for DNA and amino acid data, can often be run to completion on datasets of moderate size. Methods based on maximum likelihood are very slow (the point estimation problem alone appears intractable) and thus restricted to very small instances, and also require many more assumptions than parsimony-based methods, but appear capable of outperforming the others in terms of the quality of solutions when these assumptions are met. Both MP- and ML-based analyses are often run with various heuristics to ensure timely termination of the computation, with mostly unquantified effects on the quality of the answers returned.

Thus there is ample scope for the application of high-performance algorithm engineering in the area. As in all scientific computing areas, biologists want to study a particular dataset and are willing to spend months and even years in the process: accurate branch prediction is the main goal. However, since all exact algorithms scale exponentially (or worse, in the case of ML approaches) with the number of taxa, speed remains a crucial parameter – otherwise few datasets of more than a few dozen taxa could ever be analyzed.

Experimental Results

As an illustration, this entry briefly describes a high-performance software suite, GRAPPA (Genome Rearrangement Analysis through Parsimony and other Phylogenetic Algorithms) developed by Bader et al. *GRAPPA* extends Sankoff and Blanchette's breakpoint phylogeny algorithm [10] into the more biologically-meaningful inversion phylogeny and provides a highly-optimized code that can make use of distributed- and shared-memory parallel systems (see [1, 2, 6, 7, 8, 11] for details). In [3], Bader et al. gives the first linear-time algorithm and fast implementation for computing inversion distance between two signed permutations. *GRAPPA* was run on a 512-processor IBM Linux cluster with Myrinet and obtained a 512-fold speedup (linear speedup with respect to the number of processors): a complete breakpoint analysis (with the more demanding inversion distance used in lieu of breakpoint distance) for the 13 genomes in the Campanulaceae data set ran in less than 1.5 h in an October 2000 run, for a *million-fold* speedup over the original implementation. The latest version features significantly improved bounds and new distance correction methods and, on the same dataset, exhibits a speedup factor of *over one billion*. GRAPPA achieves this speedup through a combination of parallelism and high-performance algorithm engineering.

Although such spectacular speedups will not always be realized, many algorithmic approaches now in use in the biological, pharmaceutical, and medical communities may benefit tremendously from such an application of high-performance techniques and platforms.

This example indicates the potential of applying high-performance algorithm engineering techniques to applications in computational biology, especially in areas that involve complex optimizations: Bader's reimplementation did not require new algorithms or entirely new techniques, yet achieved gains that turned an impractical approach into a usable one.

Cross-References

- Distance-Based Phylogeny Reconstruction (Fast-Converging)
- Distance-Based Phylogeny Reconstruction: Safety and Edge Radius
- Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds
- High Performance Algorithm Engineering for Large-Scale Problems
- Local Alignment (with Affine Gap Weights)
- Local Alignment (with Concave Gap Weights)
- Multiplex PCR for Gap Closing (Whole-Genome Assembly)
- Peptide De Novo Sequencing with MS/MS
- Perfect Phylogeny Haplotyping
- Phylogenetic Tree Construction from a Distance Matrix
- Sorting Signed Permutations by Reversal (Reversal Distance)
- Sorting Signed Permutations by Reversal (Reversal Sequence)
- Sorting by Transpositions and Reversals (Approximate Ratio 1.5)
- Substring Parsimony

Recommended Reading

1. Bader DA, Moret BME, Warnow T, Wyman SK, Yan M (2001) High-performance algorithm engineering for gene-order phylogenies. In: DIMACS workshop on whole genome comparison. Rutgers University, Piscataway
2. Bader DA, Moret BME, Vawter L (2001) Industrial applications of high-performance computing for phylogeny reconstruction. In: Siegel HJ (ed) Proceedings of the SPIE commercial applications for high-performance computing, vol 4528. Denver, pp 159–168
3. Bader DA, Moret BME, Yan M (2001) A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. J Comput Biol 8(5):483–491
4. Farris JS (1983) The logical basis of phylogenetic analysis. In: Platnick NI, Funk VA (eds) Advances in cladistics. Columbia University Press, New York, pp 1–36
5. Felsenstein J (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. J Mol Evol 17:368–376
6. Moret BME, Bader DA, Warnow T, Wyman SK, Yan M (2001) GRAPPA: a high performance computational tool for phylogeny reconstruction from gene-order data. In: Proceedings of the botany, Albuquerque
7. Moret BME, Bader DA, Warnow T (2002) High-performance algorithm engineering for computational phylogenetics. J Supercomput 22:99–111, Special issue on the best papers from ICCS'01
8. Moret BME, Wyman S, Bader DA, Warnow T, Yan M (2001) A new implementation and detailed study of breakpoint analysis. In: Proceedings of the 6th Pacific symposium biocomputing (PSB 2001), Hawaii, Jan 2001, pp 583–594
9. Saitou N, Nei M (1987) The neighbor-joining method: a new method for reconstruction of phylogenetic trees. Mol Biol Evol 4:406–425
10. Sankoff D, Blanchette M (1998) Multiple genome rearrangement and breakpoint phylogeny. J Comput Biol 5:555–570
11. Yan M (2004) High performance algorithms for phylogeny reconstruction with maximum parsimony. PhD thesis, Electrical and Computer Engineering Department, University of New Mexico, Albuquerque, Jan 2004

Engineering Algorithms for Large Network Applications

Christos Zaroliagis
Department of Computer Engineering and Informatics, University of Patras, Patras, Greece

Years and Authors of Summarized Original Work

2002; Schulz, Wagner, Zaroliagis

Problem Definition

Dealing effectively with applications in large networks, it typically requires the efficient solution of one ore more underlying algorithmic problems. Due to the size of the network, a considerable effort is inevitable in order to achieve the desired efficiency in the algorithm.

One of the primary tasks in large network applications is to answer queries for finding best routes or paths as efficiently as possible. Quite often, the challenge is to process a vast number of such queries on-line: a typical situation encountered in several real-time applications (e.g., traffic information systems, public transportation systems) concerns a query-intensive scenario, where a central server has to answer a huge number of on-line customer queries asking for their best routes (or optimal itineraries). The main goal in such an application is to reduce the (average) response time for a query.

Answering a best route (or optimal itinerary) query translates in computing a minimum cost (shortest) path on a suitably defined directed graph (digraph) with nonnegative edge costs. This in turn implies that the core algorithmic problem underlying the efficient answering of queries is the single-source single-target shortest path problem.

Although the straightforward approach of precomputing and storing shortest paths for all pairs of vertices would enabling the optimal answering of shortest path queries, the quadratic space requirements for digraphs with more than 10^5 vertices makes such an approach prohibitive for large and very large networks. For this reason, the main goal of almost all known approaches is to keep the space requirements as small as possible. This in turn implies that one can afford a heavy (in time) preprocessing, which does not blow up space, in order to speed-up the query time.

The most commonly used approach for answering shortest path queries employs Dijkstra's algorithm and/or variants of it. Consequently, the main challenge is how to reduce the algorithm's *search-space* (number of vertices visited), as this would immediately yield a better query time.

Key Results

All results discussed concern answering of *optimal* (or *exact* or *distance-preserving*) shortest paths under the aforementioned query-intensive scenario, and are all based on the following generic approach. A preprocessing of the input network $G = (V, E)$ takes place that results in a data structure of size $O(|V| + |E|)$ (i.e., linear to the size of G). The data structure contains additional information regarding certain shortest paths that can be used later during querying.

Depending on the pre-computed additional information as well as on the way a shortest path query is answered, two approaches can be distinguished. In the first approach, *graph annotation*, the additional information is attached to vertices or edges of the graph. Then, speed-up techniques to Dijkstra's algorithm are employed that, based on this information, decide quickly which part of the graph does not need to be searched. In the second approach, an *auxiliary graph* G' is constructed hierarchically. A shortest path query is then answered by searching only a small part of G', using Dijkstra's algorithm enhanced with heuristics to further speed-up the query time.

In the following, the key results of the first [3, 4, 9, 11] and the second approach [1, 2, 5, 7, 8, 10] are discussed, as well as results concerning modeling issues.

First Approach: Graph Annotation

The first work under this approach concerns the study in [9] on large railway networks. In that paper, two new heuristics are introduced: the *angle-restriction* (that tries to reduce the search space by taking advantage of the geometric layout of the vertices) and the *selection of stations* (a subset of vertices is selected among which all pairs shortest paths are pre-computed). These two heuristics along with a combination of the classical *goal-directed* or A^* *search* turned out to be rather efficient. Moreover, they motivated two important generalizations [10, 11] that gave further improvements to shortest path query times.

The full exploitation of geometry-based heuristics was investigated in [11], where both street and railway networks are considered. In that paper, it is shown that the search space of Dijkstra's algorithm can be significantly reduced (to 5–10 % of the initial graph size) by extracting geometric information from a given layout of the graph and by encapsulating pre-computed shortest path information in resulted geometric objects, called *containers*. Moreover, the dynamic case of the problem was investigated, where edge costs are subject to change and the geometric containers have to be updated.

A powerful modification to the classical Dijkstra's algorithm, called *reach-based routing*, was presented in [4]. Every vertex is assigned a so-called *reach value* that determines whether a particular vertex will be considered during Dijkstra's algorithm. A vertex is excluded from consideration if its reach value is small; that is, if it does not contribute to any path long enough to be of use for the current query.

A considerable enhancement of the classical A^* *search* algorithm using landmarks (selected vertices like in [9, 10]) and the triangle inequality with respect to the shortest path distances was shown in [3]. Landmarks and triangle inequality help to provide better lower bounds and hence boost A^* search.

Second Approach: Auxiliary Graph

The first work under this approach concerns the study in [10], where a new hierarchical decomposition technique is introduced called *multi-level graph*. A multi-level graph $\mathcal{M}$ is a digraph which is determined by a sequence of subsets of V and which extends E by adding multiple levels of edges. This allows to efficiently construct, during querying, a subgraph of $\mathcal{M}$ which is substantially smaller than G and in which the shortest path distance between any of its vertices is equal to the shortest path distance between the same vertices in G. Further improvements of this approach have been presented recently in [1]. A refinement of the above idea was introduced in [5], where the multi-level overlay graphs are introduced. In such a graph, the decomposition hierarchy is not determined by application-specific information as it happens in [9, 10].

An alternative hierarchical decomposition technique, called *highway hierarchies*, was presented in [7]. The approach takes advantage of the inherent hierarchy possessed by real-world road networks and computes a hierarchy of coarser views of the input graph. Then, the shortest path query algorithm considers mainly the (much smaller in size) coarser views, thus achieving dramatic speed-ups in query time. A revision and improvement of this method was given in [8]. A powerful combination of the highway hierarchies with the ideas in [3] was reported in [2].

Modeling Issues

The modeling of the original best route (or optimal itinerary) problem on a large network to a shortest path problem in a suitably defined directed graph with appropriate edge costs also plays a significant role in reducing the query time. Modeling issues are thoroughly investigated in [6]. In that paper, the first experimental comparison of two important approaches (time-expanded versus time-dependent) is carried out, along with new extensions of them towards realistic modeling. In addition, several new heuristics are introduced to speed-up query time.

Applications

Answering shortest path queries in large graphs has a multitude of applications, especially in traffic information systems under the aforementioned scenario; that is, a central server has to answer, as fast as possible, a huge number of on-line customer queries asking for their best routes or itineraries. Other applications of the above scenario involve route planning systems for cars, bikes and hikers, public transport systems for itinerary information of scheduled vehicles (like trains or buses), answering queries

in spatial databases, and web searching. All the above applications concern real-time systems in which users continuously enter their requests for finding their best connections or routes. Hence, the main goal is to reduce the (average) response time for answering a query.

Open Problems

Real-world networks increase constantly in size either as a result of accumulation of more and more information on them, or as a result of the digital convergence of media services, communication networks, and devices. This scaling-up of networks makes the scalability of the underlying algorithms questionable. As the networks continue to grow, there will be a constant need for designing faster algorithms to support core algorithmic problems.

Experimental Results

All papers discussed in section "Key Results" contain important experimental studies on the various techniques they investigate.

Data Sets

The data sets used in [6, 11] are available from http://lso-compendium.cti.gr/ under problems 26 and 20, respectively.

The data sets used in [1, 2] are available from http://www.dis.uniroma1.it/~challenge9/.

URL to Code

The code used in [9] is available from http://doi.acm.org/10.1145/351827.384254.

The code used in [6, 11] is available from http://lso-compendium.cti.gr/ under problems 26 and 20, respectively.

The code used in [3] is available from http://www.avglab.com/andrew/soft.html.

Cross-References

- Implementation Challenge for Shortest Paths
- Shortest Paths Approaches for Timetable Information

Recommended Reading

1. Delling D, Holzer M, Müller K, Schulz F, Wagner D (2006) High-performance multi-level graphs. In: 9th DIMACS challenge on shortest paths, Rutgers University, Nov 2006
2. Delling D, Sanders P, Schultes D, Wagner D (2006) Highway hierarchies star. In: 9th DIMACS challenge on shortest paths, Rutgers University, Nov 2006
3. Goldberg AV, Harrelson C (2005) Computing the shortest path: *A** search meets graph theory. In: Proceedings of the 16th ACM-SIAM symposium on discrete algorithms – SODA. ACM, New York/SIAM, Philadelphia, pp 156–165
4. Gutman R (2004) Reach-based routing: a new approach to shortest path algorithms optimized for road networks. In: Algorithm engineering and experiments – ALENEX (SIAM, 2004). SIAM, Philadelphia, pp 100–111
5. Holzer M, Schulz F, Wagner D (2006) Engineering multi-level overlay graphs for shortest-path queries. In: Algorithm engineering and experiments – ALENEX (SIAM, 2006). SIAM, Philadelphia, pp 156–170
6. Pyrga E, Schulz F, Wagner D, Zaroliagis C (2007) Efficient models for timetable information in public transportation systems. ACM J Exp Algorithmic 12(2.4):1–39
7. Sanders P, Schultes D (2005) Highway hierarchies hasten exact shortest path queries. In: Algorithms – ESA 2005. Lecture notes in computer science, vol 3669. pp 568–579
8. Sanders P, Schultes D (2006) Engineering highway hierarchies. In: Algorithms – ESA 2006. Lecture notes in computer science, vol 4168. pp 804–816
9. Schulz F, Wagner D, Weihe K (2000) Dijkstra's algorithm on-line: an empirical case study from public railroad transport. ACM J Exp Algorithmics 5(12): 1–23
10. Schulz F, Wagner D, Zaroliagis C (2002) Using multi-level graphs for timetable information in railway systems. In: Algorithm engineering and experiments – ALENEX 2002. Lecture notes in computer science, vol 2409. pp 43–59
11. Wagner D, Willhalm T, Zaroliagis C (2005) Geometric containers for efficient shortest path computation. ACM J Exp Algorithm 10(1.3):1–30

Engineering Geometric Algorithms

Dan Halperin
School of Computer Science, Tel-Aviv University, Tel Aviv, Israel

Keywords

Certified and efficient implementation of geometric algorithms; Geometric computing with certified numerics and topology

Years and Authors of Summarized Original Work

2004; Halperin

Problem Definition

Transforming a theoretical geometric algorithm into an effective computer program abounds with hurdles. Overcoming these difficulties is the concern of *engineering geometric algorithms*, which deals, more generally, with the design and implementation of certified and efficient solutions to algorithmic problems of geometric nature. Typical problems in this family include the construction of Voronoi diagrams, triangulations, arrangements of curves and surfaces (namely, space subdivisions), two- or higher-dimensional search structures, convex hulls and more.

Geometric algorithms strongly couple topological/combinatorial structures (e.g., a graph describing the triangulation of a set of points) on the one hand, with numerical information (e.g., the coordinates of the vertices of the triangulation) on the other. Slight errors in the numerical calculations, which in many areas of science and engineering can be tolerated, may lead to detrimental mistakes in the topological structure, causing the computer program to crash, to loop infinitely, or plainly to give wrong results.

Straightforward implementation of geometric algorithms as they appear in a textbook, using standard machine arithmetic, is most likely to fail. This entry is concerned only with *certified* solutions, namely, solutions that are guaranteed to construct the exact desired structure or a good approximation of it; such solutions are often referred to as *robust*.

The goal of engineering geometric algorithms can be restated as follows: *Design and implement geometric algorithms that are at once robust and efficient in practice*.

Much of the difficulty in adapting in practice the existing vast algorithmic literature in computational geometry comes from the assumptions that are typically made in the theoretical study of geometric algorithms that (1) the input is in general position, namely, degenerate input is precluded, (2) computation is performed on an ideal computer that can carry out real arithmetic to infinite precision (so-called real RAM), and (3) the cost of operating on a small number of simple geometric objects is "unit" time (e.g., equal cost is assigned to intersecting three spheres and to comparing two integer numbers).

Now, in real life, geometric input is quite often degenerate, machine precision is limited, and operations on a small number of simple geometric objects within the same algorithm may differ 100-fold and more in the time they take to execute (when aiming for certified results). Just implementing an algorithm carefully may not suffice and often redesign is called for.

Key Results

Tremendous efforts have been invested in the design and implementation of robust computational-geometry software in recent years. Two notable large-scale efforts are the CGAL library [1] and the geometric part of the LEDA library [14]. These are jointly reviewed in the survey by Kettner and Näher [13]. Numerous other relevant projects, which for space constraints are not reviewed here, are surveyed by Joswig [12] with extensive references to papers and Web sites.

A fundamental engineering decision to take when coming to implement a geometric

algorithm is what will the underlying arithmetic be, that is, whether to opt for exact computation or use the machine floating-point arithmetic. (Other less commonly used options exist as well.) To date, the CGAL and LEDA libraries are almost exclusively based on exact computation. One of the reasons for this exclusivity is that exact computation emulates the ideal computer (for restricted problems) and makes the adaptation of algorithms from theory to software easier. This is facilitated by major headway in developing tools for efficient computation with rational or algebraic numbers (GMP [3], LEDA [14], CORE [2] and more). On top of these tools, clever techniques for reducing the amount of exact computation were developed, such as floating-point filters and the higher-level geometric filtering.

The alternative is to use the machine floating-point arithmetic, having the advantage of being very fast. However, it is nowhere near the ideal infinite precision arithmetic assumed in the theoretical study of geometric algorithms and algorithms have to be carefully redesigned. See, for example, the discussion about imprecision in the manual of QHULL, the convex hull program by Barber et al. [5]. Over the years a variety of specially tailored floating-point variants of algorithms have been proposed, for example, the carefully crafted VRONI package by Held [11], which computes the Voronoi diagram of points and line segments using standard floating-point arithmetic, based on the topology-oriented approach of Sugihara and Iri. While VRONI works very well in practice, it is not theoretically certified. *Controlled perturbation* [9] emerges as a systematic method to produce certified approximations of complex geometric constructs while using floating-point arithmetic: the input is perturbed such that all predicates are computed accurately even with the limited-precision machine arithmetic, and a method is given to bound the necessary magnitude of perturbation that will guarantee the successful completion of the computation.

Another decision to take is how to represent the output of the algorithm, where the major issue is typically how to represent the coordinates of vertices of the output structure(s). Interestingly, this question is crucial when using exact computation since there the output coordinates can be prohibitively large or simply impossible to finitely enumerate. (One should note though that many geometric algorithms are *selective* only, namely, they do not produce new geometric entities but just select and order subsets of the input coordinates. For example, the output of an algorithm for computing the convex hull of a set of points in the plane is an ordering of a subset of the input points. No new point is computed. The discussion in this paragraph mostly applies to algorithms that output new geometric constructs, such as the intersection point of two lines.) But even when using floating-point arithmetic, one may prefer to have a more compact bit-size representation than, say, machine doubles. In this direction there is an effective, well-studied solution for the case of polygonal objects in the plane, called *snap rounding*, where vertices and intersection points are snapped to grid vertices while retaining certain topological properties of the exact desired structure. Rounding with guarantees is in general a very difficult problem, and already for polyhedral objects in 3-space the current attempts at generalizing snap rounding are very costly (increasing the complexity of the rounded objects to the third, or even higher, power).

Then there are a variety of engineering issues depending on the problem at hand. Following are two examples of engineering studies where the experience in practice is different from what the asymptotic resource measures imply. The examples relate to fundamental steps in many geometric algorithms: decomposition and point location.

Decomposition

A basic step in many geometric algorithms is to decompose a (possibly complex) geometric object into simpler subobjects, where each subobject typically has constant descriptive complexity. A well-known example is the triangulation of a polygon. The choice of decomposition may have a significant effect on the efficiency in practice of various algorithms that rely on decomposition. Such is the case

when constructing Minkowski sums of polygons in the plane. The Minkowski sum of two sets A and B in $\mathbb{R}^d$ is the vector sum of the two sets $A \oplus B = \{a + b | a \in A, b \in B\}$. The simplest approach to computing Minkowski sums of two polygons in the plane proceeds in three steps: triangulate each polygon, then compute the sum of each triangle of one polygon with each triangle of the other, and finally take the union of all the subsums. In asymptotic measures, the choice of triangulation (over alternative decompositions) has no effect. In practice though, triangulation is probably the worst choice compared with other convex decompositions, even fairly simple heuristic ones (not necessarily optimal), as shown by experiments on a dozen different decomposition methods [4]. The explanation is that triangulation increases the overall complexity of the subsums and in turn makes the union stage more complex – indeed by a constant factor, but a noticeable factor in practice. Similar phenomena were observed in other situations as well. For example, when using the prevalent vertical decomposition of arrangements – often it is too costly compared with sparser decompositions (i.e., decompositions that add fewer extra features).

Point Location

A recurring problem in geometric computing is to process given planar subdivision (planar map), so as to efficiently answer *point-location* queries: Given a point q in the plane, which face of the map contains q? Over the years a variety of point-location algorithms for planar maps were implemented in CGAL, in particular, a hierarchical search structure that guarantees logarithmic query time after expected $O(n \log n)$ preprocessing time of a map with n edges. This algorithm is referred to in CGAL as the *RIC* point-location algorithm after the preprocessing method which uses randomized incremental construction. Several simpler, easier-to-program algorithms for point location were also implemented. None of the latter beats the RIC algorithm in query time. However, the RIC is by far the slowest of all the implemented algorithms in terms of preprocessing, which in many scenarios renders it less effective. One of the simpler methods devised is a variant of the well-known *jump-and-walk* approach to point location. The algorithm scatters points (so-called *landmarks*) in the map and maintains the landmarks (together with their containing faces) in a nearest-neighbor search structure. Once a query q is issued it finds the nearest landmark ℓ to q, and "walks" in the map from ℓ toward q along the straight line segment connecting them. This landmark approach offers query time that is only slightly more expensive than the RIC method while being very efficient in preprocessing. The full details can be found in [10]. This is yet another consideration when designing (geometric) algorithms: the cost of preprocessing (and storage) versus the cost of a query. Quite often the effective (practical) tradeoff between these costs needs to be deduced experimentally.

Applications

Geometric algorithms are useful in many areas. Triangulations and arrangements are examples of basic constructs that have been intensively studied in computational geometry, carefully implemented and experimented with, as well as used in diverse applications.

Triangulations

Triangulations in two and three dimensions are implemented in CGAL [7]. In fact, CGAL offers many variants of triangulations useful for different applications. Among the applications where CGAL triangulations are employed are meshing, molecular modeling, meteorology, photogrammetry, and geographic information systems (GIS). For other available triangulation packages, see the survey by Joswig [12].

Arrangements

Arrangements of curves in the plane are supported by CGAL [15], as well as en-

velopes of surfaces in three-dimensional space. Forthcoming is support also for arrangements of curves on surfaces. CGAL arrangements have been used in motion planning algorithms, computer-aided design and manufacturing, GIS, computer graphics, and more (see Chap. 1 in [6]).

Open Problems

In spite of the significant progress in certified implementation of effective geometric algorithms, the existing theoretical algorithmic solutions for many problems still need adaptation or redesign to be useful in practice. One example where progress can have wide repercussions is devising effective decompositions for curved geometric objects (e.g., arrangements) in the plane and for higher-dimensional objects. As mentioned earlier, suitable decompositions can have a significant effect on the performance of geometric algorithms in practice.

Certified fixed-precision geometric computing lags behind the exact computing paradigm in terms of available robust software, and moving forward in this direction is a major challenge. For example, creating a certified floating-point counterpart to CGAL is a desirable (and highly intricate) task.

Another important tool that is largely missing is consistent and efficient rounding of geometric objects. As mentioned earlier, a fairly satisfactory solution exists for polygonal objects in the plane. Good techniques are missing for curved objects in the plane and for higher-dimensional objects (both linear and curved).

URL to Code

http://www.cgal.org

Cross-References

- ▸ LEDA: a Library of Efficient Algorithms
- ▸ Robust Geometric Computation

Recommended Reading

Conferences publishing papers on the topic include the ACM Symposium on Computational Geometry (SoCG), the Workshop on Algorithm Engineering and Experiments (ALENEX), the Engineering and Applications Track of the European Symposium on Algorithms (ESA), its predecessor and the Workshop on Experimental Algorithms (WEA). Relevant journals include the *ACM Journal on Experimental Algorithmics, Computational Geometry: Theory and Applications* and the *International Journal of Computational Geometry and Applications*. A wide range of relevant aspects are discussed in the recent book edited by Boissonnat and Teillaud [6], titled *Effective Computational Geometry for Curves and Surfaces*.

1. The CGAL project homepage. http://www.cgal.org/. Accessed 6 Apr 2008
2. The CORE library homepage. http://www.cs.nyu.edu/exact/core/. Accessed 6 Apr 2008
3. The GMP webpage. http://gmplib.org/. Accessed 6 Apr 2008
4. Agarwal PK, Flato E, Halperin D (2002) Polygon decomposition for efficient construction of Minkowski sums. Comput Geom Theory Appl 21(1–2):39–61
5. Barber CB, Dobkin DP, Huhdanpaa HT (2008) Imprecision in QHULL. http://www.qhull.org/html/qh-impre.htm. Accessed 6 Apr 2008
6. Boissonnat J-D, Teillaud M (eds) (2006) Effective computational geometry for curves and surfaces. Springer, Berlin
7. Boissonat J-D, Devillers O, Pion S, Teillaud M, Yvinec M (2002) Triangulations in CGAL. Comput Geom Theory Appl 22(1–3):5–19
8. Fabri A, Giezeman G-J, Kettner L, Schirra S, Schönherr S (2000) On the design of CGAL a computational geometry algorithms library. Softw Pract Exp 30(11):1167–1202
9. Halperin D, Leiserowitz E (2004) Controlled perturbation for arrangements of circles. Int J Comput Geom Appl 14(4–5):277–310
10. Haran I, Halperin D (2006) An experimental study of point location in general planar arrangements. In: Proceedings of 8th workshop on algorithm engineering and experiments, Miami, pp 16–25
11. Held M (2001) VRONI: an engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. Comput Geom Theory Appl 18(2):95–123
12. Joswig M (2004) Software. In: Goodman JE, O'Rourke J (eds) Handbook of discrete and computational geometry, chapter 64, 2nd edn. Chapman & Hall/CRC, Boca Raton, pp 1415–1433

13. Kettner L, Näher S (2004) Two computational geometry libraries: LEDA and CGAL. In: Goodman JE, O'Rourke J (eds) Handbook of discrete and computational geometry, chapter 65, 2nd edn. Chapman & Hall/CRC, Boca Raton, pp 1435–1463
14. Mehlhorn K, Näher S (2000) LEDA: a platform for combinatorial and geometric computing. Cambridge University Press, Cambridge
15. Wein R, Fogel E, Zukerman B, Halperin D (2007) Advanced programming techniques applied to CGAL's arrangement package. Comput Geom Theory Appl 36(1–2):37–63

Enumeration of Non-crossing Geometric Graphs

Shin-ichi Tanigawa
Research Institute for Mathematical Sciences (RIMS), Kyoto University, Kyoto, Japan

Keywords

Enumeration; Non-crossing (crossing-free) geometric graphs; Triangulations

Years and Authors of Summarized Original Work

2009; Katoh, Tanigawa
2014; Wettstein

Problem Definition

Let P be a set of n points in the plane in general position, i.e., no three points are collinear. A *geometric graph on* P is a graph on the vertex set P whose edges are straight-line segments connecting points in P. A geometric graph is called *non-crossing* (or *crossing-free*) if any pair of its edges does not have a point in common except possibly their endpoints. We denote by $\mathcal{P}(P)$ the set of all non-crossing geometric graphs on P (which are also called *plane straight-line graphs* on P). A graph class $\mathcal{C}(P) \subseteq \mathcal{P}(P)$ can be defined by imposing additional properties such as connectivity, degree bound, or cycle-freeness. Examples of $\mathcal{C}(P)$ are the set of *triangulations* (i.e., inclusion-wise maximal graphs in $\mathcal{P}(P)$), the set of *non-crossing perfect matchings*, the set of *non-crossing spanning k-connected graphs*, the set of *non-crossing spanning trees*, and the set of *non-crossing spanning cycles* (i.e., simple polygons). The problem is to enumerate all graphs in $\mathcal{C}(P)$ for a given set P of n points in the plane.

The following notations will be used to denote the cardinality of $\mathcal{C}(P)$: $\mathsf{tri}(P)$ for triangulations, $\mathsf{pg}(P)$ for plane straight-line graphs, $\mathsf{st}(P)$ for non-crossing spanning trees, and $\mathsf{cg}(P)$ for non-crossing spanning connected graphs.

Key Results

Enumeration of Triangulations

The first efficient enumeration algorithm for triangulations was given by Avis and Fukuda [3] as an application of their reverse search technique. The algorithm relies on well-known properties of Delaunay triangulations.

A triangulation T on P is called *Delaunay* if no point in P is contained in the interior of the circumcircle of a triangle in T. If it is assumed for simplicity that no four points in P lie on a circle, then the Delaunay triangulation on P exists and is unique. The Delaunay triangulation has the lexicographically largest angle vector among all triangulations on P, where the angle vector of a triangulation is the list of all the angles sorted in nondecreasing order.

For a triangulation T, a *Lawson edge* is an edge ab which is incident to two triangles, say abc and abd in T, and the circumcircle of abc contains d in its interior. *Flipping* a Lawson edge ab (i.e., replacing ab with another diagonal edge cd) always creates a triangulation having a lexicographically larger angle vector. Moreover a triangulation has a Lawson edge if and only if it is not Delaunay. In other words, any triangulation can be converted to the Delaunay triangulation by flipping Lawson edges.

In the algorithm by Avis and Fukuda, a rooted search tree on the set of triangulations is defined such that the root is the Delaunay triangulation and the parent of a non-Delaunay triangulation T

is a triangulation obtained by flipping the smallest Lawson edge in T (assuming a fixed total ordering on edges). Since the Delaunay triangulation can be computed in $O(n \log n)$ time, all the triangulations can be enumerated by tracing the rooted search tree based on the reverse search technique. A careful implementation achieves $O(n \cdot \mathsf{tr}(P))$ time with $O(n)$ space.

An improved algorithm was given by Bespamyatnikh [5], which runs in $O(\log \log n \cdot \mathsf{tr}(P))$ time with $O(n)$ space. His algorithm is also based on the reverse search technique, but the rooted search tree is defined by using the lexicographical ordering of edge vectors rather than angle vectors. This approach was also applied to the enumeration of pointed pseudo-triangulations [4]. See [6] for another approach.

Enumeration of Non-crossing Geometric Graphs

In [3], Avis and Fukuda also developed an enumeration algorithm for non-crossing spanning trees, whose running time is $O(n^3 \cdot \mathsf{sp}(P))$. This was improved to $O(n \log n \cdot \mathsf{sp}(P))$ by Aichholzer et al. [1]. They also gave enumeration algorithms for plane straight-line graphs and non-crossing spanning connected graphs with running time $O(n \log n \cdot \mathsf{pg}(P))$ and $O(n \log n \cdot \mathsf{sc}(P))$, respectively.

Katoh and Tangiawa [8] proposed a simple enumeration technique for wider classes of non-crossing geometric graphs. The same approach was independently given by Razen and Welzl [9] for counting the number of plane straight-line graphs, and the following description in terms of Delaunay triangulations is from [9].

Since each graph in $\mathcal{C}(P)$ is a subgraph of a triangulation, one can enumerate all graphs in $\mathcal{C}(P)$ by first enumerating all triangulations and then enumerating all graphs in $\mathcal{C}(P)$ in each triangulation. The output may contain duplicates, but one can avoid duplicates by enumerating only graphs in $\{G \in \mathcal{C}(P) \mid L(T) \subseteq E(G) \subseteq E(T)\}$ for each triangulation T, where $L(T)$ denotes the set of the Lawson edges in T. This enumeration framework leads to an algorithm with time complexity $O((t^{\mathrm{pre}} + \log \log n)\mathsf{tri}(P) + t \cdot \mathsf{c}(P))$ and space complexity $O(n + s)$ provided that graphs in $\{G \in \mathcal{C}(G) \mid L(T) \subseteq E(G) \subseteq E(T)\}$ can be enumerated in $O(t)$ time per graph with $O(t^{\mathrm{pre}})$ time preprocessing and $O(s)$ space for each triangulation T. For example, in the case of non-crossing spanning trees, one can use a fast enumeration algorithm for spanning trees in a given undirected graph to solve each subproblem, and the current best implementation gives an enumeration algorithm for non-crossing spanning trees with time complexity $O(n \cdot \mathsf{tri}(P) + \mathsf{st}(P))$.

For plane straight-line graphs and spanning connected graphs, $\mathsf{pg}(P) \geq (\sqrt{8})^n \mathsf{tri}(P)$ [9] and $\mathsf{cs}(P) \geq 1.51^n \mathsf{tri}(P)$ [8] hold for any P in general position. Hence $\mathsf{tri}(P)$ is dominated by $\mathsf{pg}(P)$ and $\mathsf{cs}(P)$, respectively, and plane straight-line graphs or non-crossing spanning connected graphs can be enumerated in constant time on average with $O(n)$ space [8]. The same technique can be applied to the set of non-crossing spanning 2-connected graphs. It is not known whether there is a constant $c > 1$ such that $\mathsf{st}(P) \geq c^n \mathsf{tri}(P)$ for every P in general position.

In [8] an approach that avoids enumerating all triangulations was also discussed. Suppose that a nonempty subset $\mathcal{I}$ of $\mathcal{P}(P)$ satisfies a monotone property, i.e., for every $G, G' \in \mathcal{P}(P)$ with $G \subseteq G'$, $G' \in \mathcal{I}$ implies $G \in \mathcal{I}$, and suppose that $\mathcal{C}(P)$ is the set of all maximal elements in $\mathcal{I}$. Then all graphs in $\mathcal{C}(P)$ can be enumerated just by enumerating all triangulations T on P with $L(T) \in \mathcal{I}$, and this can be done efficiently based on the reverse search technique. This approach leads to an algorithm for enumerating non-crossing minimally rigid graphs in $O(n^2)$ time per output with $O(n)$ space, where a graph $G = (V, E)$ is called *minimally rigid* if $|E| = 2|V| - 3$ and $|E'| \leq 2|V'| - 3$ for any subgraph $G' = (V', E')$ with $|V'| \geq 2$.

Enumeration of Non-crossing Perfect Matchings

Wettstein [10] proposed a new enumeration (and counting) technique for non-crossing geometric graphs. This is motivated from a counting algorithm of triangulations by Alvarez and

Seidel [2] and can be used for enumerating, e.g., non-crossing perfect matchings, plane straight-line graphs, convex subdivisions, and triangulations. The following is a sketch of the algorithm for non-crossing perfect matchings.

A matching can be reduced to an empty graph by removing edges one by one. By fixing a rule for the removing edge in each matching, one can define a rooted search tree $\mathcal{T}$ on the set of non-crossing matchings, and the set of non-crossing matchings can be enumerated by tracing $\mathcal{T}$. To reduce time complexity, the first idea is to trace only a subgraph $\mathcal{T}'$ of $\mathcal{T}$ induced by a subclass of non-crossing matchings by a clever choice of removing edges. Another idea is a compression of the search tree $\mathcal{T}'$ by using an equivalence relation on the subclass of non-crossing matchings. The resulting graph $\mathcal{G}$ is a digraph on the set of equivalence classes, where there is a one-to-one correspondence between non-crossing perfect matchings and directed paths of length $n/2$ from the root. A crucial observation is that $\mathcal{G}$ has at most $2^n n^3$ edges while the number of non-crossing perfect matchings is known to be at least $\mathrm{poly}(n) \cdot 2^n$ for any P in general position [7]. Hence non-crossing perfect matchings can be enumerated in polynomial time on average by first constructing $\mathcal{G}$ and then enumerating all the dipaths of length $n/2$ in $\mathcal{G}$. It was also noted in [10] that the algorithm can be polynomial-time delay, but still the space complexity is exponential in n.

Open Problems

A challenging open problem is to design an efficient enumeration algorithm for the set of non-crossing spanning cycles, the set of highly connected triangulations, or the set of degree-bounded triangulations or non-crossing spanning trees. It is also not known whether triangulations can be enumerated in constant time per output.

Cross-References

- ▸ Enumeration of Paths, Cycles, and Spanning Trees
- ▸ Reverse Search; Enumeration Algorithms
- ▸ Voronoi Diagrams and Delaunay Triangulations

Recommended Reading

1. Aichholzer O, Aurenhammer F, Huemer C, Krasser H (2007) Gray code enumeration of plane straight-line graphs. Graphs Comb 23(5):467–479
2. Alvarez V, Seidel R (2013) A simple aggregative algorithm for counting triangulations of planar point sets and related problems. In: Proceedings of the 29th annual symposium on computational geometry (SoCG2013), Rio de Janeiro. ACM
3. Avis D, Fukuda K (1996) Reverse search for enumeration. Discret Appl Math 65(1–3):21–46
4. Bereg S (2005) Enumerating pseudo-triangulations in the plane. Comput Geom Theory Appl 30(3):207–222
5. Bespamyatnikh S (2002) An efficient algorithm for enumeration of triangulations. Comput Geom Theory Appl 23(3):271–279
6. Brönnimann H, Kettner L, Pocchiola M, Snoeyink J (2006) Counting and enumerating pointed pseudotriangulations with the greedy flip algorithm. SIAM J Comput 36(3):721–739
7. García A, Noy M, Tejel J (2000) Lower bounds on the number of crossing-free subgraphs of K_n. Comput Geom Theory Appl 16(4):211–221
8. Katoh N, Tanigawa S (2009) Fast enumeration algorithms for non-crossing geometric graphs. Discret Comput Geom 42(3):443–468
9. Razen A, Welzl E (2011) Counting plane graphs with exponential speed-up. In: Calude C, Rozenberg G, Salomaa A, Maurer HA (eds) Rainbow of computer science. Springer, Berlin/Heidelberg, pp 36–46
10. Wettstein M (2014) Counting and enumerating crossing-free geometric graphs. In: Proceedings of the 30th annual symposium on computational geometry (SoCG2014), Kyoto. ACM

Enumeration of Paths, Cycles, and Spanning Trees

Roberto Grossi
Dipartimento di Informatica, Università di Pisa, Pisa, Italy

Keywords

Amortized analysis; Arborescences; Cycles; Elementary circuits; Enumeration algorithms; Graphs; Paths; Spanning trees

Years and Authors of Summarized Original Work

1975; Johnson
1975; Read and Tarjan
1994; Shioura, Tamura, Uno
1995; Kapoor, Ramesh
1999; Uno
2013; Birmelé, Ferreira, Grossi, Marino, Pisanti, Rizzi, Sacomoto, Sagot

Problem Definition

Let $G = (V, E)$ be a (directed or undirected) graph with $n = |V|$ vertices and $m = |E|$ edges. A *walk* of length k is a sequence of vertices $v_0, \ldots, v_k \in V$ such that v_i and v_{i+1} are connected by an edge of E, for any $0 \leq i < k$. A *path* π of length k is a walk $v_0, \ldots, v_k$ such that any two vertices v_i and v_j are distinct, for $0 \leq i < j \leq k$: this is also called *st-path* where $s = v_0$ and $t = v_k$. A *cycle* (or, equivalently, *elementary circuit*) C of length $k + 1$ is a path $v_0, \ldots, v_k$ such that v_k and v_0 are connected by an edge of E.

We denote by $\mathcal{P}_{st}(G)$ the set of *st*-paths in G for any two given vertices $s, t \in V$ and by $\mathcal{C}(G)$ the set of cycles in G. Given a graph G, the problem of *st-path enumeration* asks for generating all the paths in $\mathcal{P}_{st}(G)$. The problem of *cycle enumeration* asks for generating all the cycles in $\mathcal{C}(G)$.

We denote by $\mathcal{S}(G)$ the set of spanning trees in a connected graph G, where a spanning tree $T \subseteq E$ is a set of $|T| = n - 1$ edges such that no cycles are contained in T and each vertex in V is incident to at least an edge of T. Given a connected graph G, the problem of *spanning tree enumeration* asks for generating all the spanning trees in $\mathcal{S}(G)$.

Typical costs of enumeration algorithms are proportional to the output size times a polynomial function of the graph size. Sometimes enumeration is meant with the stronger property of *listing*, where each solution is explicitly output. In the latter case, we define an algorithm for a listing problem to be *optimally output sensitive* if its running time is $O(n + m + K)$ where K is the following output cost for the enumeration problem at hand, namely, $\mathcal{P}_{st}(G)$, $\mathcal{C}(G)$, or $\mathcal{S}(G)$.

- $K = \sum_{\pi \in \mathcal{P}_{st}(G)} |\pi|$ where $|\pi|$ is the number of nodes in the *st*-path π.
- $K = \sum_{C \in \mathcal{C}(G)} |C|$ where $|C|$ is the number nodes in the cycle C.
- $K = \sum_{T \in \mathcal{S}(G)} |T| = |\mathcal{S}(G)| \cdot (n - 1)$ for spanning trees.

Although the above is a notion of optimality for listing solutions explicitly, it is possible in some cases that the enumeration algorithm can efficiently encode the differences between consecutive solutions in the sequence produced by the enumeration. This is the case of spanning trees, where a cost of $K = |\mathcal{S}(G)|$ is possible when they are implicitly represented during enumeration. This is called CAT (constant amortized time) enumeration in [28].

Key Results

Some possible approaches to attack the enumeration problems are listed below, where the term "search" is meant as an exploration of the space of solutions.

Backtrack search. A backtracking algorithm finds the solutions for a listing problem by exploring the search space and abandoning a partial solution (thus the name "backtracking") that cannot be completed to a valid one.

Binary partition search. An algorithm divides the search space into two parts. In the case of graphs, this is generally done by taking an edge (or a vertex) and (i) searching for all solutions that include that edge (resp. vertex) and (ii) searching for all solutions that do not include that edge (resp. vertex). Point (i) can sometimes be implemented by contracting the edge, i.e., merging the endpoints of the edge and their adjacency list.

Differential encoding search. The space of solutions is encoded in such a way that consecutive solutions differ by a constant number of modifications. Although not every enumeration

problem has properties that allow such encoding, this technique leads to very efficient algorithms.

Reverse search. This is a general technique to explore the space of solutions by reversing a local search algorithm. This approach implicitly generates a tree of the search space that is traversed by the reverse search algorithm. One of the properties of this tree is that it has bounded height, a useful fact for proving the time complexity of the algorithm.

Although there is some literature on techniques for enumeration problems [38, 39, 41], many more techniques and "tricks" have been introduced when attacking particular problems. For a deep understanding of the topic, the reader is recommended to review the work of researchers such as David Avis, Komei Fukuda, Shin-ichi Nakano, and Takeaki Uno.

Path and Cycles

Listing all the cycles in a graph is a classical problem whose efficient solutions date back to the early 1970s. In particular, at the turn of the 1970s, several algorithms for enumerating all cycles of an undirected graph were proposed. There is a vast body of work, and the majority of the algorithms listing all the cycles can be divided into the following three classes (see [1, 23] for excellent surveys).

Search space algorithms. Cycles are looked for in an appropriate search space. In the case of undirected graphs, the *cycle vector space* [6] turned out to be the most promising choice: from a basis for this space, all vectors are computed, and it is tested whether they are a cycle. Since the algorithm introduced in [43], many algorithms have been proposed: however, the complexity of these algorithms turns out to be exponential in the dimension of the vector space and thus in n. For the special case of planar graphs, the paper in [34] describes an algorithm listing all the cycles in $O((|\mathcal{C}(G)| + 1)n)$ time.

Backtrack algorithms. All paths are generated by backtrack, and, for each path, it is tested whether it is a cycle. One of the first algorithms based on this approach is the one proposed in [37], which is however exponential in $|\mathcal{C}(G)|$. By adding a simple pruning strategy, this algorithm has been successively modified in [36]: it lists all the cycles in $O(nm(|\mathcal{C}(G)| + 1))$ time. Further improvements were proposed in [16], [35], and [27], leading to $O((|\mathcal{C}(G)| + 1)(m + n))$ time algorithms that work for both directed and undirected graphs. Apart from the algorithm in [37], all the algorithms based on this approach are *polynomial-time delay*, that is, the time elapsed between the outputting of two cycles is polynomial in the size of the graph (more precisely, $O(nm)$ in the case of the algorithm of [36] and $O(m)$ in the case of the other three algorithms).

Algorithms using the powers of the adjacency matrix. This approach uses the so-called variable adjacency matrix, that is, the formal sum of edges joining two vertices. A nonzero element of the pth power of this matrix is the sum of all walks of length p: hence, to compute all cycles, we compute the nth power of the variable adjacency matrix. This approach is not very efficient because of the non-simple walks. All algorithms based on this approach (e.g., [26] and [45]) basically differ only on the way they avoid to consider walks that are neither paths nor cycles.

For directed graphs, the best known algorithm for listing cycles is Johnson's [16]. It builds upon Tarjan's backtracking search [36], where the search starts from the least vertex of each strongly connected component. After that, a new strongly connected component is discovered, and the search starts again from the least vertex in it. When exploring a strongly connected component with a recursive backtracking procedure, it uses an enhanced marking system to avoid visiting the same cycle multiple times. A vertex is marked each time it enters the backtracking stack. Upon leaving the stack, if a cycle is found, then the vertex is unmarked. Otherwise, it remains marked until another vertex involved in a cycle is popped from the stack, and there exists a path of marked vertices (not in the stack) between these two vertices. This strategy is implemented using a collection of lists B, one list per vertex containing its marked neighbors not in the stack. Unmarking is done by a recursive procedure. The complexity of the algorithm is $O(n+m+|\mathcal{C}(G)|m)$ time and $O(n + m)$ space.

For undirected graphs, Johnson's bound can be improved with an optimal output-sensitive algorithm [2]. First of all, the cycle enumeration problem is reduced to the *st*-path enumeration by considering any spanning tree of the given graph G and its non-tree edges $b_1, b_2, \ldots, b_r$. Then, for $i = 1, 2, \ldots, r$, the cycles in $\mathcal{C}(G)$ can be listed as *st*-paths in $G \setminus \{b_1, \ldots, b_i\}$, where s and t are the endpoint of non-tree edge b_i. Hence, the subproblem to be solved with an optimal output-sensitive algorithm is the *st*-path enumeration problem. Binary partition search is adopted to avoid duplicated output, but the additional ingredient is the notion of *certificate*, which is a suitable data structure that maintains the biconnected components of the residual graph and guarantees that each recursive call thus produces at least one solution. Its amortized analysis is based on a lower bound on the number of *st*-paths that can be listed in the residual graph, so as to absorb the cost of maintaining the certificate. The final cost is $O(m+n+\sum_{\pi \in \mathcal{P}_{st}(G)} |\pi|)$ time and $O(n+m)$ space, which is optimal for listing.

Spanning Trees

Listing combinatorial structures in graphs has been a long-time problem of interest. In his 1970 book [25], Moon remarks that "many papers have been written giving algorithms, of varying degrees of usefulness, for listing the spanning trees of a graph" (citation taken from [28]). Among others, he cites [7, 9, 10, 13, 42] – some of these early papers date back to the beginning of the twentieth century. More recently, in the 1960s, Minty proposed an algorithm to list all spanning trees [24].

The first algorithmic solutions appeared in the 1960s [24] and the combinatorial papers even much earlier [25]. Other results from Welch, Tiernan, and Tarjan for this and other problems soon followed [36, 37, 43] and used backtracking search. Read and Tarjan presented an algorithm taking $O(m + n + |\mathcal{S}(G)| \cdot m)$ time and $O(m + n)$ space [27]. Gabow and Myers proposed the first algorithm [11] which is optimal when the spanning trees are explicitly listed, taking $O(m + n + |\mathcal{S}(G)| \cdot n)$ time and $O(m + n)$ space.

When the spanning trees are implicitly enumerated, Kapoor and Ramesh [17] showed that an elegant incremental representation is possible by storing just the $O(1)$ information needed to reconstruct a spanning tree from the previously enumerated one, requiring a total of $O(m + n + |\mathcal{S}(G)|)$ time and $O(mn)$ space [17], later reduced to $O(m)$ space by Shioura et al. [32]. These methods use the reverse search where the elements are the spanning trees. The rule for moving along these elements and for their differential encoding is based upon the observation that adding a non-tree edge and removing a tree edge of the cycle thus formed produces another spanning tree from the current one. Some machinery is needed to avoid duplicated spanning trees and to spend $O(1)$ amortized cost per generated spanning tree.

A simplification of the incremental enumeration of spanning trees is based on matroids and presented by Uno [39]. It is a binary partition search giving rise to a binary enumeration tree, where the two children calls generated by the current call correspond to the fact that the current edge is either contracted in $O(n)$ time or deleted in $O(m - n)$ time. There is a trimming and balancing phase in $O(n(m - n))$ time: trimming removes the edges that do not appear in any of the spanning trees that will be generated by the current recursive call and contracts the edges that appear in all of these spanning trees. Balancing splits the recursive calls as in the divide-and-conquer paradigm. A crucial property proved in [39] is that the residual graph will generate at least $\Omega(n(m - n))$ spanning trees, and thus the total cost per call, which is dominated by trimming and balancing, can be amortized as $O(1)$ per spanning tree. The method in [39] works also for directed spanning trees (arborescences) with an amortized $O(\log n)$ time cost per directed spanning tree.

Applications

The classical problem of listing all the cycles of a graph has been extensively studied for its many

applications in several fields, ranging from the mechanical analysis of chemical structures [33] to the design and analysis of reliable communication networks and the graph isomorphism problem [43]. Almost 40 years after, the problem of efficiently listing all cycles of a graph is still an active area of research (e.g., [14, 15, 22, 29, 30, 44]). New application areas have emerged in the last decade, such as bioinformatics: for example, two algorithms for this problem have been proposed in [20] and [21] while studying biological interaction graphs, with important network properties derived for feedback loops, signaling paths, and dependency matrix, to name a few.

When considering weighted cycles, the paper in [19] proves that there is no polynomial total time algorithm (unless $P = NP$) to enumerate negative-weight (simple) cycles in directed weighted graphs. Uno [40] and Ferreira et al. [8] considered the enumeration of chordless cycles and paths. A chordless or induced cycle (resp., path) in an undirected graph is a cycle (resp., path) such that the subgraph induced by its vertices contains exactly the edges of the cycle (resp., path). Both chordless cycles and paths are very natural structures in undirected graphs with an important history, appearing in many papers in graph theory related to chordal graphs, perfect graphs, and co-graphs (e.g., [4, 5, 31]), as well as many NP-complete problems involving them (e.g., [3, 12, 18]).

As for spanning trees, we refer to the section "K-best enumeration" of this book.

Recommended Reading

1. Bezem G, Leeuwen Jv (1987) Enumeration in graphs. Technical Report RUU-CS-87-07, Utrecht University
2. Birmelé E, Ferreira R, Grossi R, Marino A, Pisanti N, Rizzi R, Sacomoto G, Sagot MF (2013) Optimal listing of cycles and st-paths in undirected graphs. In: Proceedings of the twenty-fourth annual ACM-SIAM symposium on discrete algorithms, New Orleans. SIAM, pp 1884–1896
3. Chen Y, Flum J (2007) On parameterized path and chordless path problems. In: IEEE conference on computational complexity, San Diego, pp 250–263
4. Chudnovsky M, Robertson N, Seymour P, Thomas R (2006) The strong perfect graph theorem. Ann Math 164:51–229
5. Conforti M, Rao MR (1992) Structural properties and decomposition of linear balanced matrices. Math Program 55:129–168
6. Diestel R (2005) Graph theory. Graduate texts in mathematics. Springer, Berlin/New York
7. Duffin R (1959) An analysis of the wang algebra of networks. Trans Am Math Soc 93:114–131
8. Ferreira RA, Grossi R, Rizzi R, Sacomoto G, Sagot M (2014) Amortized $\tilde{O}(|V|)$-delay algorithm for listing chordless cycles in undirected graphs. In: Proceedings of European symposium on algorithms. LNCS, vol 8737. Springer, Berlin/Heidelberg, pp 418–429
9. Feussner W (1902) Uber stromverzweigung in netzformigen leitern. Ann Physik 9:1304–1329
10. Feussner W (1904) Zur berechnung der stromstarke in netzformigen leitern. Ann Physik 15:385–394
11. Gabow HN, Myers EW (1978) Finding all spanning trees of directed and undirected graphs. SIAM J Comput 7(3):280–287
12. Haas R, Hoffmann M (2006) Chordless paths through three vertices. Theor Comput Sci 351(3):360–371
13. Hakimi S (1961) On trees of a graph and their generation. J Frankl Inst 272(5):347–359
14. Halford TR, Chugg KM (2004) Enumerating and counting cycles in bipartite graphs. In: IEEE Communication Theory Workshop, Cancun
15. Horváth T, Gärtner T, Wrobel S (2004) Cyclic pattern kernels for predictive graph mining. In: Proceedings of 10th ACM SIGKDD, Seattle, pp 158–167
16. Johnson DB (1975) Finding all the elementary circuits of a directed graph. SIAM J Comput 4(1):77–84
17. Kapoor S, Ramesh H (1995) Algorithms for enumerating all spanning trees of undirected and weighted graphs. SIAM J Comput 24:247–265
18. Kawarabayashi K, Kobayashi Y (2008) The induced disjoint paths problem. In: Lodi A, Panconesi A, Rinaldi G (eds) IPCO. Lecture notes in computer science, vol 5035. Springer, Berlin/Heidelberg, pp 47–61
19. Khachiyan L, Boros E, Borys K, Elbassioni K, Gurvich V (2006) Generating all vertices of a polyhedron is hard. In: Proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithm, society for industrial and applied mathematics, Philadelphia, SODA '06, Miami, pp 758–765
20. Klamt S et al (2006) A methodology for the structural and functional analysis of signaling and regulatory networks. BMC Bioinform 7:56
21. Klamt S, von Kamp A (2009) Computing paths and cycles in biological interaction graphs. BMC Bioinform 10:181
22. Liu H, Wang J (2006) A new way to enumerate cycles in graph. In: AICT and ICIW, Washington, DC, USA pp 57–59
23. Mateti P, Deo N (1976) On algorithms for enumerating all circuits of a graph. SIAM J Comput 5(1):90–99
24. Minty G (1965) A simple algorithm for listing all the trees of a graph. IEEE Trans Circuit Theory 12(1):120–120

25. Moon J (1970) Counting labelled trees. Canadian mathematical monographs, vol 1. Canadian Mathematical Congress, Montreal
26. Ponstein J (1966) Self-avoiding paths and the adjacency matrix of a graph. SIAM J Appl Math 14:600–609
27. Read RC, Tarjan RE (1975) Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. Networks 5(3):237–252
28. Ruskey F (2003) Combinatorial generation. Preliminary working draft University of Victoria, Victoria
29. Sankar K, Sarad A (2007) A time and memory efficient way to enumerate cycles in a graph. In: Intelligent and advanced systems, Kuala Lumpur pp 498–500
30. Schott R, Staples GS (2011) Complexity of counting cycles using Zeons. Comput Math Appl 62:1828–1837
31. Seinsche D (1974) On a property of the class of n-colorable graphs. J Comb Theory, Ser B 16(2):191–193
32. Shioura A, Tamura A, Uno T (1994) An optimal algorithm for scanning all spanning trees of undirected graphs. SIAM J Comput 26:678–692
33. Sussenguth E (1965) A graph-theoretical algorithm for matching chemical structures. J Chem Doc 5:36–43
34. Syslo MM (1981) An efficient cycle vector space algorithm for listing all cycles of a planar graph. SIAM J Comput 10(4):797–808
35. Szwarcfiter JL, Lauer PE (1976) A search strategy for the elementary cycles of a directed graph. BIT Numer Math 16:192–204
36. Tarjan RE (1973) Enumeration of the elementary circuits of a directed graph. SIAM J Comput 2(3):211–216
37. Tiernan JC (1970) An efficient search algorithm to find the elementary circuits of a graph. Commun ACM 13:722–726
38. Uno T (1998) New approach for speeding up enumeration algorithms. Algorithms and computation. Springer, Berlin/Heidelberg, pp 287–296
39. Uno T (1999) A new approach for speeding up enumeration algorithms and its application for matroid bases. In: COCOON, Tokyo, pp 349–359
40. Uno T (2003) An output linear time algorithm for enumerating chordless cycles. In: 92nd SIGAL of information processing society Japan, Tokyo pp 47–53, (in Japanese)
41. Uno T (2003) Two general methods to reduce delay and change of enumeration algorithms. National Institute of Informatics, Technical Report NII-2003-004E, Tokyo, Apr. 2003
42. Wang K (1934) On a new method for the analysis of electrical networks. Nat Res Inst for Eng Academia Sinica Memoir (2):19
43. Welch JT Jr (1966) A mechanical analysis of the cyclic structure of undirected linear graphs. J ACM 13:205–210
44. Wild M (2008) Generating all cycles, chordless cycles, and hamiltonian cycles with the principle of exclusion. J Discret Algorithms 6:93–102
45. Yau S (1967) Generation of all hamiltonian circuits, paths, and centers of a graph, and related problems. IEEE Trans Circuit Theory 14:79–81

Equivalence Between Priority Queues and Sorting

Rezaul A. Chowdhury
Department of Computer Sciences, University of Texas, Austin, TX, USA
Stony Brook University (SUNY), Stony Brook, NY, USA

Keywords

AC^0 operation; Pointer machine; Priority queue; Sorting; Word RAM

Synonyms

Heap

Years and Authors of Summarized Original Work

2007 (2002); Thorup

Problem Definition

A *priority queue* is an abstract data structure that maintains a set Q of elements, each with an associated value called a *key*, under the following set of operations [5,6]:

insert(Q, x, k): Inserts element x with key k into Q.
find-min(Q): Returns an element of Q with the minimum key but does not change Q.
delete(Q, x, k): Deletes element x with key k from Q.

Additionally, the following operations are often supported:

delete-min(Q): Deletes an element with the minimum key value from Q and returns it.
decrease-key(Q, x, k): Decreases the current key k' of x to k assuming $k < k'$.
meld(Q_1 , Q_2): Given priority queues Q_1 and Q_2, returns the priority queue $Q_1 \cup Q_2$.

Observe that a *delete-min* can be implemented as a *find-min* followed by a *delete*, a *decrease-key* as a *delete* followed by an *insert*, and a *meld* as a series of *find-min*, *delete* and *insert*. However, more efficient implementations of *decrease-key* and *meld* often exist [5, 6].

Priority queues have many practical applications including event-driven simulation, job scheduling on a shared computer, and computation of shortest paths, minimum spanning forests, minimum cost matching, optimum branching, etc. [5, 6].

A priority queue can trivially be used for sorting by first inserting all keys to be sorted into the priority queue and then by repeatedly extracting the current minimum. The major contribution of Mikkel Thorup's 2002 article (Full version published in 2007) titled "Equivalence between Priority Queues and Sorting" [17] is a reduction showing that the converse is also true. Taken together, these two results imply that priority queues are computationally equivalent to sorting, that is, asymptotically, the per key cost of sorting is the update time of a priority queue.

A result similar to those in the current work [17] was presented earlier by the same author [14] which resulted in monotone priority queues (i.e., meaning that the extracted minimums are nondecreasing) with amortized time bounds only. In contrast, the current work [17] constructs general priority queues with worst-case bounds.

In addition to establishing the equivalence between priority queues and sorting, Thorup's reductions [17] are also used to translate several known sorting results into new results on priority queues.

Background

Some relevant background information is summarized below which will be useful in understanding the key results in section "Key Results."

- A standard *word RAM* models what one programs in a standard imperative programming language such as C. In addition to direct and indirect addressing and conditional jumps, there are functions, such as addition and multiplication, operating on a constant number of words. The memory is divided into words, addressed linearly starting from 0. The running time of a program is the number of instructions executed and the space is the maximal address used. The word length is a machine-dependent parameter which is big enough to hold a key and at least logarithmic in the number of input keys so that they can be addressed.
- A pointer machine is like the word RAM except that addresses cannot be manipulated.
- The AC^0 complexity class consists of constant-depth circuits with unlimited fan-in [18]. Standard AC^0 operations refer to the operations available via C but where the functions on words are in AC^0. For example, this includes addition but not multiplication.
- Integer keys will refer to nonnegative integers. However, if the input keys are signed integers, the correct ordering of the keys is obtained by flipping their sign bits and interpreting them as unsigned integers. Similar tricks work for floating point numbers and integer fractions [14].
- The atomic heaps of Fredman and Willard [7] are used in one of Thorup's reductions [17]. These heaps can support updates and searches in sets of $\mathcal{O}\left(\log^2 n\right)$ keys in $\mathcal{O}(1)$ worst-case time [20]. However, atomic heaps use multiplication operations which are not in AC^0.

Key Results

The main results in this paper are two reductions from priority queues to sorting. The stronger of the two, stated in Theorem 1, is for integer priority queues running on a standard word RAM.

Theorem 1 *If for some nondecreasing function S, up to n integer keys can be sorted in $S(n)$ time per key, an integer priority queue can be im-*

plemented supporting find-min *in constant time, and updates, i.e.,* insert *and* delete*, in* $\mathcal{O}(S(n))$ *time. Here* n *is the current number of keys in the queue. The reduction uses linear space. The reduction runs on a standard word RAM assuming that each integer key is contained in a single word.*

The reduction above provides the following new bounds for linear space integer priority queues improving previous bounds given by Han [8] and Thorup [14], respectively:

1. **(Deterministic)** $\mathcal{O}(\log\log n)$ update time using a sorting algorithm by Han [9].
2. **(Randomized)** $\mathcal{O}\left(\sqrt{\log\log n}\right)$ expected update time using a sorting algorithm given by Han and Thorup [10].

The reduction in Theorem 1 employs atomic heaps [7] which, in addition to being very complicated, use AC^0 operations. The following slightly weaker recursive reduction which does not restrict the domain of the keys is completely combinatorial.

Theorem 2 *If for some nondecreasing function* S*, up to* n *keys can be sorted in* $S(n)$ *time per key, a priority queue can be implemented supporting* find-min *in constant time, and updates in* $T(n)$ *time where* n *is the current number of keys in the queue and* $T(n)$ *satisfies the recurrence:*

$$T(n) = \mathcal{O}(S(n)) + T(\mathcal{O}(\log n))$$

The reduction runs on a pointer machine in linear space using only standard AC^0 *operations.*

This reduction implies the following new integer priority queue bounds not implied by Theorem 1, which improve previous bounds given by Thorup in 1998 [13] and 1997 [15], respectively:

1. **(Deterministic in AC^0)** $\mathcal{O}\left((\log\log n)^{1+\epsilon}\right)$ update time for any constant $\epsilon > 0$ using a standard AC^0 sorting algorithm given by Han and Thorup [10].
2. **(Randomized in AC^0)** $\mathcal{O}(\log\log n)$ expected update time using a randomized AC^0 sorting algorithm given by Thorup [15].

The Reduction in Theorem 1

Given a sorting routine that can sort up to n keys in $S(n)$ time per key, the priority queue is constructed as follows. All keys are assumed to be distinct.

The data structure has two major components: a partially sorted list of keys called a *base list* and a set of *level buffers* (also called *update buffers*). Most keys of the priority queue reside in the base list partitioned into logarithmic-sized disjoint sets called *base sets*. While the keys inside any given base set are not required to be sorted, each of those keys must be larger than every key in the base set (if any) appearing before it in the list. Keys inside each base set are stored in a doubly linked list allowing constant time updates. The first base set in the list containing the smallest key among all base sets is also maintained in an atomic heap so that the current minimum can be found in constant time. Each level buffer has a different capacity and accumulates updates (*insert*/*delete*) with key values in a different range. Smaller level buffers accept updates with smaller keys. An atomic heap is used to determine in constant time which level buffer collects a new update. When a level buffer accumulates enough updates, they first enter a sorting phase and then a merging phase. In the merging phase each update is applied on the proper base set in the key list, and invariants on base set size and ranges of level buffers are fixed. These phases are not executed immediately, instead they are executed in fixed time increments over a period of time. A level buffer continues to accept new updates, while some updates accepted by it earlier are still in the sorting phase, and some even older updates are in the merging phase. Every time it accepts a new update, $\mathcal{O}(S(n))$ time is spent on the sorting phase associated with it and $\mathcal{O}(1)$ time on its merging phase including rebalancing of base sets and scanning. This strategy allows the sorting and merging phases to complete execution by the time the level buffer becomes full again and thus keep-

ing the movement of updates through different phases smooth while maintaining an $\mathcal{O}(S(n))$ worst-case time bound per update. Moreover, the size and capacity constraints ensure that the smallest key in the data structure is available in $\mathcal{O}(1)$ time. More details are given below.

The Base List: The base list consists of base sets $A_1, A_2, \ldots, A_k$, where $\frac{\Phi}{4} \leq |A_i| \leq \Phi$ for $i < k$, and $|A_k| \leq \Phi$ for some $\Phi = \Theta(\log n)$. The exact value of Φ is chosen carefully to make sure that it conforms with the requirements of the delicate worst-case base set rebalancing protocol used by the reduction. The base sets are partitioned by *base splitters* $s_0, s_1, \ldots, s_{k+1}$, where $s_0 = -\infty$, $s_{k+1} = \infty$, and for $i = 1, \ldots, k-1$, $\max A_{i-1} < s_i \leq \min A_i$. If a base set becomes too large or too small, it is split or joined with an adjacent set, respectively.

Level Buffers: Among the base splitters $l + 2 = \Theta(\log n)$ are chosen to become *level splitters* $t_0, t_1, \ldots, t_l, t_{l+1}$ with $t_0 = s_0 = -\infty$ and $t_{l+1} = s_{k+1} = \infty$, so that for $j > 0$, the number of keys in the base list below t_j is around $4^{j+1}\Phi$. These splitters are placed in an atomic heap. As the base list changes the level splitters are moved, as needed, in order to maintain their exponential distribution.

Associated with each level splitter t_j, $1 \leq j \leq l$, is a *level buffer* B_j containing keys in $[t_{j-1}, t_{j+2})$, where $t_{l+2} = \infty$. Buffer B_j consists of an *entrance* buffer, a *sorter*, and a *merger*, each with capacity for 4^j keys. Level j works in a cycle of 4^j steps. The cycle starts with an empty entrance, at most 4^j updates in the sorter, and a sorted list of at most 4^j updates in the merger. In each step one may accept an update for the entrance, spend $S\left(4^j\right) = \mathcal{O}(S(n))$ time in the sorter and $\mathcal{O}(1)$ time in merging the sorted list in the merger with the $\mathcal{O}\left(4^j\right)$ base splitters in $[t_{j-1}, t_{j+2})$ and scanning for a new t_j among them. Therefore, after 4^j such steps, the sorted list is correctly merged with the base list, a new t_j is found, and a new sorted list is produced. The sorter then takes the role of the merger, the entrance becomes the sorter, and the empty merger becomes the new entrance.

Handling Updates: When a new update key k (*insert*/*delete*) is received, the atomic heap of level splitters is used to find in $\mathcal{O}(1)$ time the t_j such that $k \in [t_{j-1}, t_j)$. If $k \in [t_0, t_1)$, its position is identified among the $\mathcal{O}(1)$ base splitters below t_1, and the corresponding base set is updated in $\mathcal{O}(1)$ time using the doubly linked list and the atomic heap (if exists) over the keys of that set. If $k \in [t_{j-1}, t_j)$ for some $j > 1$, the update is placed in the entrance of B_j, performing one step of the cycle of B_j in $\mathcal{O}(S(n))$ time. Additionally, during each update another splitter t_r is chosen in a round-robin fashion, and a step of a cycle of level r is executed in $\mathcal{O}(S(n))$ time. This additional work ensures that after every l updates some progress is made on moving each level splitter.

A *find-min* returns the minimum element of the base list which is available in $\mathcal{O}(1)$ time.

The Reduction in Theorem 2

This reduction follows from the previous reduction by replacing the atomic heap containing the level splitters with a data structure similar to a level buffer and the atomic heap over the keys of the first base set with a recursively defined priority queue satisfying the following recurrence for update time: $T(n) = \mathcal{O}(S(n)) + T(\mathcal{O}(\Phi))$.

Further Improvement

Alstrup et al. [1] presented a general reduction that transforms a priority queue to support *insert* in $\mathcal{O}(1)$ time while keeping the other bounds unchanged. This reduction can be used to reduce the cost of insertion to a constant in Theorems 1 and 2.

Applications

Thorup's equivalence results [17] can be used to translate known sorting results into new results on priority queues for integers and strings in different computational models (see section "Key Results"). These results can also be viewed as a

new means of proving lower bounds for sorting via priority queues.

A new RAM priority queue that matches the bounds in Theorem 1 and also supports *decrease-key* in $\mathcal{O}(1)$ time is presented by Thorup [16]. This construction combines Andersson's exponential search trees [2] with the priority queues implied by Theorem 1. The reduction in Theorem 1 is also used by Pagh et al. [12] in order to develop an adaptive integer sorting algorithm for the word RAM and by Arge and Thorup [3] to develop a sorting algorithm that is simultaneously I/O efficient and internal memory efficient in the RAM model of computation. Cohen et al. [4] use a priority queue generated through this reduction to obtain a simple and fast amortized implementation of a reservoir sampling scheme that provides variance optimal unbiased estimation of subset sums. Reductions from meldable priority queues to sorting presented by Mendelson et al. [11] use the reductions from non-meldable priority queues to sorting given in [17].

An external-memory version of Theorem 1 has been proved by Wei and Yi [19].

Open Problems

One major open problem is to find a general reduction (if one exists) that allows us to decrease the value of a key in constant time. Another open question is whether the gap between the bounds implied by Theorems 1 and 2 can be reduced or removed. For example, for a hypothetical linear time-sorting algorithm, Theorem 1 implies a priority queue with an update time of $\mathcal{O}(1)$, while Theorem 2 implies only $\mathcal{O}\left(\log^* n\right)$-time updates.

Cross-References

- Cache-Oblivious Sorting
- External Sorting and Permuting
- Minimum Spanning Trees
- Single-Source Shortest Paths
- String Sorting
- Suffix Tree Construction

Recommended Reading

1. Alstrup S, Husfeldt T, Rauhe T, Thorup M (2005) Black box for constant-time insertion in priority queues (note). ACM TALG 1(1):102–106
2. Andersson A (1996) Faster deterministic sorting and searching in linear space. In: Proceedings of the 37th FOCS, Burlington, pp 135–141
3. Arge L, Thorup M (2013) RAM-efficient external memory sorting. In: Proceedings of the 24th ISAAC, Hong Kong, pp 491–501
4. Cohen E, Duffield N, Kaplan H, Lund C, Thorup M (2009) Stream sampling for variance-optimal estimation of subset sums. In: Proceedings of the 20th SODA, New York, pp 1255–1264
5. Cormen T, Leiserson C, Rivest R, Stein C (2009) Introduction to algorithms. MIT, Cambridge
6. Fredman M, Tarjan R (1987) Fibonacci heaps and their uses in improved network optimization algorithms. J ACM 34(3):596–615
7. Fredman M, Willard D (1994) Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J Comput Syst Sci 48:533–551
8. Han Y (2001) Improved fast integer sorting in linear space. Inf Comput 170(8):81–94. Announced at STACS'00 and SODA'01
9. Han Y (2004) Deterministic sorting in $O(n \log\log n)$ time and linear space. J Algorithms 50(1):96–105. Announced at STOC'02
10. Han Y, Thorup M (2002) Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In: Proceedings of the 43rd FOCS, Vancouver, pp 135–144
11. Mendelson R, Tarjan R, Thorup M, Zwick U (2006) Melding priority queues. ACM TALG 2(4):535–556. Announced at SODA'04
12. Pagh A, Pagh R, Thorup M (2004) On adaptive integer sorting. In: Proceedings of the 12th ESA, Bergen, pp 556–579
13. Thorup M (1998) Faster deterministic sorting and priority queues in linear space. In: Proceedings of the 9th SODA, San Francisco, pp 550–555
14. Thorup M (2000) On RAM priority queues. SIAM J Comput 30(1):86–109. Announced at SODA'96
15. Thorup M (2002) Randomized sorting in $O(n \log\log n)$ time and linear space using addition, shift, and bit-wise boolean operations. J Algorithms 42(2):205–230. Announced at SODA'97
16. Thorup M (2004) Integer priority queues with decrease key in constant time and the single source shortest paths problem. J Comput Syst Sci (special issue on STOC'03) 69(3):330–353
17. Thorup M (2007) Equivalence between priority queues and sorting. J ACM 54(6):28. Announced at FOCS'02
18. Vollmer H (1999) Introduction to circuit complexity: a uniform approach. Springer, Berlin/New York

19. Wei Z, Yi K (2014) Equivalence between priority queues and sorting in external memory. In: Proceedings of 22nd ESA, Wroclaw, pp 830–841
20. Willard D (2000) Examining computational geometry, van Emde Boas trees, and hashing from the perspective of the fusion tree. SIAM J Comput 29(3):1030–1049. Announced at SODA'92

Estimating Simple Graph Parameters in Sublinear Time

Oded Goldreich[1] and Dana Ron[2]
[1]Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel
[2]School of Electrical Engineering, Tel-Aviv University, Ramat-Aviv, Israel

Keywords

Graph parameters; Sublinear-time algorithms

Years and Authors of Summarized Original Work

2008; Goldreich, Ron

Problem Definition

A *graph parameter* σ is a real-valued function over graphs that is invariant under graph isomorphism. For example, the average degree of the graph, the average distance between pairs of vertices, and the minimum size of a vertex cover are graph parameters. For a fixed graph parameter σ and a graph $G = (V, E)$, we would like to compute an estimate of $\sigma(G)$. To this end we are given query access to G and would like to perform this task in time that is *sublinear* in the size of the graph and with high success probability. In particular, this means that we do not read the entire graph but rather only access (random) parts of it (via the query mechanism). Our main focus here is on a very basic graph parameter: its average degree, denoted $\overline{d}(G)$.

The estimation algorithm is given an approximation parameter $\epsilon > 0$. It should output a value $\hat{d}$ such that with probability at least $2/3$ (over the random choices of the algorithm) it holds that $\overline{d}(G) \leq \hat{d} \leq (1+\epsilon) \cdot \overline{d}(G)$. (The error probability can be decreased to 2^{-k} by invoking the algorithm $\Theta(k)$ times and outputting the median value.) For any vertex $v \in V = [n]$ of its choice, where $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$, the estimation algorithm may query the degree of v, denoted $d(v)$. We refer to such queries as *degree queries*. In addition, the algorithm may ask for the ith neighbor of v, for any $1 \leq i \leq d(v)$. These queries are referred to as *neighbor queries*. We assume for simplicity that G does not contain any isolated vertices (so that, in particular, $\overline{d}(G) \geq 1$). This assumption can be removed.

Key Results

The problem of estimating the average degree of a graph in sublinear time was first studied by Feige [7]. He considered this problem when the algorithm is allowed only degree queries, so that the problem is a special case of estimating the average value of a function given query access to the function. For a general function $d : [n] \to [n-1]$, obtaining a constant-factor estimate of the average value of the function (with constant success probability) requires $\Omega(n)$ queries to the function. Feige showed that when d is the degree function of a graph, for any $\gamma \in (0, 1]$, it is possible to obtain an estimate of the average degree that is within a factor of $(2+\gamma)$ by performing only $O(\sqrt{n}/\gamma)$ (uniformly selected) queries. He also showed that in order to go below a factor of 2 in the quality of the estimate, $\Omega(n)$ queries are necessary.

However, given that the object in question is a graph, it is natural to allow the algorithm to query the neighborhood of vertices of its choice and not only their degrees; indeed, the aforementioned problem definition follows this natural convention. Goldreich and Ron [10] showed that by giving the algorithm this extra power, it is possible to break the factor-2 barrier. They provide an algorithm that, given $\epsilon > 0$, outputs a $(1+\epsilon)$-factor estimate of the average degree (with probability at least $2/3$) after performing $O(\sqrt{n} \cdot$

poly$(\log n, 1/\epsilon)$) degree and neighbor queries. In fact, since a degree query to vertex v can be replaced by $O(\log d(v)) = O(\log n)$ neighbor queries, which implement a binary search, degree queries are not necessary. Furthermore, when the average degree increases, the performance of the algorithm improves, as stated next.

Theorem 1 *There exists an algorithm that makes only neighbor queries to the input graph and satisfies the following condition. On input $G = (V, E)$ and $\epsilon \in (0, 1)$, with probability at least $2/3$, the algorithm halts within $O\left(\sqrt{n/\overline{d}(G)} \cdot \text{poly}(\log n, 1/\epsilon)\right)$ steps and outputs a value in $[\overline{d}(G), (1+\epsilon) \cdot \overline{d}(G)]$.*

The running time stated in Theorem 1 is essentially optimal in the sense that (as shown in [10]) a $(1+\epsilon)$-factor estimate requires $\Omega(\sqrt{n/(\epsilon\overline{d}(G))})$ queries, for every value of n, for $\overline{d}(G) \in [2, o(n)]$, and for $\epsilon \in [\omega(n^{-1/4}), o(n/\overline{d}(G))]$.

The following is a high-level description of the algorithm and the ideas behind its analysis. For the sake of simplicity, we only show how to obtain a $(1+\epsilon)$-factor estimate by performing $O\left(\sqrt{n} \cdot \text{poly}(\log n, 1/\epsilon)\right)$ queries (under the assumption that $\overline{d}(G) \geq 1$). For the sake of the presentation, we also allow the algorithm to perform degree queries. We assume that $\epsilon \leq 1/2$, or else we run the algorithm with $\epsilon = 1/2$. We first show how to obtain a $(2+\epsilon)$-approximation by performing only degree queries and then explain how to improve the approximation by using neighbor queries as well.

Consider a partition of the graph vertices into *buckets* $B_1, \ldots, B_r$, where

$$B_i \stackrel{\text{def}}{=} \{v : (1+\epsilon/8)^{i-1} \leq d(v) < (1+\epsilon/8)^i\} \tag{1}$$

and $r = O(\log n/\epsilon)$. By this definition,

$$\frac{1}{n}\sum_{i=1}^{r} |B_i| \cdot (1+\epsilon/8)^i \in \left[\overline{d}(G), (1+\epsilon/8) \cdot \overline{d}(G)\right]. \tag{2}$$

Suppose we could obtain an estimate $\hat{b}_i$ of the size of each bucket B_i such that $\hat{b}_i \in [(1-\epsilon/8)|B_i|, (1+\epsilon/8)|B_i|]$. Then

$$\frac{1}{n}\sum_{i=1}^{r} \hat{b}_i \cdot (1+\epsilon/8)^i \in \left[(1-\epsilon/8) \cdot \overline{d}(G), (1+3\epsilon/8) \cdot \overline{d}(G)\right]. \tag{3}$$

Now, for each B_i, if we uniformly at random select $\Omega\left(\frac{n}{|B_i|} \cdot \frac{\log r}{\epsilon^2}\right)$ vertices, then, by a multiplicative Chernoff bound, with probability $1 - O(1/r)$, the fraction of sampled vertices that belong to B_i is in the interval $\left[(1-\epsilon/8)\frac{|B_i|}{n}, (1+\epsilon/8)\frac{|B_i|}{n}\right]$. By querying the degree of each sampled vertex, we can determine to which bucket it belongs and obtain an estimate of $|B_i|$. Unfortunately, if B_i is much smaller than $\sqrt{n}$, then the sample size required to estimate $|B_i|$ is much larger than the desired $O\left(\sqrt{n} \cdot \text{poly}(\log n, 1/\epsilon)\right)$. Let $L \stackrel{\text{def}}{=} \{i : |B_i| \geq \sqrt{\epsilon n/8r}\}$ denote the set of indices of *large* buckets. The basic observation is that if, for each $i \in L$, we have an estimate $\hat{b}_i \in [(1-\epsilon/8)|B_i|, (1+\epsilon/8)|B_i|]$, then

$$\frac{1}{n}\sum_{i\in L} \hat{b}_i \cdot (1+\epsilon/8)^i \in \left[(1/2-\epsilon/4) \cdot \overline{d}(G), (1+3\epsilon/8) \cdot \overline{d}(G)\right]. \tag{4}$$

The reasoning is essentially as follows. Recall that $\sum_v d(v) = 2|E|$. Consider an edge (u, v) where $u \in B_j$ and $v \in B_k$. If $j, k \in L$, then this edge contributes twice to the sum in Eq. (4): once when $i = j$ and once when $i = k$. If $j \in L$ and $k \notin L$ (or vice verse), then this edge contributes only once. Finally, if $j, k \notin L$, then the edge does not contribute at all, but there are at most $\epsilon n/8$ edges of this latter type. Since it is possible to obtain such estimates $\hat{b}_i$ for all $i \in L$ simultaneously, with constant success probability, by sampling $O\left((\sqrt{n} \cdot \text{poly}(\log n, 1/\epsilon)\right)$ vertices, we can get a

E

$(2+\epsilon)$-factor estimate by performing this number of degree queries. Recall that we cannot obtain an approximation factor below 2 by performing $o(n)$ queries if we use only degree queries.

In order to obtain the desired factor of $(1+\epsilon)$, we estimate the number of edges (u,v) such that $u \in B_j$ and $v \in B_k$ with $j \in L$ and $k \notin L$, which are counted only once in Eq. (4). Here is where neighbor queries come into play. For each $i \in L$ (more precisely, for each i such that $\hat{b}_i$ is sufficiently large), we estimate $e_i \stackrel{\text{def}}{=} |\{(u,v) : u \in B_i, v \in B_k \text{ for } k \notin L\}|$. This is done by uniformly sampling *neighbors* of vertices in B_i, querying their degree, and therefore estimating the fraction of edges incident to vertices in B_i whose other endpoint belongs to B_k for $k \notin L$. If we denote the estimate of e_i by $\hat{e}_i$, then we can get that by performing $O\left((\sqrt{n} \cdot \text{poly}(\log n, 1/\epsilon)\right)$ neighbor queries, with high constant probability, the $\hat{e}_i$'s are such that

$$\frac{1}{n}\sum_{i \in L}\left(\hat{b}_i \cdot (1+\epsilon/8)^i + \hat{e}_i\right) \in \left[(1-\epsilon/2)\cdot \overline{d}(G), (1+\epsilon/2)\cdot \overline{d}(G)\right]. \tag{5}$$

By dividing the left-hand side in Eq. (5) by $(1-\epsilon/2)$, we obtain the $(1+\epsilon)$-factor we sought.

Estimating the Average Distance

Another graph parameter considered in [10] is the average distance between vertices. For this parameter, the algorithm is given access to a *distance-query oracle*. Namely, it can query the distance between any two vertices of its choice. As opposed to the average degree parameter where neighbor queries could be used to improve the quality of the estimate (and degree queries were not actually necessary), distance queries are crucial for estimating the average distance, and neighbor queries are not of much use. The main (positive) result concerning the average distance parameter is stated next.

Theorem 2 *There exists an algorithm that makes only distance queries to the input graph and satisfies the following condition. On input $G = (V,E)$ and $\epsilon \in (0,1)$, with probability at least $2/3$, the algorithm halts within $O\left(\sqrt{n/\overline{D}(G)} \cdot \text{poly}(1/\epsilon)\right)$ steps and outputs a value in $[\overline{D}(G), (1+\epsilon)\cdot \overline{D}(G)]$, where $\overline{D}(G)$ is the average of the all-pairs distances in G. A corresponding algorithm exists for the average distance to a given vertex $s \in V$.*

Comments for the Recommended Reading

The current entry falls within the scope of sublinear-time algorithms (see, e.g., [4]).

Other graph parameters that have been studied in the context of sublinear-time algorithms include the minimum weight of a spanning tree [2,3,5], the number of stars [11] and the number of triangles [6], the minimum size of a vertex cover [13–15,17], the size of a maximum matching [14, 17], and the distance to having various properties [8, 13, 16]. Related problems over weighted graphs that represent distance metrics were studied in [12] and [1].

Recommended Reading

1. Bădoiu M, Czumaj A, Indyk P, Sohler C (2005) Facility location in sublinear time. In: Automata, languages and programming: thirty-second international colloquium (ICALP), Lisbon, pp 866–877
2. Chazelle B, Rubinfeld R, Trevisan L (2005) Approximating the minimum spanning tree weight in sublinear time. SIAM J Comput 34(6):1370–1379
3. Czumaj A, Sohler C (2009) Estimating the weight of metric minimum spanning trees in sublinear time. SIAM J Comput 39(3):904–922
4. Czumaj A, Sohler C (2010) Sublinear-time algorithms, in [9]
5. Czumaj A, Ergun F, Fortnow L, Magen A, Newman I, Rubinfeld R, Sohler C (2005) Approximating the weight of the euclidean minimum spanning tree in sublinear time. SIAM J Comput 35(1):91–109
6. Eden T, Levi A, Ron D, Seshadhri C (2015) Approximately counting triangles in sublinear time. In: Proceedings of FOCS 2015, Berkeley. (To appear) (see also arXiv:1504.00954 and arXiv:1505.01927)
7. Feige U (2006) On sums of independent random variables with unbounded variance, and estimating

the average degree in a graph. SIAM J Comput 35(4):964–984
8. Fischer E, Newman I (2007) Testing versus estimation of graph properties. SIAM J Comput 37(2):482–501
9. Goldreich O (ed) (2010) Property testing: current research and surveys. LNCS, vol 6390. Springer, Heidelberg
10. Goldreich O, Ron D (2008) Approximating average parameters of graphs. Random Struct Algorithms 32(4):473–493
11. Gonen M, Ron D, Shavitt Y (2011) Counting stars and other small subgraphs in sublinear time. SIAM J Discret Math 25(3):1365–1411
12. Indyk P (1999) Sublinear-time algorithms for metric space problems. In: Proceedings of the thirty-first annual ACM symposium on the theory of computing (STOC), Atlanta, pp 428–434
13. Marko S, Ron D (2009) Distance approximation in bounded-degree and general sparse graphs. Trans Algorithms 5(2):article number 22
14. Nguyen HN, Onak K (2008) Constant-time approximation algorithms via local improvements. In: Proceedings of the forty-ninth annual symposium on foundations of computer science (FOCS), Philadelphia, pp 327–336
15. Parnas M, Ron D (2007) Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. Theor Comput Sci 381(1–3):183–196
16. Parnas M, Ron D, Rubinfeld R (2006) Tolerant property testing and distance approximation. J Comput Syst Sci 72(6):1012–1042
17. Yoshida Y, Yamamoto M, Ito H (2009) An improved constant-time approximation algorithm for maximum matchings. In: Proceedings of the fourty-first annual ACM symposium on the theory of computing (STOC), Bethesda, pp 225–234

Euclidean Traveling Salesman Problem

Artur Czumaj
Department of Computer Science, Centre for Discrete Mathematics and Its Applications, University of Warwick, Coventry, UK

Keywords

Approximation algorithms; Euclidean graphs; PTAS; TSP

Years and Authors of Summarized Original Work

1998; Arora
1999; Mitchell

Problem Definition

This entry considers geometric optimization $\mathcal{NP}$-hard problems like the Euclidean traveling salesman problem and the Euclidean Steiner tree problem. These problems are geometric variants of standard graph optimization problems, and the restriction of the input instances to geometric or Euclidean case arises in numerous applications (see [1, 2]). The main focus of this entry is on the Euclidean traveling salesman problem.

The Euclidean Traveling Salesman Problem (TSP)

For a given set S of n points in the Euclidean space $\mathbb{R}^d$, find the minimum length path that visits each point exactly once. The cost $\delta(x, y)$ of an edge connecting a pair of points $x, y \in \mathbb{R}^d$ is equal to the Euclidean distance between points x and y, that is, $\delta(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$, where $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$. More generally, the distance could be defined using other norms, such as ℓ_p norms for any $p > 1$, $\delta(x, y) = \left(\sum_{i=1}^{p} (x_i - y_i)^p \right)^{1/p}$.

For a given set S of points in Euclidean space $\mathbb{R}^d$, for a certain integer d, $d \geq 2$, a *Euclidean graph* (network) is a graph $G = (S, E)$, where E is a set of straight-line segments connecting pairs of points in S. If all pairs of points in S are connected by edges in E, then G is called a *complete Euclidean graph on S*. The cost of the graph is equal to the sum of the costs of the edges of the graph, $\text{cost}(G) = \sum_{(x,y)\in E} \delta(x, y)$.

A *polynomial-time approximation scheme (PTAS)* is a family of algorithms $\{\mathcal{A}_\varepsilon\}$ such that, for each fixed $\varepsilon > 0$, $\mathcal{A}_\varepsilon$ runs in polynomial time in the size of the input and produces a $(1 + \varepsilon)$-approximation.

Related Work

The classical book by Lawler et al. [16] provides extensive information about the TSP. Also, the survey exposition of Bern and Eppstein [8] presents the state of the art for geometric TSP until 1995, and the survey of Arora [2] discusses the research after 1995.

Key Results

We begin with the hardness results. The TSP in general graphs is well known to be $\mathcal{NP}$-hard, and the same claim holds for the Euclidean TSP [14, 18].

Theorem 1 *The Euclidean TSP is $\mathcal{NP}$-hard.*

Perhaps rather surprisingly, it is still not known if the decision version of the problem is $\mathcal{NP}$-complete [14]. (The decision version of the Euclidean TSP: given a point set in the Euclidean space $\mathbb{R}^d$ and a number t, verify if there is a simple path of length smaller than t that visits each point exactly once.)

The approximability of TSP has been studied extensively over the last few decades. It is not hard to see that TSP is not approximable in polynomial time (unless $\mathcal{P} = \mathcal{NP}$) for arbitrary graphs with arbitrary edge costs. When the weights satisfy the triangle inequality (the so-called metric TSP), there is a polynomial-time 3/2-approximation algorithm due to Christofides [9], and it is known that no PTAS exists (unless $\mathcal{P} = \mathcal{NP}$). This result has been strengthened by Trevisan [21] to include Euclidean graphs in high dimensions (the same result holds also for any ℓ_p metric).

Theorem 2 (Trevisan [21]) *If $d \geq \log n$, then there exists a constant $\varepsilon > 0$ such that the Euclidean TSP in $\mathbb{R}^d$ is $\mathcal{NP}$-hard to approximate within a factor of $1 + \varepsilon$.*

In particular, this result implies that if $d \geq \log n$, then the Euclidean TSP in $\mathbb{R}^d$ has no PTAS unless $\mathcal{P} = \mathcal{NP}$.

The same result holds also for any ℓ_p metric. Furthermore, Theorem 2 implies that Euclidean TSP in $\mathbb{R}^{\log n}$ is *APX PB*-hard under E-reductions and *APX*-complete under AP-reductions.

It has been believed for some time that Theorem 2 might hold for smaller values of d, in particular even for $d = 2$, but this has been disproved independently by Arora [1] and Mitchell [17].

Theorem 3 (Arora [1] and Mitchell [17]) *The Euclidean TSP on the plane has a PTAS.*

The main idea of the algorithms of Arora and Mitchell is rather simple, but the details of the analysis are quite complicated. Both algorithms follow the same approach. One first proves a so-called structure theorem, which demonstrates that there is a $(1 + \varepsilon)$-approximation that has some local properties (in the case of the Euclidean TSP, there is a quadtree partition of the space containing all the points such that there is a $(1 + \varepsilon)$-approximation in which each cell of the quadtree is crossed by the tour at most a constant number of times and only in some prespecified locations). Then, one uses dynamic programming to find an optimal (or almost optimal) solution that obeys the local properties specified in the structure theorem.

The original algorithms presented in the first conference version of [1] and in the early version of [17] have the running times of the form $\mathcal{O}(n^{1/\epsilon})$ to obtain a $(1 + \varepsilon)$-approximation, but this has been subsequently improved. In particular, Arora's randomized algorithm in [1] runs in time $\mathcal{O}(n(\log n)^{1/\epsilon})$, and it can be derandomized with a slowdown of $\mathcal{O}(n)$. The result from Theorem 3 can be also extended to higher dimensions. Arora shows the following result.

Theorem 4 (Arora [1]) *For every constant d, the Euclidean TSP in $\mathbb{R}^d$ has a PTAS.*

For every fixed $c > 1$ and given any n points in $\mathbb{R}^d$, there is a randomized algorithm that finds a $\left(1 + \frac{1}{c}\right)$-approximation of the optimum traveling salesman tour in $\mathcal{O}\left(n(\log n)^{(\mathcal{O}(\sqrt{dc}))^{d-1}}\right)$ time. In particular, for any constant d and c, the running time is $\mathcal{O}\left(n(\log n)^{\mathcal{O}(1)}\right)$. The algorithm can be derandomized by increasing the running time by a factor of $\mathcal{O}(n^d)$.

This has been later extended by Rao and Smith [19], who proved the following.

Theorem 5 (Rao and Smith [19]) *There is a deterministic algorithm that computes a* $\left(1+\frac{1}{c}\right)$*-approximation of the optimum traveling salesman tour in* $\mathcal{O}\left(2^{(cd)^{\mathcal{O}(d)}}n + (cd)^{\mathcal{O}(d)}n \log n\right)$ *time.*

There is a randomized algorithm that succeeds with probability at least $\frac{1}{2}$ *and that computes a* $\left(1+\frac{1}{c}\right)$*-approximation of the optimum traveling salesman tour in expected* $\left(c\sqrt{d}\right)^{\mathcal{O}\left(d(c\sqrt{d})^{d-1}\right)} n + \mathcal{O}(d\, n \log n)$ *time.*

These results are essentially asymptotically optimal in the decision tree model thanks to a lower bound of $\Omega(n \log n)$ for any sublinear approximation for 1-dimensional Euclidean TSP due to Das et al. [12]. In the *real RAM* model, one can further improve the randomized results.

Theorem 6 (Bartal and Gottlieb [6]) *Given a set* S *of* n *points in* d*-dimensional grid* $\{0,\ldots,\Delta\}^d$ *with* $\Delta = 2^{(cd)^{\mathcal{O}(d)}}n$*, there is a randomized algorithm that with probability* $1 - e^{-\mathcal{O}_d(n^{1/3d})}$ *computes a* $\left(1+\frac{1}{c}\right)$*-approximation of the optimum traveling salesman tour for* S *in time* $2^{(cd)^{\mathcal{O}(d)}}n$ *in the integer RAM model.*

If the data is not given in the integral form, then one may round the data into this form using the floor or mod functions, and assuming these functions are atomic operations, the rounding can be done in $\mathcal{O}(dn)$ total time, leading to the following theorem.

Theorem 7 (Bartal and Gottlieb [6]) *Given a set of* n *points in* $\mathbb{R}^d$*, there is a randomized algorithm that with probability* $1 - e^{-\mathcal{O}_d(n^{1/3d})}$ *computes a* $\left(1+\frac{1}{c}\right)$*-approximation of the optimum traveling salesman tour in time* $2^{(cd)^{\mathcal{O}(d)}}n$ *in the real RAM model with atomic floor or mod operations.*

Applications

The techniques developed by Arora [1] and Mitchell [17] found numerous applications in the design of polynomial-time approximation schemes for geometric optimization problems.

Euclidean Minimum Steiner Tree

For a given set S of n points in the Euclidean space $\mathbb{R}^d$, find the minimum-cost network connecting all the points in S (where the cost of a network is equal to the sum of the lengths of the edges defining it).

Euclidean k-median

For a given set S of n points in the Euclidean space $\mathbb{R}^d$ and an integer k, find k-medians among the points in S so that the sum of the distances from each point in S to its closest median is minimized.

Euclidean k-TSP

For a given set S of n points in the Euclidean space $\mathbb{R}^d$ and an integer k, find the shortest tour that visits at least k points in S.

Euclidean k-MST

For a given set S of n points in the Euclidean space $\mathbb{R}^d$ and an integer k, find the shortest tree that visits at least k points in S.

Euclidean Minimum-Cost k-Connected Subgraph

For a given set S of n points in the Euclidean space $\mathbb{R}^d$ and an integer k, find the minimum-cost subgraph (of the complete graph on S) that is k-connected.

Theorem 8 *For every constant d, the following problems have a PTAS:*

- *Euclidean minimum Steiner tree problem in* $\mathbb{R}^d$ *[1, 19]*
- *Euclidean k-median problem in* $\mathbb{R}^d$ *[5]*
- *Euclidean k-TSP and the Euclidean k-MST problems in* $\mathbb{R}^d$ *[1]*
- *Euclidean minimum-cost k-connected subgraph problem in* $\mathbb{R}^d$ *(constant k) [10]*

The technique developed by Arora [1] and Mitchell [17] led also to some quasi-polynomial-time approximation schemes, that is, the algorithms with the running time of $n^{\mathcal{O}(\log n)}$. For example, Arora and Karokostas [4] gave a quasi-polynomial-time approximation scheme

for the Euclidean minimum latency problem, Das and Mathieu [13] gave a quasi-polynomial-time approximation scheme for the Euclidean capacitated vehicle routing problem, and Remy and Steger [20] gave a quasi-polynomial-time approximation scheme for the minimum-weight triangulation problem.

For more discussion, see the survey by Arora [2] and Czumaj and Lingas [11].

Extensions to Planar Graphs and Metric Spaces with Bounded Doubling Dimension

The dynamic programming approach used by Arora [1] and Mitchell [17] is also related to the recent advances for a number of optimization problems for planar graphs and in graphs in metric spaces with bounded doubling dimension. For example, Arora et al. [3] designed a PTAS for the TSP in weighted planar graphs (cf. [15] for a linear-time PTAS), and there is a PTAS for metric spaces with bounded doubling dimension [7].

Open Problems

An interesting open problem is if the quasi-polynomial-time approximation schemes mentioned above (for the minimum latency, the capacitated vehicle routing, and the minimum-weight triangulation problems) can be extended to obtain PTAS. For more open problems, see Arora [2].

Experimental Results

The Web page of the 8th DIMACS Implementation Challenge, http://dimacs.rutgers.edu/Challenges/TSP/, contains a lot of instances.

URLs to Code and Data Sets

The Web page of the 8th DIMACS Implementation Challenge, http://dimacs.rutgers.edu/Challenges/TSP/, contains a lot of instances.

Cross-References

- Approximation Schemes for Geometric Network Optimization Problems
- Metric TSP
- Minimum k-Connected Geometric Networks
- Minimum Weight Triangulation

Recommended Reading

1. Arora S (1998) Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J Assoc Comput Mach 45(5):753–782
2. Arora S (2003) Approximation schemes for NP-hard geometric optimization problems: a survey. Math Program Ser B 97:43–69
3. Arora S, Grigni M, Karger D, Klein P, Woloszyn A (1998) A polynomial time approximation scheme for weighted planar graph TSP. In: Proceedings of the 9th annual ACM-SIAM symposium on discrete algorithms (SODA), San Francisco, pp 33–41
4. Arora S, Karakostas G (1999) Approximation schemes for minimum latency problems. In: Proceedings of the 31st annual ACM symposium on theory of computing (STOC), Atlanta, pp 688–693
5. Arora S, Raghavan P, Rao S (1998) Approximation schemes for Euclidean k-medians and related problems. In: Proceedings of the 30th annual ACM symposium on theory of computing (STOC), Dallas, pp 106–113
6. Bartal Y, Gottlieb LA (2013) A linear time approximation scheme for Euclidean TSP. In: Proceedings of the 54th IEEE symposium on foundations of computer science (FOCS), Berkeley, pp 698–706
7. Bartal Y, Gottlieb LA, Krauthgamer R (2012) The traveling salesman problem: low-dimensionality implies a polynomial time approximation scheme. In: Proceedings of the 44th annual ACM symposium on theory of computing (STOC), New York, pp 663–672
8. Bern M, Eppstein D (1996) Approximation algorithms for geometric problems. In: Hochbaum D (ed) Approximation algorithms for NP-hard problems. PWS Publishing, Boston
9. Christofides N (1976) Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh
10. Czumaj A, Lingas A (1999) On approximability of the minimum-cost k-connected spanning subgraph problem. In: Proceedings of the 10th annual ACM-SIAM symposium on discrete algorithms (SODA), Baltimore, pp 281–290
11. Czumaj A, Lingas A (2007) Approximation schemes for minimum-cost k-connectivity problems in geometric graphs. In: Gonzalez TF (ed) Handbook of

approximation algorithms and metaheuristics. CRC, Boca Raton
12. Das G, Kapoor S, Smid M (1997) On the complexity of approximating Euclidean traveling salesman tours and minimum spanning trees. Algorithmica 19(4):447–462
13. Das A, Mathieu C (2010) A quasi-polynomial time approximation scheme for Euclidean capacitated vehicle routing. In: Proceedings of the 21st annual ACM-SIAM symposium on discrete algorithms (SODA), Austin, pp 390–403
14. Garey MR, Graham RL, Johnson DS (1976) Some NP-complete geometric problems. In: Proceedings of the 8th annual ACM symposium on theory of computing (STOC), Hershey, pp 10–22
15. Klein P (2008) A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. SIAM J Comput 37(6):1926–1952
16. Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB (1985) The traveling salesman problem: a guided tour of combinatorial optimization. Wiley, Chichester/New York
17. Mitchell JSB (1999) Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. SIAM J Comput 28(4):1298–1309
18. Papadimitriou CH (1977) Euclidean TSP is NP-complete. Theor Comput Sci 4:237–244
19. Rao SB, Smith WD (1998) Approximating geometrical graphs via "spanners" and "banyans." In: Proceedings of the 30th annual ACM symposium on theory of computing (STOC), Dallas, pp 540–550
20. Remy J, Steger A (2006) A quasi-polynomial time approximation scheme for minimum weight triangulation. In: Proceedings of the 38th annual ACM symposium on theory of computing (STOC), Seattle, pp 316–325
21. Trevisan L (2000) When Hamming meets Euclid: the approximability of geometric TSP and Steiner tree. SIAM J Comput 30(2):475–485

Exact Algorithms and Strong Exponential Time Hypothesis

Joshua R. Wang and Ryan Williams
Department of Computer Science, Stanford University, Stanford, CA, USA

Keywords

Exact algorithms; Exponential-time hypothesis; Satisfiability; Treewidth

Years and Authors of Summarized Original Work

2010; Pătraşscu, Williams
2011; Lokshtanov, Marx, Saurabh
2012; Cygan, Dell, Lokshtanov, Marx, Nederlof, Okamoto, Paturi, Saurabh, Wahlström

Problem Definition

All problems in NP can be exactly solved in $2^{\text{poly}(n)}$ time via exhaustive search, but research has yielded faster exponential-time algorithms for many NP-hard problems. However, some key problems have not seen improved algorithms, and problems with improvements seem to converge toward $O(C^n)$ for some unknown constant $C > 1$.

The satisfiability problem for Boolean formulas in conjunctive normal form, CNF-SAT, is a central problem that has resisted significant improvements. The complexity of CNF-SAT and its special case k-SAT, where each clause has k literals, is the canonical starting point for the development of NP-completeness theory.

Similarly, in the last 20 years, two hypotheses have emerged as powerful starting points for understanding exponential-time complexity. In 1999, Impagliazzo and Paturi [5] defined the *exponential-time hypothesis* (ETH), which asserts that 3-*SAT cannot be solved in subexponential time*. Namely, it asserts there is an $\epsilon > 0$ such that 3-SAT cannot be solved in $O((1+\epsilon)^n)$ time. ETH has been a surprisingly useful assumption for ruling out subexponential-time algorithms for other problems [2, 6]. A stronger hypothesis has led to more fine-grained lower bounds, which is the focus of this article. Many NP-hard problems are solvable in C^n time via exhaustive search (for some $C > 1$) but are not known to be solvable in $(C-\epsilon)^n$ time, for any $\epsilon > 0$. The *strong exponential-time hypothesis* (SETH) [1,5] asserts that for every $\epsilon > 0$, there exists a k such that k-SAT cannot be solved in time $O((2-\epsilon)^n)$. SETH has been very useful in establishing tight (and

exact) lower bounds for many problems. Here we survey some of these tight results.

Key Results

The following results are reductions from k-SAT to other problems. They can be seen either as new attacks on the complexity of SAT or as lower bounds for exact algorithms that are conditional on SETH.

Lower Bounds on General Problems

The following problems have lower bounds conditional on SETH. The reduction for the first problem is given to illustrate the technique.

k-Dominating Set

A *dominating set* of a graph $G = (V, E)$ is a subset $S \subseteq V$ such that every vertex is either in S or is a neighbor of a vertex in S. The k-DOMINATING SET problem asks to find a dominating set of size k. Assuming SETH, for any $k \geq 3$ and $\epsilon > 0$, k-DOMINATING SET cannot be solved in $O(n^{k-\epsilon})$ time [8].

The reduction from SAT to k-DOMINATING SET proceeds as follows. Fix some $k \geq 3$ and let F be a CNF formula on n variables; we build a corresponding graph G_F. Partition its variables into k equally sized parts of n/k variables. For each part, take all $2^{n/k}$ partial assignments and make a node for each partial assignment. Make each of the k parts into a clique (disjoint from the others). Add a dummy node for each partial assignment clique that is connected to every node in that clique but has no other edges. Add m more nodes, one for each clause. Finally, make an edge from a partial assignment node to a clause node iff the partial assignment satisfies the clause. We observe that there is a k-dominating set in G_F if F is satisfiable.

2Sat+2Clauses

The 2SAT+2CLAUSES problem asks whether a Boolean formula is satisfiable, given that it is a 2-CNF with two additional clauses of arbitrary length. Assuming SETH, for any $m = n^{1+o(1)}$ and $\epsilon > 0$, 2SAT+2CLAUSES cannot be solved in $O(n^{2-\epsilon})$ time [8]. It is known that 2SAT+2CLAUSES can be solved in $O(mn + n^2)$ time [8].

HornSat+kClauses

The HORNSAT+kCLAUSES problem asks whether a Boolean formula is satisfiable, given that it is a CNF of clauses that contain at most one nonnegative literal per clause (a Horn CNF), conjoined with k additional clauses of arbitrary length but only positive literals. Assuming SETH, for any $k \geq 2$ and $\epsilon > 0$, HORNSAT+kCLAUSES cannot be solved in $O((n + m)^{k-\epsilon})$ time [8]. It can be trivially solved in $O(n^k \cdot (m + n))$ time by guessing a variable to set to true for each of the k additional clauses and checking if the remaining Horn CNF is satisfiable in linear time.

3-Party Set Disjointness

The 3-PARTY SET DISJOINTNESS problem is a communication problem with three parties and three subsets $S_1, S_2, S_3 \subseteq [m]$, where the ith party has access to all sets except for S_i. The parties wish to determine if $S_1 \cap \cdots \cap S_3 = \varnothing$. Clearly this can be done with $O(m)$ bits of communication. Assuming SETH, 3-PARTY SET DISJOINTNESS cannot be solved using protocols running in $2^{o(n)}$ time and communicating only $o(m)$ bits [8].

k-SUM

The k-SUM problem asks whether a set of n numbers contains a k-tuple that sums to zero. Assuming SETH, k-SUM on n numbers cannot be solved in $n^{o(k)}$ time for any $k < n^{0.99}$. (It is well known that k-SUM is in $O(n^{\lceil k/2 \rceil})$ time) [8].

For all the problems below, we can solve in $2^n n^{O}(1)$ time via exhaustive search.

k-Hitting Set

Given a set system $\mathcal{F} \subseteq 2^U$ in some universe U, a *hitting set* is a subset $H \subseteq U$ such that $H \cap S \neq \varnothing$ for every $S \in \mathcal{F}$. The k-HITTINGSET problem asks whether there is a hitting set of size at most t, given that each set $S \in \mathcal{F}$ has at most k elements. SETH is equivalent to the claim that for all $\epsilon > 0$,

there is a k for which k-HittingSet cannot be solved in time $O((2-\epsilon)^n)$ [3].

k-Set Splitting

Given a set system $\mathcal{F} \subseteq 2^U$ in some universe U, a set splitting is a subset $X \subseteq U$ such that the first element of the universe is in X and for every $S \in \mathcal{F}$, neither $S \subseteq X$ nor $S \subseteq (U \setminus X)$. The k-SetSplitting problem asks whether there is a set splitting, given that each set $S \in \mathcal{F}$ has at most k elements. SETH is equivalent to the claim that for all $\epsilon > 0$, there is a k for which k-SetSplitting cannot be solved in time $O((2-\epsilon)^n)$ [3].

k-NAE-Sat

The k-NAE-Sat problem asks whether a k-CNF has an assignment where the first variable is set to true and each clause has both a true literal and a false literal. SETH is equivalent to the claim that for all $\epsilon > 0$, there is a k for which k-NAE-Sat cannot be solved in time $O((2-\epsilon)^n)$ [3].

c-VSP-Circuit-SAT

The c-VSP-Circuit-SAT problem asks whether a cn-size Valiant series-parallel circuit over n variables has a satisfying assignment. SETH is equivalent to the claim that for all $\epsilon > 0$, there is a k for which c-VSP-Circuit-SAT cannot be solved in time $O((2-\epsilon)^n)$ [3].

Problems Parameterized by Treewidth

A variety of NP-complete problems have been shown to be much easier on graphs of bounded treewidth. Reductions starting from SETH given by Lokshtanov, Marx, and Saurabh [7] can also prove lower bounds that depend on the treewidth of an input graph, $\mathrm{tw}(G)$. The following are proven via analyzing the pathwidth of a graph, $\mathrm{pw}(G)$, and the fact that $\mathrm{tw}(G) \leq \mathrm{pw}(G)$.

Independent Set

An *independent set* of a graph $G = (V, E)$ is a subset $S \subseteq V$ such that the subgraph induced by S contains no edges. The Independent Set problem asks to find an independent set of maximum size. Assuming SETH, for any $\epsilon > 0$, Independent Set cannot be solved in $(2-\epsilon)^{\mathrm{tw}(G)} n^{O(1)}$ time.

Dominating Set

A *dominating set* of a graph $G = (V, E)$ is a subset $S \subseteq V$ such that every vertex is either in S or is a neighbor of a vertex in S. The Dominating Set problem asks to find a dominating set of minimum size. Assuming SETH, for any $\epsilon > 0$, Dominating Set cannot be solved in $(3-\epsilon)^{\mathrm{tw}(G)} n^{O(1)}$ time.

Max Cut

A *cut* of a graph $G = (V, E)$ is a partition of V into S and $V \setminus S$. The size of a cut is the number of edges that have one endpoint in S and the other in $V \setminus S$. The Max Cut problem asks to find a cut of maximum size. Assuming SETH, for any $\epsilon > 0$, Max Cut cannot be solved in $(2-\epsilon)^{\mathrm{tw}(G)} n^{O(1)}$ time.

Odd Cycle Transversal

An *odd cycle transversal* of a graph $G = (V, E)$ is a subset $S \subseteq V$ such that the subgraph induced by $V \setminus S$ is bipartite. The Odd Cycle Transversal problem asks to, given an integer k, determine whether there is an odd cycle transversal of size k. Assuming SETH, for any $\epsilon > 0$, Odd Cycle Transversal cannot be solved in $(3-\epsilon)^{\mathrm{tw}(G)} n^{O(1)}$ time.

Graph Coloring

A *q-coloring* of a graph $G = (V, E)$ is a function $\mu : V \to [q]$. A q-coloring is *proper* if for all edges $(u, v) \in E$, $\mu(u) \neq \mu(v)$. The q-coloring problem asks to decide whether the graph has a proper q-coloring. Assuming SETH, for any $q \geq 3$ and $\epsilon > 0$, q-coloring cannot be solved in $(q-\epsilon)^{\mathrm{tw}(G)} n^{O(1)}$ time.

Partition Into Triangles

A graph $G = (V, E)$ can be partitioned into triangles if there is a partition of the vertices into $S_1, S_2, \ldots, S_{n/3}$ such that each S_i induces a triangle in G. The Partition Into Triangles problem asks to decide whether the graph can be partitioned into triangles. Assuming SETH, for

any $\epsilon > 0$, PARTITION INTO TRIANGLES cannot be solved in $(2-\epsilon)^{\mathrm{tw}(G)} n^{O(1)}$ time.

All of the above results are tight, in the sense that when $\epsilon = 0$, there is an algorithm for each of them.

Showing Difficulty Via Set Cover

Given a set system $\mathcal{F} \subseteq 2^U$ in some universe U, a set cover is a subset $\mathcal{C} \subseteq \mathcal{F}$ such that $\bigcup_{S \in \mathcal{C}} S = U$. The SET COVER problem asks whether there is a set cover of size at most t.

Cygan et al. [3] also gave reductions from SET COVER to several other problems, showing lower bounds conditional on the assumption that for all $\epsilon > 0$, there is a k such that SET COVER where sets in $\mathcal{F}$ have size at most k cannot be computed in time $O^*((2-\epsilon)^n)$.

It is currently unknown how SET COVER is related to SETH; if there is a reduction from CNF-SAT to SET COVER, then all of these problems would have conditional lower bounds as well.

Steiner Tree

Given a graph $G = (V, E)$ and a set of terminals $T \subseteq V$, a *Steiner Tree* is a subset $X \subseteq V$ such that the graph induced by X is connected and $T \subseteq X$. The STEINER TREE problem asks whether G has a Steiner tree of size at most t. With the above SET COVER assumption, for all $\epsilon > 0$, STEINER TREE cannot be solved in $O^*((2-\epsilon)^t)$ time.

Connected Vertex Cover

A *connected vertex cover* of a graph $G = (V, E)$ is a subset $X \subseteq V$ such that the subgraph induced by X is connected and every edge contains at least one endpoint in X. The CONNECTED VERTEX COVER problem asks whether G has a connected vertex cover of size at most t. With the above SET COVER assumption, for all $\epsilon > 0$, CONNECTED VERTEX COVER cannot be solved in $O^*((2-\epsilon)^t)$ time.

Set Partitioning

Given a set system $\mathcal{F} \subseteq 2^U$ in some universe U, a set partitioning is a set cover $\mathcal{C}$ where pairwise disjoint elements have an empty intersection. The SET PARTITIONING problem asks whether there is a set partitioning of size at most t. With the above SET COVER assumption, for all $\epsilon > 0$, SET PARTITIONING cannot be solved in $O^*((2-\epsilon)^n)$ time.

Subset Sum

The SUBSET SUM problem asks whether a set of n positive numbers contains a subset that sums to a target t. With the above SET COVER assumption, for all $\delta < 1$, SUBSET SUM cannot be solved in $O^*(t^\delta)$ time. Note that there is a dynamic programming solution that runs in $O(nt)$ time.

Open Problems

- Does ETH imply SETH?
- Does SETH imply SET COVER requires $O^*((2-\epsilon)^n)$ time for all $\epsilon > 0$?
- Does SETH imply that the Traveling Salesman Problem in its most general, weighted form requires $O^*((2-\epsilon)^n)$ time for all $\epsilon > 0$?
- Given two graphs F and G, on k and n nodes, respectively, the SUBGRAPH ISOMORPHISM problem asks whether a (noninduced) subgraph of G is isomorphic to F. Does SETH imply that SUBGRAPH ISOMORPHISM cannot be solved in $2^{O(n)}$?

Cross-References

- ▸ Backtracking Based k-SAT Algorithms
- ▸ Exact Algorithms for General CNF SAT
- ▸ Exact Algorithms for Treewidth

Recommended Reading

1. Calabro C, Impagliazzo R, Paturi R (2009) The complexity of satisfiability of small depth circuits. In: Chen J, Fomin F (eds) Parameterized and exact computation. Lecture notes in computer science, vol 5917. Springer, Berlin/Heidelberg, pp 75–85. doi:10.1007/978-3-642-11269-0_6. http://dx.doi.org/10.1007/978-3-642-11269-0_6
2. Chen J, Chor B, Fellows M, Huang X, Juedes D, Kanj IA, Xia G (2005) Tight lower bounds for certain parameterized np-hard problems. Inf Comput 201(2):216–231. doi:http://dx.doi.org/10.1016/j.ic.

2005.05.001. http://www.sciencedirect.com/science/article/pii/S0890540105000763
3. Cygan M, Dell H, Lokshtanov D, Marx D, Nederlof J, Okamoto Y, Paturi R, Saurabh S, Wahlstrom M (2012) On problems as hard as cnf-sat. In: Proceedings of the 2012 IEEE conference on computational complexity (CCC '12), Washington, DC. IEEE Computer Society, pp 74–84. doi:10.1109/CCC.2012.36. http://dx.doi.org/10.1109/CCC.2012.36
4. Fomin F, Kratsch D (2010) Exact exponential algorithms. Texts in theoretical computer science, an EATCS series. Springer, Berlin/Heidelberg
5. Impagliazzo R, Paturi R (2001) On the complexity of k-sat. J Comput Syst Sci 62(2):367–375. doi:http://dx.doi.org/10.1006/jcss.2000.1727. http://www.sciencedirect.com/science/article/pii/S0022000000917276
6. Impagliazzo R, Paturi R, Zane F (2001) Which problems have strongly exponential complexity? J Comput Syst Sci 63(4):512–530. doi:http://dx.doi.org/10.1006/jcss.2001.1774. http://www.sciencedirect.com/science/article/pii/S002200000191774X
7. Lokshtanov D, Marx D, Saurabh S (2011) Known algorithms on graphs of bounded treewidth are probably optimal. In: Proceedings of the twenty-second Annual ACM-SIAM symposium on discrete algorithms (SODA '11), San Francisco. SIAM, pp 777–789. http://dl.acm.org/citation.cfm?id=2133036.2133097
8. Pătraşcu M, Williams R (2010) On the possibility of faster sat algorithms. In: Proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms (SODA '10), Philadelphia. Society for Industrial and Applied Mathematics, pp 1065–1075. http://dl.acm.org/citation.cfm?id=1873601.1873687
9. Woeginger G (2003) Exact algorithms for np-hard problems: a survey. In: Jünger M, Reinelt G, Rinaldi G (eds) Combinatorial optimization Eureka, you shrink! Lecture notes in computer science, vol 2570. Springer, Berlin/Heidelberg, pp 185–207. doi:10.1007/3-540-36478-1_17. http://dx.doi.org/10.1007/3-540-36478-1_17

Exact Algorithms and Time/Space Tradeoffs

Jesper Nederlof
Technical University of Eindhoven, Eindhoven, The Netherlands

Keywords

Dynamic programming; Knapsack; Space efficiency; Steiner tree; Subset discrete Fourier transform

Years and Authors of Summarized Original Work

2010; Lokshtanov, Nederlof

Problem Definition

In the *subset sum problem*, we are given integers $a_1, \ldots, a_n, t$ and are asked to find a subset $X \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in X} a_i = t$. In the *Knapsack problem*, we are given $a_1, \ldots, a_n, b_1, \ldots, b_n, t, u$ and are asked to find a subset $X \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in X} a_i \leq t$ and $\sum_{i \in X} b_i \geq u$. It is well known that both problems can be solved in $O(nt)$ time using dynamic programming. However, as is typical for dynamic programming, these algorithms require a lot of working memory and are relatively hard to execute in parallel on several processors: the above algorithms use $O(t)$ space which may be *exponential* in the input size.

This raises the question: when can we avoid these disadvantages and still be (approximately) as fast as dynamic programming algorithms? It appears that by (slightly) loosening the time budget, space usage and parallelization can be significantly improved in many dynamic programs.

Key Results

A Space Efficient Algorithm for Subset Sum

In this article, we will use $\tilde{O}(\cdot)$ to suppress factors that are poly-logarithmic in the input size. In what follows, we will discuss how to prove the following theorem:

Theorem 1 (Lokshtanov and Nederlof, [7]) *There is an algorithm counting the number of solutions of a subset sum instance in* $\tilde{O}(n^2 t(n + \log t))$ *time and* $(n + lg(t))(\lg nt)$ *space.*

The Discrete Fourier Transform

We use Iverson's bracket notation: given a Boolean predicate b, $[b]$ denotes 1 if b is true

and 0 otherwise. Let $P(x)$ be a polynomial of degree $N-1$, and let $p_0,\ldots,p_{N-1}$ be its coefficients. Thus, $P(x) = \sum_{i=0}^{N-1} p_i x^i$. Let ω_N denote the N'th root of unity, that is, $\omega_N = e^{\frac{2\pi\iota}{N}}$. Let k,t be integers such that $k \neq t$. By the summation formula for geometric progressions $\left(\sum_{\ell=0}^{N-1} r^\ell = \frac{1-r^N}{1-r} \text{for } r \neq 1\right)$, we have:

$$\sum_{\ell=0}^{N-1} \omega_N^{\ell(k-t)} = \frac{1-\omega_N^{(k-t)N}}{1-\omega_N^{k-t}} = \frac{1-\left(\omega_N^N\right)^{k-t}}{1-\omega_N^{k-t}}$$
$$= \frac{1-(1)^{k-t}}{1-\omega_N^{k-t}} = 0.$$

On the other hand, if $k=t$, then $\sum_{\ell=0}^{N-1} \omega_N^{\ell(k-t)} = \sum_{\ell=0}^{N-1} 1 = N$. Thus, both cases can be compactly summarized as $\sum_{\ell=0}^{N-1} \omega_N^{\ell(k-t)} = [k=t]N$. As a consequence, we can express a coefficient p_t of $P(x)$ directly in terms of its evaluations:

$$\begin{aligned} p_t &= \sum_{k=0}^{N-1} [k=t] p_k \\ &= \sum_{k=0}^{N-1} \frac{1}{N} \sum_{\ell=0}^{N-1} \omega_N^{\ell(k-t)} \sum p_k \\ &= \frac{1}{N} \sum_{\ell=0}^{N-1} \omega_N^{-\ell t} \sum_{k=0}^{N-1} p_k \left(\omega_N^\ell\right)^k \\ &= \frac{1}{N} \sum_{\ell=0}^{N-1} \omega_N^{-\ell t} P(\omega_N^\ell) \end{aligned} \tag{1}$$

Using the Discrete Fourier Transform for Subset Sum

Given an instance $a_1,\ldots,a_n,t$ of subset sum, define the polynomial $P(x)$ to be $P(x) = \prod_{i=1}^n (1+x^{a_i})$. Clearly, we can discard integers a_i larger than t, and assume that $P(x)$ has degree at most $N = nt$. If we expand the products in this polynomial to get rid of the parentheses, we get a sum of 2^n products and each of these products is of the type x^k and corresponds to a subset $X \subseteq \{1,\ldots,n\}$ such that $\sum_{i\in X} a_i = k$. Thus, if we aggregate these products, we obtained the normal form $P(x) = \sum_{k=0}^{N-1} p_k x^k$, where p_k equals the number of subsets $X \subseteq [n]$ such that $\sum_{i\in X} a_i = k$. Plugging this into Eq. 1, we have that the number of subset sum solutions equals

$$p_t = \frac{1}{N} \sum_{\ell=0}^{N-1} \omega_N^{-\ell t} \prod_{i=1}^{n} \left(1+\omega_N^{\ell a_i}\right). \tag{2}$$

Given Eq. 2, the algorithm suggests itself: evaluation of the right-hand side gives the number of solutions of the subset sum instance. Given ω_N, this would be a straightforward on the unit-cost RAM model (recall that in this model arithmetic instructions as $+,-,*$ and $/$ are assumed to take constant time): the required powering operations are performed in $\log(N)$ arithmetic operations so an overall upper bound would be $O(n^2 t \log(nt))$ time.

However, still the value of this algorithm is not clear yet: for example, ω_N may be irrational, so it is not clear how to perform the arithmetic efficiently. This is an issue that also arises for the folklore fast Fourier transform (see, e.g., [3] for a nice exposition), and this issue is usually not addressed (a nice exception is Knuth [6]). Moreover in our case we should also be careful on the space budget: for example, we cannot even store 2^t in the usual way within our space budget. But, as we will now see, it turns out that we can simply evaluate Eq. 2 with finite precision and round to the nearest integer within the resource bounds claimed in Theorem 1.

Evaluating Equation 2 with Finite Precision

The algorithm establishing Theorem 1 is presented in Algorithm 1. Here, ρ represents the amount of precision the algorithm works with. The procedure $\texttt{tr}_\rho$ truncates ρ bits after the decimal point. The procedure $\texttt{apxr}_\rho(z)$ returns an estimate of ω_N^z. In order to do this, estimates of ω_N^z with z being powers of 2 are precomputed in Lines 3–4. We omit an explicit implementation of the right-hand side of Line 4 since this is very standard; for example, one can use an approximation of π together with a binary splitting approach

Algorithm 1 Approximate evaluation of Eq. 2

Algorithm: SSS$(a_1, \ldots, a_n, t)$
Require: for every $1 \le i \le n, a_i < t$.
1: $\rho \leftarrow 3n + 6 \log nt$
2: $s \leftarrow 0$
3: **for** $0 \le q \le \log N - 1$ **do**
4: //store roots of unity for powers of two
5: $r_q \leftarrow \mathtt{tr}_\rho(e^{\frac{2\pi 2^q}{Nt}})$
6: **end for**
7: **for** $0 \le \ell \le N - 1$ **do**
8: $p \leftarrow \mathtt{apxr}_\rho(-\ell t \ \% \ N)$
9: **for** $1 \le i \le n$ **do**
10: $p \leftarrow \mathtt{tr}_\rho(p * (1 + \mathtt{apxr}_\rho(\ell a_i \ \% \ N)))$
11: **end for**
12: $s \leftarrow s + p$
13: **end for**
14: **return** $\mathtt{rnd}(\mathtt{tr}_\rho(\frac{s}{N}))$ //round to nearest int

Algorithm: $\mathtt{apxr}_\rho(z)$
Require: $z < N$
15: $p' \leftarrow 1$
16: **for** $1 \le q \le \log N - 1$ **do**
17: **if** 2^q divides z **then**
18: $p \leftarrow \mathtt{tr}_\rho(p' * r_q)$
19: **end if**
20: **end for**
21: **return** p'

(see [1, Section 4.9.1]) or a Taylor expansion-based approach (see [2]). Crude upper bounds on the time and space usage of both approaches are $O(\rho^2 \log N)$ time and $O(n \log nt + \log nt)$ space.

Let us proceed with verifying whether Algorithm 1 satisfies the resource bounds of Theorem 1. It is easy to see that all intermediate numbers have modulus at most $2^n N$, so their estimates can be represented with $O(\rho)$ bits. For all multiplications we will use an asymptotically fast algorithm running in $\tilde{O}(n)$ time (e.g., [5]). Then, Line 3–4 take $\tilde{O}(\rho^2 \log N)$ time. Line 6 takes $\tilde{O}(\rho \lg N)$ time; Lines 7–8 take $n\rho \lg N$ time, which is the bottleneck. So overall, the algorithm uses $\tilde{O}(Nn\rho) = \tilde{O}(n^2 t(n + \log t))$ time. The space usage is dominated by the precomputed values which use $O(\log N\rho) = O(n + \log t(\log nt))$ space.

For the correctness of Algorithm 1, let us first study what happens if we work with infinite precision (i.e., $\rho = \infty$). Note that $\mathtt{apxr}_\infty(z) = \omega_N^z$ since it computes

$$\prod_{q=1}^{\log N - 1} [2^q \text{ divides } z] r_q = \prod_{q=1}^{\log N - 1} [2^q \text{ divides } z] \omega_N^{2^q} = \omega_N^{\sum_{q=1}^{\log N-1} [2^q \text{ divides } z] 2^q} = \omega^z.$$

Moreover, note that on iteration ℓ of the for-loop of Line 5, we will have on Line 9 that $p = P(\omega_N^\ell)$ by the definition of $P(x)$. Then, it is easy to see that Algorithm 1 indeed evaluates the right-hand side of Eq. 2.

Now, let us focus on the finite precision. The algorithm computes an N-sized sum of $(n + 2 \log N)$-sized products of precomputed values, (increased by one). Note that it is sufficient to guarantee that on Line 9 in every iteration ℓ, $|p - \omega_N^{-\ell t} \prod_{i=1}^n (1 + \omega_N^{\ell a_i})| \le 0.4$, since then the total error of s on Line 10 is at most $0.4N$ and the total error of s/N is 0.4, which guarantees rounding to the correct integer. Recall that p is the result of an $(n + 2 \log N)$-sized product, so let us analyze how the approximation error propagates in this situation. If $\hat{a}, \hat{b}$ are approximations of a, b and we approximate c by $\mathtt{tr}_\rho \hat{a} * \hat{b}$, we have

$$|c - \hat{c}| \le |a - \hat{a}||b| + |b - \hat{b}||a| + |a - \hat{a}||b - \hat{b}| + 2^{-\rho}.$$

Thus, if a is the result of a (i-1)-sized product, and using an upper bound of 2 for the modulus of any of the product terms in the algorithm, we can upper bound the error of E_i estimating an i-length product as follows: $E_1 \le 2^{-\rho}$ and for $i > 1$:

$$E_i \le 2E_{i-1} + 2^{-\rho} 2^{i-1} + 2^{-\rho} E_{i-1} + 2^{-\rho} \le 3E_{i-1} + 2^{-\rho} 2^i.$$

Using straightforward induction we have that $E_i \le 6^i 2^{-\rho}$ So indeed, setting $\rho = 3n + 6 \log nt$ suffices.

A Generic Framework

For Theorem 1, we only used that the to be determined value is a coefficient of a (relatively)

small degree polynomial that we can evaluate efficiently. Whether this is the case for other problems solved by dynamic programming can be seen from the structure of the used recurrence: when the recurrence can be formulated over a polynomial ring where the polynomials have small degree, we can evaluate it fast and interpolate with the same technique as above to find a required coefficient. For example, for Knapsack, one can use the polynomial $P(x, y) = \prod_{i=1}^{n}(x^{a_i} y^{b_i})$ and look for a nonzero coefficient of $x^{t'} y^{u'}$ where $t' \leq t$ and $u' \geq u$ to obtain a pseudo-polynomial time and polynomial space algorithm as well.

Naturally, this technique does not only apply to the polynomial ring. In general, if the ring would be $R \subseteq \mathcal{C}^{N \times N}$ equipped with matrix addition and multiplication, we just need a matrix that simultaneously diagonalizes all matrices of R (in the above case, R are all circulant matrices which are simultaneously diagonalized by the Fourier matrix).

Applications

The framework applies to many dynamic programming algorithms. A nice additional example is the algorithm of Dreyfus and Wagner for Steiner tree [4, 7, 8].

Cross-References

▸ Knapsack

Recommended Reading

1. Brent R, Zimmermann P (2010) Modern computer arithmetic. Cambridge University Press, New York
2. Chudnovsky DV, Chudnovsky GV (1997) Approximations and complex multiplication according to ramanujan. In: Pi: a source book. Springer, New York, pp 596–622
3. Dasgupta S, Papadimitriou CH, Vazirani U (2006) Algorithms. McGraw-Hill, Boston
4. Dreyfus SE, Wagner RA (1972) The Steiner problem in graphs. Networks 1:195–207
5. Fürer M (2009) Faster integer multiplication. SIAM J Comput 39(3):979–1005
6. Knuth DE (1997) The art of computer programming, vol 2 (3rd edn.): seminumerical algorithms. Addison-Wesley Longman, Boston
7. Lokshtanov D, Nederlof J (2010) Saving space by algebraization. In: Proceedings of the forty-second ACM symposium on theory of computing, STOC '10, Cambridge. ACM, New York, pp 321–330
8. Nederlof J (2013) Fast polynomial-space algorithms using inclusion-exclusion. Algorithmica 65(4):868–884

Exact Algorithms for Bandwidth

Marek Cygan
Institute of Informatics, University of Warsaw, Warsaw, Poland

Keywords

Bandwidth; Exponential time algorithms; Graph ordering

Years and Authors of Summarized Original Work

2000; Feige
2008; Cygan, Pilipczuk
2010; Cygan, Pilipczuk
2012; Cygan, Pilipczuk

Problem Definition

Given a graph G with n vertices, an *ordering* is a bijective function $\pi : V(G) \to \{1, 2, \ldots, n\}$. The bandwidth of π is a maximal length of an edge, i.e.,

$$\mathrm{bw}(\pi) = \max_{uv \in E(G)} |\pi(u) - \pi(v)|.$$

The bandwidth problem, given a graph G and a positive integer b, asks if there exists an ordering of bandwidth at most b.

Key Results

An exhaustive search for the bandwidth problem enumerates all the $n!$ orderings, trying to find one of bandwidth at most b. The first single exponential time algorithm is due to Feige and Kilian [6], which we are going to describe now.

Bucketing

Definition 1 For a positive integer k, let $\mathcal{I}_k$ be the collection of $\lceil n/k \rceil$ sets obtained by splitting the set $\{1, \ldots, n\}$ into equal parts (except the last one), i.e., $\mathcal{I}_k = \{\{1, \ldots, k\}, \{k+1, \ldots, 2k\}, \ldots\}$. A function $f : V(G) \to \mathcal{I}_k$ is called a k-bucket assignment, if for every edge $uv \in E(G)$ at least one of the following conditions is satisfied:

- $f(u) = f(v)$,
- $|\max f(u) - \min f(v)| \leq b$,
- $|\min f(u) - \max f(v)| \leq b$.

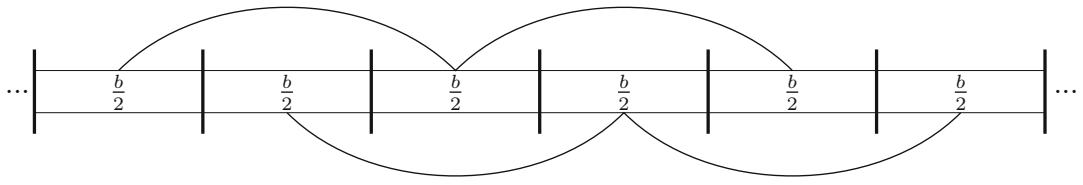
Clearly, if a function $f : V(G) \to \mathcal{I}_k$ is not a k-bucket assignment, then there is no ordering π of bandwidth at most b consistent with f, where π is *consistent* with f iff $\pi(v) \in f(v)$ for each $v \in V(G)$. A bucket function can be seen as a rough assignment – instead of assigning vertices to their final positions in the ordering, we assign them to intervals.

The $\mathcal{O}(10^n \text{poly}(n))$ time algorithm of [6] is based on two ideas, both related to the notion of bucket assignments. For the sake of presentation, let us assume that n is divisible by b, whereas b is a power of two. Moreover, we assume that G is connected, as otherwise it is enough to consider each connected component of G separately.

First, one needs to show that there is a family of at most $n3^{n-1}$ b-bucket assignments $\mathcal{F}$, such that any ordering of bandwidth at most b is consistent with some b-bucket assignment from $\mathcal{F}$. We create $\mathcal{F}$ recursively by branching. First, fix an arbitrary vertex v_0, and assign it to some interval from $\mathcal{I}_k$ (there are at most n choices here). Next, consider any vertex v without assigned interval, which has a neighbor u with already assigned interval. By the assumption that G is connected, v always exists. Note that in order to create a valid bucket assignment, v has to be assigned either to the same interval as u or to one of its two neighboring intervals. This gives at most three branches to be explored.

In the second phase, consider some b-bucket assignment $f \in \mathcal{F}$. We want to check whether there exists some ordering of bandwidth at most b consistent with f. To do this, for each vertex v, we branch into two choices, deciding whether v should be assigned to the left half of $f(v)$ or to the right half of $f(v)$. This leads to at most 2^n $b/2$-bucket assignments to be processed. The key observation is that each of those assignments can be naturally split into two independent subproblems. This is because each edge within an interval of length $b/2$ and each edge between two neighboring intervals of length $b/2$ will be of length at most $b-1$. Additionally, each edge connecting two vertices with at least two intervals of length $b/2$ in between would lead to violating the constraint of being a valid $b/2$-bucket assignment. Therefore, it is enough to consider vertices in even and odd intervals separately (see Fig. 1). Such routine of creating more and more refined bucket assignments can be continued, where the running time used for n vertices satisfies

$$T(n) = 2^n \cdot 2 \cdot T(\frac{n}{2})$$



Exact Algorithms for Bandwidth, Fig. 1 *Thick vertical lines* separate subsequent intervals from $\mathcal{I}_{b/2}$. Meaningful edges connect vertices with exactly one interval in between

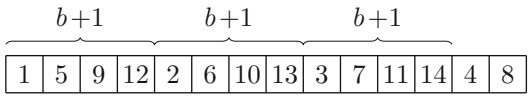
which in turn gives $T(n) = 4^n \text{poly}(n)$. Since we have $|\mathcal{F}| \leq n3^{n-1}$, we end up with $\mathcal{O}(12^n \text{poly}(n))$ time algorithm. If instead of generating b-bucket assignments one uses $b/2$-bucket assignments (there are at most $n5^{n-1}$ of them), then the running time can be improved to $10^n \text{poly}(n)$.

Dynamic Programming

In [2, 5], Cygan and Pilipczuk have shown that for a single $(b+1)$-bucket assignment, one can check in time and space $\mathcal{O}(2^n \text{poly}(n))$ whether there exists an ordering of bandwidth at most b consistent with it. Since there are at most $n3^{n-1}$ $(b+1)$-bucket assignments, this leads to $\mathcal{O}(6^n \text{poly}(n))$ time algorithm.

The key idea is to assign vertices to their final positions consistent with some $f \in \mathcal{F}$ in a very specific order. Let us color the set of positions $\{1, \ldots, n\}$ with $\text{color}(i) = (i-1) \bmod (b+1)$. Define a *color order* of positions, where positions from $\{1, \ldots, n\}$ are sorted by their color values, breaking ties with position values (see Fig. 2).

A lemma that proves usefulness of the color order shows that if we assign vertices to positions in the color order, then we can use the standard Held-Karp dynamic programming over subsets approach. In particular, in a state of dynamic programming, it is enough to store the subset $S \subseteq V(G)$ of vertices already assigned to the first $|S|$ positions in the color order (see Fig. 3).



Exact Algorithms for Bandwidth, Fig. 2 An index of each position in a color order for $n = 14$ and $b = 3$

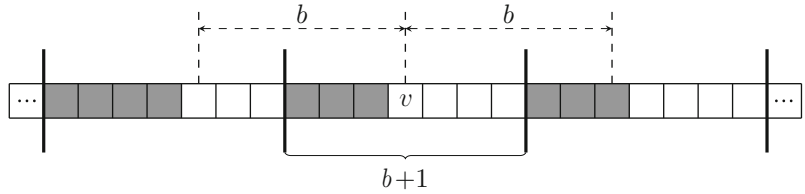
Further Improvements

Instead of upper bounding running time of the algorithm for each $(b+1)$-bucket assignment separately, one can count the number of states of the dynamic programming routine used by the algorithm throughout the processing of all the bucket assignments. As shown in [2], this leads to $\mathcal{O}(5^n \text{poly}(n))$ running time, which with more insights and more technical analysis can be further improved to $O(4.83^n)$ [5] and $O(4.383^n)$ [3]. If only polynomial space is allowed, then the best known algorithm needs $O(9.363^n)$ running time [4].

Related Work

Concerning small values of b, Saxe [8] presented a nontrivial $O(n^{b+1})$ time and space dynamic programming, consequently proving the problem to be in XP. However, Bodlaender et al. [1] have shown that bandwidth is hard for any fixed level of the W hierarchy.

For a related problem of minimum distortion embedding, Fomin et al. [7] obtained a $\mathcal{O}(5^n \text{poly}(n))$ time algorithm, improved by Cygan and Pilipczuk [4] to running times same as for the best known bandwidth algorithms.



Exact Algorithms for Bandwidth, Fig. 3 When a vertex v is to be assigned to the next position in the color order, then all its neighbors from the left interval cannot be yet assigned a position, whereas all its neighbors from the right interval have to be already assigned in order to obtain an ordering of bandwidth at most b

Open Problems

Many vertex ordering problems admit $\mathcal{O}(2^n \text{poly}(n))$ time and space algorithms, like Hamiltonicity, cutwidth, pathwidth, optimal linear arrangement, etc. In [2], Cygan and Pilipczuk have shown that a dynamic programming routine with such a running time is possible, provided a $(b+1)$-bucket assignment is given. A natural question to ask is whether it is possible to obtain $\mathcal{O}(2^n \text{poly}(n))$ without the assumption of having a fixed assignment to be extended.

Cross-References

▸ Graph Bandwidth

Recommended Reading

1. Bodlaender HL, Fellows MR, Hallett MT (1994) Beyond np-completeness for problems of bounded width: hardness for the W hierarchy. In: Leighton FT, Goodrich MT (eds) Proceedings of the twenty-sixth annual ACM symposium on theory of computing, Montréal, 23–25 May 1994. ACM, pp 449–458, doi:10.1145/195058.195229. http://doi.acm.org/10.1145/195058.195229
2. Cygan M, Pilipczuk M (2008) Faster exact bandwidth. In: Broersma H, Erlebach T, Friedetzky T, Paulusma D (eds) Graph-theoretic concepts in computer science, 34th international workshop, WG 2008, Durham, June 30–July 2, 2008. Revised papers, Lecture notes in computer science, vol 5344, pp 101–109. doi:10.1007/978-3-540-92248-3_10. http://dx.doi.org/10.1007/978-3-540-92248-3_10
3. Cygan M, Pilipczuk M (2010) Exact and approximate bandwidth. Theor Comput Sci 411(40–42):3701–3713. doi:10.1016/j.tcs.2010.06.018. http://dx.doi.org/10.1016/j.tcs.2010.06.018
4. Cygan M, Pilipczuk M (2012) Bandwidth and distortion revisited. Discret Appl Math 160(4–5):494–504. doi:10.1016/j.dam.2011.10.032. http://dx.doi.org/10.1016/j.dam.2011.10.032
5. Cygan M, Pilipczuk M (2012) Even faster exact bandwidth. ACM Trans Algorithms 8(1):8. doi:10.1145/2071379.2071387. http://doi.acm.org/10.1145/2071379.2071387
6. Feige U (2000) Coping with the np-hardness of the graph bandwidth problem. In: Halldórsson MM (ed) Proceedings of the 7th Scandinavian workshop on algorithm theory (SWAT), Bergen. Lecture notes in computer science, vol 1851. Springer, pp 10–19
7. Fomin FV, Lokshtanov D, Saurabh S (2011) An exact algorithm for minimum distortion embedding. Theor Comput Sci 412(29):3530–3536. doi:10.1016/j.tcs.2011.02.043. http://dx.doi.org/10.1016/j.tcs.2011.02.043
8. Saxe J (1980) Dynamic programming algorithms for recognizing small-bandwidth graphs in polynomial time. SIAM J Algebr Methods 1:363–369

Exact Algorithms for Dominating Set

Dieter Kratsch
UFM MIM – LITA, Université de Lorraine, Metz, France

Keywords

Branch and reduce; Dominating set; Exact exponential algorithm; Measure and conquer

Years and Authors of Summarized Original Work

2005; Fomin, Grandoni, Kratsch
2008; van Rooij, Bodlaender
2011; Iwata

Problem Definition

The dominating set problem is a classical NP-hard optimization problem which fits into the broader class of covering problems. Hundreds of papers have been written on this problem that has a natural motivation in facility location.

Definition 1 For a given undirected, simple graph $G = (V, E)$, a subset of vertices $D \subseteq V$ is called a *dominating set* if every vertex $u \in V - D$ has a neighbor in D. The minimum dominating set problem (abbr. MDS) is to find a *minimum dominating set* of G, i.e., a dominating set of G of minimum cardinality.

Problem 1 (MDS)

INPUT: Undirected simple graph $G = (V, E)$.
OUTPUT: A minimum dominating set D of G.

Various modifications of the dominating set problem are of interest, some of them obtained by putting additional constraints on the dominating set as, e.g., requesting it to be an independent set or to be connected. In graph theory, there is a huge literature on domination dealing with the problem and its many modifications. In graph algorithms, the MDS problem and some of its modifications like independent dominating set and connected dominating set have been studied as benchmark problems for attacking NP-hard problems under various algorithmic approaches.

Known Results

The algorithmic complexity of MDS and its modifications when restricted to inputs from a particular graph class has been studied extensively. Among others, it is known that MDS remains NP-hard on bipartite graphs, split graphs, planar graphs, and graphs of maximum degree 3. Polynomial time algorithms to compute a minimum dominating set are known, e.g., for permutation, interval, and k-polygon graphs. There is also a $O(3^k\, n^{O(1)})$ time algorithm to solve MDS on graphs of treewidth at most k.

The dominating set problem is one of the basic problems in parameterized complexity; it is W[2]-complete and thus it is unlikely that the problem is fixed parameter tractable. On the other hand, the problem is fixed parameter tractable on planar graphs. Concerning approximation, MDS is equivalent to MINIMUM SET COVER under L-reductions. There is an approximation algorithm solving MDS within a factor of $1 + \log|V|$, and it cannot be approximated within a factor of $(1-\epsilon)\ln|V|$ for any $\epsilon > 0$, unless NP $\subset$ DTIME($n^{\log\log n}$).

Exact Exponential Algorithms

If P $\neq$ NP, then no polynomial time algorithm can solve MDS. Even worse, it has been observed in [5] that unless SNP$\subseteq$ SUBEXP (which is considered to be highly unlikely), there is not even a subexponential time algorithm solving the dominating set problem.

The trivial $O(2^n\,(n+m))$ algorithm, which simply checks all the 2^n vertex subsets whether they are dominating, clearly solves MDS. Three faster algorithms have been established in 2004. The algorithm of Fomin et al. [5] uses a deep graph-theoretic result due to B. Reed, stating that every graph on n vertices with minimum degree at least three has a dominating set of size at most $3n/8$, to establish an $O(2^{0.955n})$ time algorithm solving MDS. The $O(2^{0.919n})$ time algorithm of Randerath and Schiermeyer [9] uses very nice ideas including matching techniques to restrict the search space. Finally, Grandoni [6] established an $O(2^{0.850n})$ time algorithm to solve MDS.

Key Results

Branch and Reduce and Measure and Conquer

The work of Fomin, Grandoni, and Kratsch presents a simple and easy way to implement a recursive branch and reduce algorithm to solve MDS. It was first presented at ICALP 2005 [2] and later published in 2009 in [3]. The running time of the algorithm is significantly faster than the ones stated for the previous algorithms. This is heavily based on the analysis of the running time by measure and conquer, which is a method to analyze the worst case running time of (simple) branch and reduce algorithms based on a sophisticated choice of the measure of a problem instance.

Theorem 1 *There is a branch and reduce algorithm solving* MDS *in time* $O(2^{0.610n})$ *using polynomial space.*

Theorem 2 *There is an algorithm solving* MDS *in time* $O(2^{0.598n})$ *using exponential space.*

The algorithms of Theorems 1 and 2 are simple consequences of a transformation from MDS to MINIMUM SET COVER (abbr. MSC) combined with new exact exponential time algorithms for MSC.

Problem 2 (MSC)

INPUT: Finite set $\mathcal{U}$ and a collection $\mathcal{S}$ of subsets $S_1, S_2, \ldots S_t$ of $\mathcal{U}$.

OUTPUT: A minimum set cover $\mathcal{S}'$, where $\mathcal{S}' \subseteq \mathcal{S}$ is a set cover of $(\mathcal{U}, \mathcal{S})$ if $\bigcup_{S_i \in \mathcal{S}'} S_i = \mathcal{U}$.

Theorem 3 *There is a branch and reduce algorithm solving* MSC *in time* $O(2^{0.305(|\mathcal{U}|+|\mathcal{S}|)})$ *using polynomial space.*

Applying memorization to the polynomial space algorithm of Theorem 3, the running time can be improved as follows.

Theorem 4 *There is an algorithm solving* MSC *in time* $O(2^{0.299(|\mathcal{S}|+|\mathcal{U}|)})$ *needing exponential space.*

The analysis of the worst case running time of the simple branch and reduce algorithm solving MSC (of Theorem 3) is done by a careful choice of the measure of a problem instance which allows to obtain an upper bound that is significantly smaller than the one that could be obtained using the standard measure. The refined analysis leads to a collection of recurrences. Then, random local search was used to compute the weights, used in the definition of the measure, aiming at the best achievable upper bound of the worst case running time. By now various other methods to do these time-consuming computations are available; see, e.g., [1].

Getting Faster MDS Algorithms

There is a lot of interest in exact exponential algorithms for solving MDS and in improving their best known running times. Two important improvements on the running times of the original algorithm stated in Theorems 1 and 2 have been achieved. To simplify the comparison, let us mention that in [4] those running times are stated as $O(1.5259^n)$ using polynomial space and $O(1.5132^n)$ needing exponential space.

Van Rooij and Bodlaender presented faster exact exponential algorithms solving MDS that are strongly based on the algorithms of Fomin et al. and the methods of their analysis. By introducing new reduction rules in the algorithm and a refined analysis, they achieved running time $O(1.5134^n)$ using polynomial space and time $O(1.5063^n)$, presented at STACS 2008. This analysis has been further improved in [11] to achieve a running time of $O(1.4969^n)$ using polynomial space, which was published in 2011. It should be emphasized that memorization cannot be applied to the latter algorithm.

The currently best known algorithms solving MDS have been obtained by Ywata [7] and presented at IPEC 2011.

Theorem 5 *There is a branch and reduce algorithm solving* MDS *in time* $O(1.4864^n)$ *using polynomial space.*

Theorem 6 *There is an algorithm solving* MDS *in time* $O(1.4689^n)$ *needing exponential space.*

Ywata's polynomial space branch and reduce algorithm is also strongly related to the algorithm of Fomin et al. and its analysis. The improvement in the running time is achieved by some crucial change in the order of branchings in the algorithm solving MSC, i.e., the algorithm branches on the same element consecutively. These consecutive branchings can then be exploited by a refined analysis using global weights called potentials. Thus, such an analysis is dubbed "potential method." By a variant of memorization where dynamic programming memorizes only solutions of subproblems with small number of elements, an algorithm of running time $O(1.4689^n)$ needing exponential space has been obtained.

Counting Dominating Sets

A strongly related problem is #DS that asks to determine for a given graph G the number of dominating sets of size k, for any k. In [8], Nederlof, van Rooij, and van Dijk show how to combine inclusion/exclusion and a branch and reduce algorithm while using measure and conquer, as to obtain an algorithm (needing exponential space) of running time $O(1.5002^n)$. Clearly, this also solves MDS.

Applications

There are various other NP-hard domination-type problems that can be solved by exact exponential algorithms based on an algorithm solving

MINIMUM SET COVER: any instance of the initial problem is transformed to an instance of MSC, and then an algorithm solving MSC is applied and thus the initial problem is solved. Examples of such problems are TOTAL DOMINATING SET, k-DOMINATING SET, k-CENTER, and MDS on split graphs. Measure and conquer and the strongly related quasiconvex analysis of Eppstein [1] have been used to design and analyze a variety of exact exponential branch and reduce algorithms for NP-hard problems, optimization, counting, and enumeration problems; see [4].

Open Problems

While for many algorithms it is easy to show that the worst case analysis is tight, this is not the case for the nowadays time analysis of branch and reduce algorithms. For example, the worst case running times of the branch and reduce algorithms of Fomin et al. [3] solving MDS and MSC remain unknown; a lower bound of $\Omega(3^{n/4})$ for the MDS algorithm is known. The situation is similar for many other branch and reduce algorithms. Consequently, there is a strong need for new and better tools to analyze the worst case running time of branch and reduce algorithms.

Cross-References

- Connected Dominating Set
- Exact Algorithms for Induced Subgraph Problems
- Exact Algorithms for Maximum Independent Set
- Minimal Dominating Set Enumeration

Recommended Reading

1. Eppstein D (2006) Quasiconvex analysis of backtracking algorithms. ACM Trans Algorithms 2(4):492–509
2. Fomin FV, Grandoni F, Kratsch D (2005) Measure and conquer: domination – a case study. In: Proceedings of ICALP 2005, Lisbon
3. Fomin FV, Grandoni F, Kratsch D (2009) A measure & conquer approach for the analysis of exact algorithms. J ACM 56(5)
4. Fomin FV, Kratsch D (2010) Exact exponential algorithms. Springer, Heidelberg
5. Fomin FV, Kratsch D, Woeginger GJ (2004) Exact (exponential) algorithms for the dominating set problem. In: Proceedings of WG 2004, Bonn. LNCS, vol 3353. Springer, pp 245–256
6. Grandoni F (2004) Exact algorithms for hard graph problems. PhD thesis, Università di Roma "Tor Vergata", Roma, Mar 2004
7. Iwata Y (2011) A faster algorithm for Dominating Set analyzed by the potential method. In: Proceedings of IPEC 2011, Saarbrücken. LNCS, vol 7112. Springer, pp 41–54
8. Nederlof J, van Rooij JMM, van Dijk TC (2014) Inclusion/exclusion meets measure and conquer. Algorithmica 69(3):685–740
9. Randerath B, Schiermeyer I (2004) Exact algorithms for MINIMUM DOMINATING SET. Technical Report, zaik-469, Zentrum für Angewandte Informatik Köln, Apr 2004
10. van Rooij JMM (2011) Exact exponential-time algorithms for domination problems in graphs. PhD thesis, University Utrecht
11. van Rooij JMM, Bodlaender HL (2011) Exact algorithms for dominating set. Discret Appl Math 159(17):2147–2164
12. Woeginger GJ (2003) Exact algorithms for NP-hard problems: a survey. Combinatorial optimization – Eureka, you shrink. LNCS, vol 2570. Springer, Berlin/Heidelberg, pp 185–207

Exact Algorithms for General CNF SAT

Edward A. Hirsch
Laboratory of Mathematical Logic, Steklov Institute of Mathematics, St. Petersburg, Russia

Keywords

Boolean satisfiability; Exponential-time algorithms; SAT

Years and Authors of Summarized Original Work

1998; Hirsch
2003; Schuler

Problem Definition

The satisfiability problem (*SAT*) for Boolean formulas in conjunctive normal form (*CNF*) is one of the first **NP**-complete problems [2, 13]. Since its **NP**-completeness currently leaves no hope for polynomial-time algorithms, the progress goes by decreasing the exponent. There are several versions of this parametrized problem that differ in the parameter used for the estimation of the running time.

Problem 1 (SAT)

INPUT: Formula F in CNF containing n variables, m clauses, and l literals in total.
OUTPUT: "Yes" if F has a *satisfying assignment*, i.e., a substitution of Boolean values for the variables that makes F true. "No" otherwise.

The bounds on the running time of SAT algorithms can be thus given in the form $|F|^{O(1)} \cdot \alpha^n$, $|F|^{O(1)} \cdot \beta^m$, or $|F|^{O(1)} \cdot \gamma^l$, where $|F|$ is the length of a reasonable bit representation of F (i.e., the formal input to the algorithm). In fact, for the present algorithms, the bases β and γ are constants, while α is a function $\alpha(n, m)$ of the formula parameters (because no better constant than $\alpha = 2$ is known).

Notation

A formula in conjunctive normal form is a set of clauses (understood as the conjunction of these clauses), a clause is a set of literals (understood as the disjunction of these literals), and a literal is either a Boolean variable or the negation of a Boolean variable. A truth assignment assigns Boolean values (*false* or *true*) to one or more variables. An assignment is abbreviated as the list of literals that are made true under this assignment (e.g., assigning *false* to x and *true* to y is denoted by $\neg x, y$). The result of the application of an assignment A to a formula F (denoted $F[A]$) is the formula obtained by removing the clauses containing the true literals from F and removing the falsified literals from the remaining clauses. For example, if $F = (x \vee \neg y \vee z) \wedge (y \vee \neg z)$, then $F[\neg x, y] = (z)$. A *satisfying* assignment for F is an assignment A such that $F[A] =$ true. If such an assignment exists, F is called *satisfiable*.

Key Results

Bounds for β and γ

General Approach and a Bound for β

The trivial brute-force algorithm enumerating all possible assignments to the n variables runs in 2^n polynomial-time steps. Thus $\alpha \leq 2$, and by trivial reasons also $\beta, \gamma \leq 2$. In the early 1980s, Monien and Speckenmeyer noticed that β could be made smaller. (They and other researchers also noticed that α could be made smaller for a special case of the problem where the length of each clause is bounded by a constant; the reader is referred to another entry (*Local search algorithms for k-SAT*) of the *Encyclopedia* for relevant references and algorithms.) Then Kullmann and Luckhardt [12] set up a framework for divide-and-conquer (Also called *DPLL* due to the papers of Davis and Putnam [6] and Davis, Logemann, and Loveland [7].) algorithms for SAT that split the original problem into several (yet usually a constant number of) subproblems by substituting the values of some variables and simplifying the obtained formulas. This line of research resulted in the following upper bounds for β and γ:

Theorem 1 (Hirsch [8]) *SAT can be solved in time*

1. $|F|^{O(1)} \cdot 2^{0.30897m}$;
2. $|F|^{O(1)} \cdot 2^{0.10299l}$.

A typical divide-and-conquer algorithm for SAT consists of two phases: splitting of the original problem into several subproblems (e.g., reducing SAT(F) to SAT(F[x]) and SAT(F[¬x])) and simplification of the obtained subproblems using polynomial-time transformation rules that do not affect the satisfiability of the subproblems (i.e., they replace a formula by an equisatisfiable one). The subproblems $F_1, \ldots, F_k$ for splitting are chosen so that the corresponding recurrent inequality using the simplified problems

$F'_1, \ldots, F'_k$,

$$T(F) \leq \sum_{i=1}^{k} T\left(F'_i\right) + const,$$

gives a desired upper bound on the number of leaves in the recurrence tree and, hence, on the running time of the algorithm. In particular, in order to obtain the bound $|F|^{O(1)} \cdot 2^{0.30897m}$ *one takes either two subproblems* $F[x], F[\neg x]$ *with recurrent inequality*

$$t_m \leq t_{m-3} + t_{m-4}$$

or four subproblems $F[x, y], F[x, \neg y], F[\neg x, y], F[\neg x, \neg y]$ *with recurrent inequality*

$$t_m \leq 2t_{m-6} + 2t_{m-7}$$

where $t_i = max_{m(G)\leq i} T(G)$. *The simplification rules used in the* $|F|^{O(1)} \cdot 2^{0.30897m}$*-time and the* $|F|^{O(1)} \cdot 2^{0.10299l}$*-time algorithms are as follows*:

Simplification Rules

Elimination of 1-Clauses If F contains a 1-clause (a), replace F by $F[a]$.

Subsumption If F contains two clauses C and D such that $C \subseteq D$, replace F by $F \backslash \{D\}$.

Resolution with Subsumption Suppose a literal a and clauses C and D are such that a is the only literal satisfying both conditions $a \in C$ and $\neg a \in D$. In this case, the clause $(C \cup D) \backslash \{a, \neg a\}$ is called the *resolvent by the literal* a of the clauses C and D and denoted by $R(C, D)$.

The rule is: if $R(C, D) \subseteq D$, replace F by $(F \backslash \{D\}) \cup \{R(C, D)\}$.

Elimination of a Variable by Resolution [6] Given a literal a, construct the formula $\mathrm{DP}_a(F)$ by

1. Adding to F all resolvents by a
2. Removing from F all clauses containing a or $\neg a$

The rule is: if $\mathrm{DP}_a(F)$ is not larger in m (resp., in l) than F, then replace F by $\mathrm{DP}_a(F)$.

Elimination of Blocked Clauses A clause C is *blocked* for a literal a w.r.t. F if C contains the literal a, and the literal $\neg a$ occurs only in the clauses of F that contain the negation of at least one of the literals occurring in $C \backslash \{a\}$. For a CNF-formula F and a literal a occurring in it, the assignment $I(a, F)$ is defined as

$$\{a\} \cup \{\text{literals } x \notin \{a, \neg a\} \mid \text{the clause } \{\neg a, x\} \text{ is blocked for } \neg a \text{ w.r.t. } F\}.$$

Lemma 2 (Kullmann [11])

(1) If a clause C *is blocked for a literal* a *w.r.t.* F*, then* F *and* $F \backslash \{C\}$ *are equi-satisfiable.*
(2) Given a literal a*, the formula* F *is satisfiable iff at least one of the formulas* $F[\neg a]$ *and* $F[I(a, F)]$ *is satisfiable.*

The first claim of the lemma is employed as a simplification rule.

Application of the Black and White Literals Principle Let P be a binary relation between literals and formulas in CNF such that for a variable v and a formula F, at most one of $P(v, F)$ and $P(\neg v, F)$ holds.

Lemma 3 *Suppose that each clause of* F *that contains a literal* w *satisfying* $P(w, F)$ *contains also at least one literal* b *satisfying* $P(\neg b, F)$*. Then* F *and* $F[\{l \mid P(\neg l, F)\}]$ *are equi-satisfiable.*

A Bound for γ

To obtain the bound $|F|^{O(1)} \cdot 2^{0.10299l}$, it is enough to use a pair $F[\neg a], F[I(a, F)]$ of subproblems (see Lemma 2(2)) achieving the desired recurrent inequality $t_l \leq t_{l-5} + t_{l-17}$ and to switch to the $|F|^{O(1)} \cdot 2^{0.30897m}$-time algorithm if there are none. A recent (much more technically involved) improvement to this algorithm [16] achieves the bound $|F|^{O(1)} \cdot 2^{0.0926l}$.

A Bound for α

Currently, no non-trivial constant upper bound for α is known. However, starting with [14] there was an interest to non-constant bounds. A series of randomized and deterministic algorithms showing successive improvements was developed, and at the moment the best possible bound is achieved by a deterministic divide-and-conquer algorithm employing the following recursive procedure. The idea behind it is a dichotomy: either each clause of the input formula can be shortened to its first k literals (then a k-CNF algorithm can be applied), or all these literals in one of the clauses can be assumed false. (This clause-shortening approach can be attributed to Schuler [15], who used it in a randomized fashion. The following version of the deterministic algorithm achieving the best known bound both for deterministic and randomized algorithms appears in [5].)

Procedure S

Input: a CNF formula F and a positive integer k.

1. Assume F consists of clauses $C_1, \ldots, C_m$. Change each clause C_i to a clause D_i as follows: If $|C_i| > k$ then choose any k literals in C_i and drop the other literals; otherwise leave C_i as is, i.e., $D_i = C_i$. Let F' denote the resulting formula.
2. Test satisfiability of F' using the $m \cdot \text{poly}(n) \cdot (2 - 2/(k+1))^n$-time k-CNF algorithm defined in [4].
3. If F' is satisfiable, output "satisfiable" and halt. Otherwise, for each i, do the following:
   1. Convert F to F_i as follows:
      1. Replace C_j by D_j for all $j < i$.
      2. Assign *false* to all literals in D_i.
   2. Recursively invoke *Procedure S* on (F_i, k).
4. Return "unsatisfiable".

The algorithm just invokes *Procedure S* on the original formula and the integer parameter $k = k*(m,n)$. The most accurate analysis of this family of algorithms by Calabro, Impagliazzo, and Paturi [1] implies that, assuming that $m > n$, one can obtain the following bound by taking $k(m,n) = 2\log(m/n) + \text{const}$. (This explicit bound is not stated in [1] and is inferred in [3].)

Theorem 4 (Dantsin, Hirsch [3]) *Assuming $m > n$, SAT can be solved in time*

$$|F|^{O(1)} \cdot 2^n \left(1 - \frac{1}{O(\log(m/n))}\right).$$

Applications

While SAT has numerous applications, the presented algorithms have no direct effect on them.

Open Problems

Proving a constant upper bound on $\alpha < 2$ remains a major open problem in the field, as well as the hypothetic existence of $(1+\varepsilon)^l$-time algorithms for arbitrary small $\varepsilon > 0$.

It is possible to perform the analysis of a divide-and-conquer algorithm and even to generate simplification rules automatically [10]. However, this approach so far led to new bounds only for the (NP-complete) optimization version of 2-SAT [9].

Experimental Results

Jun Wang has implemented the algorithm yielding the bound on β and collected some statistics regarding the number of applications of the simplification rules [17].

Cross-References

▸ Exact Algorithms for k SAT Based on Local Search

Recommended Reading

1. Calabro C, Impagliazzo R, Paturi R (2006) A duality between clause width and clause density for SAT. In: Proceedings of the 21st annual IEEE conference on computational complexity (CCC 2006), Prague. IEEE Computer Society, pp 252–260
2. Cook SA (2006) The complexity of theorem proving procedures. In: Proceedings of the third annual ACM symposium on theory of computing, Shaker Heights, May 1971. ACM, pp 151–158
3. Dantsin E, Hirsch EA (2008) Worst-case upper bounds. In: Biere A, van Maaren H, Walsh T (eds) Handbook of satisfiability. IOS. In: Biere A, Heule MJH, van Maaren H, Walsh T (eds) Handbook of satisfiability. Frontiers in artificial intelligence and applications, vol 185. IOS Press, Amsterdam/Washington, DC, p 980. ISBN:978-1-58603-929-5, ISSN:0922-6389
4. Dantsin E, Goerdt A, Hirsch EA, Kannan R, Kleinberg J, Papadimitriou C, Raghavan P, Schöning U (2002) A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search. Theor Comput Sci 289(1):69–83
5. Dantsin E, Hirsch EA, Wolpert A (2006) Clause shortening combined with pruning yields a new upper bound for deterministic SAT algorithms. In: Proceedings of CIAC-2006, Rome. Lecture notes in computer science, vol 3998. Springer, Berlin, pp 60–68
6. Davis M, Putnam H (1960) A computing procedure for quantification theory. J ACM 7: 201–215
7. Davis M, Logemann G, Loveland D (1962) A machine program for theorem-proving. Commun ACM 5:394–397
8. Hirsch EA (2000) New worst-case upper bounds for SAT. J Autom Reason 24(4):397–420
9. Kojevnikov A, Kulikov A (2006) A new approach to proving upper bounds for MAX-2-SAT. In: Proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithms (SODA 2006), Miami. ACM/SIAM, pp 11–17
10. Kulikov A (2005) Automated generation of simplification rules for SAT and MAXSAT. In: Proceedings of the eighth international conference on theory and applications of satisfiability testing (SAT 2005), St. Andrews. Lecture notes in computer science, vol 3569. Springer, Berlin, pp 430–436
11. Kullmann O (1999) New methods for 3-{SAT} decision and worst-case analysis. Theor Comput Sci 223(1–2):1–72
12. Kullmann O, Luckhardt H (1998) Algorithms for SAT/TAUT decision based on various measures, 71pp. Preprint, http://cs-svr1.swan.ac.uk/csoliver/papers.html
13. Levin LA (1973) Universal search problems. Probl Inf Trans 9(3):265–266. In Russian. English translation Trakhtenbrot BA (1984) A survey of russian approaches to perebor (Brute-force Search) algorithms. Ann Hist Comput 6(4):384–400
14. Pudlák P (1998) Satisfiability – algorithms and logic. In: Proceedings of the 23rd international symposium on mathematical foundations of computer science, MFCS'98. Lecture notes in computer science, Brno, vol 1450. Springer, Berlin, pp 129–141
15. Schuler R (2005) An algorithm for the satisfiability problem of formulas in conjunctive normal form. J Algorithms **54**(1), 40–44
16. Wahlström M (2005) An algorithm for the SAT problem for formulae of linear length. In: Proceedings of the 13th annual European symposium on algorithms, ESA 2005. Lecture notes in computer science, Mallorca, vol 3669. Springer, Berlin, pp 107–118
17. Wang J (2002) Generating and solving 3-SAT. MSc thesis, Rochester Institute of Technology, Rochester

Exact Algorithms for Induced Subgraph Problems

Michał Pilipczuk
Institute of Informatics, University of Warsaw, Warsaw, Poland
Institute of Informatics, University of Bergen, Bergen, Norway

Keywords

Exact algorithms; Hereditary property; Induced subgraph; 2^n barrier

Years and Authors of Summarized Original Work

2010; Fomin, Villanger
2013; Bliznets, Fomin, Pilipczuk, Villanger

Problem Definition

A graph class Π is a set of simple graphs. One can also think of Π as a *property*: Π comprises all the graphs that satisfy a certain condition. We say that class (property) Π is *hereditary* if it is closed under taking induced subgraphs. More precisely, whenever $G \in \Pi$ and H is an induced subgraph of G, then also $H \in \Pi$.

We shall consider the MAXIMUM INDUCED Π-SUBGRAPH problem: given a graph G, find the largest (in terms of the number of vertices) induced subgraph of G that belongs to Π. Suppose now that class Π is polynomial-time recognizable: there exists an algorithm that decides whether a given graph H belongs to Π in polynomial time. Then MAXIMUM INDUCED Π-SUBGRAPH on an n-vertex graph G can be solved by brute force in time (The $\mathcal{O}^\star(\cdot)$ notation hides factors polynomial in the input size.) $\mathcal{O}^\star(2^n)$: we iterate through all the induced subgraphs of G, and on each of them, we run a polynomial-time test deciding whether it belongs to Π.

Can we do anything smarter? Of course, this very much depends on the class Π we are working with. MAXIMUM INDUCED Π-SUBGRAPH is a generic problem that encompasses many other problems as special cases; examples include CLIQUE (Π = complete graphs), INDEPENDENT SET (Π = edgeless graphs), or FEEDBACK VERTEX SET (Π = forests). It is convenient to assume that Π is also hereditary; this assumption is satisfied in many important examples, including the aforementioned special cases.

So far, the MAXIMUM INDUCED Π-SUBGRAPH problem has been studied for many graph classes Π, and basically in all the cases it turned out that it is possible to find an algorithm with running time $\mathcal{O}(c^n)$ for some $c < 2$. Obtaining a result of this type is often informally called *breaking the* 2^n *barrier*. While the algorithms share a common general methodology, vital details differ depending on the structural properties of the class Π. This makes each and every algorithm of this type contrived to a particular scenario. However, it is tempting to formulate the following general conjecture.

Conjecture 1 ([1]) *For every hereditary, polynomial-time recognizable class of graphs Π, there exists a constant $c_\Pi < 2$ for which there is an algorithm solving* MAXIMUM INDUCED Π-SUBGRAPH *in time $\mathcal{O}(c_\Pi^n)$.*

On one hand, current partial progress on this conjecture consists of scattered results exploiting different properties of particular classes Π, without much hope for proving more general statements. On the other hand, finding a counterexample refuting Conjecture 1 based, e.g., on the Strong Exponential Time Hypothesis seems problematic: the input to MAXIMUM INDUCED Π-SUBGRAPH consists only of $\binom{n}{2}$ bits of information about adjacencies between the vertices, and it seems difficult to model the search space of a general k-SAT using such input under the constraint that Π has to be hereditary and polynomial-time recognizable.

It can be that Conjecture 1 is either false or very difficult to prove, and therefore, one can postulate investigating its certain subcases connected to well-studied classes of graphs. For instance, one could assume that graphs from Π have constant treewidth or that Π is a subclass of chordal or interval graphs. Another direction is to strengthen the assumption about the description of the class Π by requiring that belonging to Π can be expressed in some formalism (e.g., some variant of logic). Finally, one can investigate the algorithms for MAXIMUM INDUCED Π-SUBGRAPH where Π is not required to be hereditary; here, natural nonhereditary properties are connectivity and regularity.

Key Results

Table 1 presents a selection of results on the MAXIMUM INDUCED Π-SUBGRAPH problem. Since the algorithms are usually quite technical when it comes to details, we now present an overview of the general methodology and most important techniques. In the following, we assume that Π is hereditary and polynomial-time recognizable.

Most often, the general approach is to examine the structure of the input instance and of a fixed, unknown optimum solution. The goal is to identify as broad spectrum of situations as possible where the solution can be found by examining $\mathcal{O}((2-\varepsilon)^n)$ candidates, for some $\varepsilon > 0$. By checking the occurrence of each of these situations, we eventually narrow down our investigations to the case where we have a well-defined structure of the input instance and

Exact Algorithms for Induced Subgraph Problems, Table 1 Known results for MAXIMUM INDUCED Π-SUBGRAPH. The first part of the table presents results for problems for which breaking the 2^n barrier follows directly from branching on forbidden subgraphs. The second part contains results for which breaking the barrier requires a nontrivial insight into the structure of Π. Finally, the last part contains results for nonhereditary classes Π. Here, ε denotes a small, positive constant, and its index specifies a parameter on which the value of this constant depends

Property	Time complexity	Reference
Edgeless	$\mathcal{O}(1.2109^n)$	Robson [10]
Biclique	$\mathcal{O}(1.3642^n)$	Gaspers et al. [6]
Cluster graph	$\mathcal{O}(1.6181^n)$	Fomin et al. [3]
Bipartite	$\mathcal{O}(1.62^n)$	Raman et al. [9]
Acyclic	$\mathcal{O}(1.7347^n)$	Fomin et al. [2]
Constant treewidth	$\mathcal{O}(1.7347^n)$	Fomin et al. [2]
Planar	$\mathcal{O}(1.7347^n)$	Fomin et al. [4]
d-degenerate	$\mathcal{O}((2-\varepsilon_d)^n)$	Pilipczuk×2 [8]
Chordal	$\mathcal{O}((2-\varepsilon)^n)$	Bliznets et al. [1]
Interval	$\mathcal{O}((2-\varepsilon)^n)$	Bliznets et al. [1]
r-regular	$\mathcal{O}((2-\varepsilon_r)^n)$	Gupta et al. [7]
Matching	$\mathcal{O}(1.6957^n)$	Gupta et al. [7]

a number of assumptions about how the solution looks like. Then, hopefully, a direct algorithm can be devised.

Let us consider a very simple example of this principle, which is also a technique used in many algorithms for breaking the 2^n barrier. Suppose the input graph has n vertices and assume the optimum solution is of size larger that $(1/2+\delta)n$, for some $\delta > 0$. Then, as candidates for the optimum solution, we can consider all the vertex subsets of at least this size: there is only $(2-\varepsilon)^n$ of them, where $\varepsilon > 0$ depends on δ. Similarly, if the optimum solution has size smaller than $(1/2-\delta)n$, then we can identify this situation by iterating through all the vertex subsets of size $(1/2-\delta)n$ (whose number is again $(2-\varepsilon)^n$ for some $\varepsilon > 0$) and verifying that none of them induces a graph belonging to Π; note that here we use the assumption that Π is hereditary. In this case we can solve the problem by looking at all vertex subsets of size at most $(1/2-\delta)n$. All in all, we can solve the problem faster than $\mathcal{O}^\star(2^n)$ provided that the number of vertices in the optimum solution differs by at least δn from $n/2$, for some $\varepsilon > 0$. More precisely, for every $\delta > 0$ we will obtain a running time of the form $\mathcal{O}((2-\varepsilon)^n)$, where ε tends to 0 when δ tends to 0. Hence, we can focus only on the situation when the number of vertices in the optimum solution is very close to $n/2$.

We now give an overview of some other important techniques.

Branching on Forbidden Induced Subgraphs

Every hereditary graph class Π can be characterized by giving a minimal set of forbidden induced subgraphs $\mathcal{F}$: a graph belongs to Π if and only if it does not contain any graph from $\mathcal{F}$ as an induced subgraph, and $\mathcal{F}$ is inclusion-wise minimal with this property. For instance, the class of forests is characterized by $\mathcal{F}$ being the family of all the cycles, whereas taking $\mathcal{F}$ to be the family of all the cycles of length at least 4 gives the class of chordal graphs. For many important classes the family $\mathcal{F}$ is infinite, but there are notable examples where it is finite, like cluster, trivially perfect, or split graphs.

If Π is characterized by a finite set of forbidden subgraphs $\mathcal{F}$, then already a simple branching strategy yields an algorithm working in time $\mathcal{O}((2-\varepsilon)^n)$, for some $\varepsilon > 0$ depending on $\mathcal{F}$. Without going into details, we iteratively find a forbidden induced subgraph that is not yet removed by the previous choices and branch on the fate of all the undecided vertices in this subgraph, omitting the branch where all of them are included in the solution. Since this forbidden induced subgraph is of constant size, a standard analysis shows that the running time of this algorithm is $\mathcal{O}((2-\varepsilon)^n)$ for some $\varepsilon > 0$ depending on $\max_{H\in\mathcal{F}} |V(H)|$. This simple observation can be combined with more sophisticated techniques in case when $\mathcal{F}$ is infinite. We can namely start the algorithm by branching on forbidden induced

subgraphs that are of constant size and, when their supply is exhausted, turn to some other algorithms. The following lemma provides a formalization of this concept; a graph is called $\mathcal{F}$-*free* if it does not contain any graph from $\mathcal{F}$ as an induced subgraph.

Lemma 1 ([1]) *Let $\mathcal{F}$ be a finite set of graphs and let ℓ be the maximum number of vertices in a graph from $\mathcal{F}$. Let Π be a hereditary graph class that is polynomial-time recognizable. Assume that there exists an algorithm $\mathcal{A}$ that for a given $\mathcal{F}$-free graph G on n vertices, in time $\mathcal{O}((2-\varepsilon)^n)$ finds a maximum induced subgraph of G that belongs to Π, for some $\varepsilon > 0$. Then there exists an algorithm $\mathcal{A}'$ that for a given graph G on n vertices, in time $\mathcal{O}((2-\varepsilon')^n)$ finds a maximum induced subgraph of G that is $\mathcal{F}$-free and belongs to Π, where $\varepsilon' > 0$ is a constant depending on ε and ℓ.*

Thus, for the purpose of breaking the 2^n barrier, it is sufficient to focus on the case when no constant-size forbidden induced subgraph is present in the input graph.

Exploiting a Large Substructure

Here, the general idea is to look for a large substructure in the graph that can be leveraged to design an algorithm breaking the barrier. Let us take as an example the Maximum Induced Chordal Subgraph problem, considered by Bliznets et al. [1]. Suppose that in the input graph G one can find a clique Q of size δn, for some $\delta > 0$; recall that the largest clique in a graph can be found as fast as in time $\mathcal{O}(1.2109^n)$ [10]. Then consider the following algorithm: guess, by considering $2^{n-|Q|}$ possibilities, the intersection of the optimum solution with $V(G) \setminus Q$. Then observe that, since Q is a clique, every induced cycle in G can have only at most two vertices in common with Q. Hence, the problem of optimally extending the choice on $V(G) \setminus Q$ to Q essentially boils down to solving a Vertex Cover instance on $|Q|$ vertices, which can be done in time $\mathcal{O}(1.2109^{|Q|})$. As Q constitutes a linear fraction of all the vertices, the overall running time is $\mathcal{O}(1.2109^{|Q|} \cdot 2^{n-|Q|})$, which is $\mathcal{O}((2-\varepsilon)^n)$ for some $\varepsilon > 0$ depending on δ. Thus, one can focus on the case where the largest clique in the input graph, and hence also in any maximum-sized induced chordal subgraph, has less than δn vertices.

Potential Maximal Cliques

A *potential maximal clique* (*PMC*) in a graph G is a subset of vertices that becomes a clique in some inclusion-wise minimal triangulation (By a triangulation of a graph we mean any its chordal supergraph.) of G. Fomin and Villanger in [2] observed two facts. Firstly, whenever H is an induced subgraph of G of treewidth t, then there exists a minimal triangulation TG of G that captures H in the following sense: every clique of TG intersects $V(H)$ only at a subset of some bag of a fixed width-t tree decomposition of H. Secondly, a graph G on n vertices can have only $\mathcal{O}(1.734601^n)$ PMCs, which can be enumerated in time $\mathcal{O}(1.734601^n)$. Intuitively, this means that we can effectively search the space of treewidth-t induced subgraphs of G in time $\mathcal{O}(1.734601^n \cdot n^{\mathcal{O}(t)})$ using dynamic programming. Slightly more precisely, treewidth-t induced subgraphs of G can be assembled in a dynamic programming manner using states of the form (Ω, X), where Ω is a PMC in G and X is a subset of Ω of size at most $t+1$, corresponding to $\Omega \cap V(H)$. In this manner one can obtain an algorithm with running time $\mathcal{O}(1.734601^n \cdot n^{\mathcal{O}(t)})$ for finding the maximum induced treewidth-t subgraph, which in particular implies a $\mathcal{O}(1.734601^n)$-time algorithm for Maximum Induced Forest, equivalent to Feedback Vertex Set. Recently, Fomin et al. [5] extended this framework to encapsulate also problems where the induced subgraph H is in addition required to satisfy a property expressible in *Monadic Second-Order Logic*.

Recommended Reading

1. Bliznets I, Fomin FV, Pilipczuk M, Villanger Y (2013) Largest chordal and interval subgraphs faster than 2^n. In: Bodlaender HL, Italiano GF (eds) ESA, Sophia Antipolis. Lecture Notes in Computer Science, vol 8125. Springer, pp 193–204
2. Fomin FV, Villanger Y (2010) Finding induced subgraphs via minimal triangulations. In: Marion JY, Schwentick T (eds) STACS, Nancy. LIPIcs, vol 5. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, pp 383–394
3. Fomin FV, Gaspers S, Kratsch D, Liedloff M, Saurabh S (2010) Iterative compression and exact algorithms. Theor Comput Sci 411(7–9):1045–1053
4. Fomin FV, Todinca I, Villanger Y (2011) Exact algorithm for the maximum induced planar subgraph problem. In: Demetrescu C, Halldórsson MM (eds) ESA, Saarbrücken. Lecture notes in computer science, vol 6942. Springer, pp 287–298
5. Fomin FV, Todinca I, Villanger Y (2014) Large induced subgraphs via triangulations and CMSO. In: Chekuri C (ed) SODA, Portland. SIAM, pp 582–583
6. Gaspers S, Kratsch D, Liedloff M (2012) On independent sets and bicliques in graphs. Algorithmica 62(3–4):637–658
7. Gupta S, Raman V, Saurabh S (2012) Maximum r-regular induced subgraph problem: fast exponential algorithms and combinatorial bounds. SIAM J Discr Math 26(4):1758–1780
8. Pilipczuk M, Pilipczuk M (2012) Finding a maximum induced degenerate subgraph faster than 2^n. In: Thilikos DM, Woeginger GJ (eds) IPEC, Ljubljana. Lecture notes in computer science, vol 7535. Springer, pp 3–12
9. Raman V, Saurabh S, Sikdar S (2007) Efficient exact algorithms through enumerating maximal independent sets and other techniques. Theory Comput Syst 41(3):563–587
10. Robson JM (1986) Algorithms for maximum independent sets. J Algorithms 7(3):425–440

Exact Algorithms for *k* SAT Based on Local Search

Kazuo Iwama
Computer Engineering, Kyoto University, Sakyo, Kyoto, Japan
School of Informatics, Kyoto University, Sakyo, Kyoto, Japan

Keywords

CNF satisfiability; Exponential-time algorithm; Local search

Years and Authors of Summarized Original Work

1999; Schöning

Problem Definition

The CNF satisfiability problem is to determine, given a CNF formula F with n variables, whether or not there exists a satisfying assignment for F. If each clause of F contains at most k literals, then F is called a k-CNF formula and the problem is called k-SAT, which is one of the most fundamental NP-complete problems. The trivial algorithm is to search 2^n 0/1-assignments for the n variables. But since [6], several algorithms which run significantly faster than this $O(2^n)$ bound have been developed. As a simple exercise, consider the following straightforward algorithm for 3-SAT, which gives us an upper bound of 1.913^n: choose an arbitrary clause in F, say, $(x_1 \vee \overline{x_2} \vee x_3)$. Then generate seven new formulas by substituting to these x_1, x_2, and x_3 all the possible values except $(x_1, x_2, x_3) = (0, 1, 0)$ which obviously unsatisfies F. Now one can check the satisfiability of these seven formulas and conclude that F is satisfiable iff at least one of them is satisfiable. (Let $T(n)$ denote the time complexity of this algorithm. Then one can get the recurrence $T(n) \leq 7 \times T(n-3)$ and the above bound follows.)

Key Results

In the long history of k-SAT algorithms, the one by Schöning [11] is an important breakthrough. It is a standard local search and the algorithm itself is not new (see, e.g., [7]). Suppose that y is the current assignment (its initial value is selected uniformly at random). If y is a satisfying assignment, then the algorithm answers yes and terminates. Otherwise, there is at least one clause whose three literals are all false under y. Pick an arbitrary such clause and select one of the three

literals in it at random. Then flip (true to false and vice versa) the value of that variable, replace y with that new assignment, and then repeat the same procedure. More formally:

SCH(CNF formula F, integer I)
repeat I times
$y =$ uniformly random vector $\in \{0,1\}^n$
$z =$ **RandomWalk**(F, y);
if z satisfies F
then output(z); exit;
end
output('Unsatisfiable');
RandomWalk(CNF formula $G(x_1, x_2, \ldots, x_n)$, assignment y);
$y' = y$;
for $3n$ times
if y' satisfies G
then return y'; exit;
$C \leftarrow$ an arbitrary clause of G that is not satisfied by y';
Modify y' as follows:
select one literal of C uniformly at random and flip the assignment to this literal;
end
return y'

Schöning's analysis of this algorithm is very elegant. Let $d(a, b)$ denote the Hamming distance between two binary vectors (assignments) a and b. For simplicity, suppose that the formula F has only one satisfying assignment y^* and the current assignment y is far from y^* by Hamming distance d. Suppose also that the currently false clause C includes three variables, x_i, x_j, and x_k. Then y and y^* must differ in at least one of these three variables. This means that if the value of x_i, x_j, or x_k is flipped, then the new assignment gets closer to y^* by Hamming distance one with probability at least 1/3. Also, the new assignment gets farther by Hamming distance one with probability at most 2/3. The argument can be generalized to the case that F has multiple satisfying assignments. Now here comes the key lemma:

Lemma 1 *Let F be a satisfiable formula and y^* be a satisfying assignment for F. For each assignment y, the probability that a satisfying assignment (that may be different from y^*) is found by* ***RandomWalk*** *(F, y) is at least $(1/(k-1))^{d(y,y^*)}/p(n)$, where $p(n)$ is a polynomial in n.*

By taking the average over random initial assignments, the following theorem follows:

Theorem 1 *For any satisfiable formula F on n variables, the success probability of* ***RandomWalk*** *(F, y) is at least $(k/2(k-1))^n/p(n)$ for some polynomial p. Thus, by setting $I = (2(k-1)/k)^n \cdot p(n)$,* ***SCH*** *finds a satisfying assignment with high probability. When $k = 3$, this value of I is $O(1.334^n)$.*

Applications

The Schöning's result has been improved by a series of papers [1, 3, 9] based on the idea of [3]. Namely, RandomWalk is combined with the (polynomial time) 2SAT algorithm, which makes it possible to choose better initial assignments. For derandomization of **SCH**, see [2]. Iwama and Tamaki [4] developed a nontrivial combination of **SCH** with another famous, backtrack-type algorithm by [8], resulting in the then fastest algorithm with $O(1.324^n)$ running time. The current fastest algorithm is due to [10], which is based on the same approach as [4] and runs in time $O(1.32216^n)$.

Open Problems

k-SAT is probably the most popular NP-complete problem for which numerous researchers are competing for its fastest algorithm. Thus, improving its time bound is always a good research target.

Experimental Results

AI researchers have also been very active in SAT algorithms including local search; see, e.g., [5].

Cross-References

- Exact Algorithms for General CNF SAT
- Random Planted 3-SAT

Recommended Reading

1. Baumer S, Schuler R (2003) Improving a probabilistic 3-SAT algorithm by dynamic search and independent clause pairs. ECCC TR03-010. Also presented at SAT
2. Dantsin E, Goerdt A, Hirsch EA, Kannan R, Kleinberg J, Papadimitriou C, Raghavan P, Schöning U (2002) A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search. Theor Comput Sci 289(1):69–83
3. Hofmeister T, Schöning U, Schuler R, Watanabe O (2002) Probabilistic 3-SAT algorithm further improved. In: Proceedings 19th symposium on theoretical aspects of computer science, Juan-les-Pins. LNCS, vol 2285, pp 193–202
4. Iwama K, Tamaki S (2004) Improved upper bounds for 3-SA T. In: Proceedings 15th annual ACM-SIAM symposium on discrete algorithms, New Orleans, pp 321–322
5. Kautz H, Selman B (2003) Ten challenges redux: recent progress in propositional reasoning and search. In: Proceedings 9th international conference on principles and practice of constraint programming, Kinsale, pp 1–18
6. Monien B, Speckenmeyer E (1985) Solving satisfiability in less than 2^n steps. Discret Appl Math 10:287–295
7. Papadimitriou CH (1991) On selecting a satisfying truth assignment. In: Proceedings 32nd annual symposium on foundations of computer science, San Juan, pp 163–169
8. Paturi R, Pudlák P, Saks ME, Zane F (1998) An improved exponential-time algorithm for k-SAT. In: Proceedings 39th annual symposium on foundations of computer science, Palo Alto, pp 628–637; J ACM 52(3):337–364 (2006)
9. Rolf D (2003) 3-SAT $\in$ $RTIME(O(1.32793^n))$. ECCC TR03-054
10. Rolf D (2006) Improved bound for the PPSZ/Schöning-algorithm for 3-SAT. J Satisf Boolean Model Comput 1:111–122
11. Schöning U (1999) A probabilistic algorithm for k-SAT and constraint satisfaction problems. In: Proceedings 40th annual symposium on foundations of computer science, New York, pp 410–414

Exact Algorithms for Maximum Independent Set

Fabrizio Grandoni
IDSIA, USI-SUPSI, University of Lugano, Lugano, Switzerland

Keywords

Exact exponential algorithms; Maximum independent set

Years and Authors of Summarized Original Work

1977; Tarjan, Trojanowski
1985; Robson
1999; Beigel
2009; Fomin, Grandoni, Kratsch

Problem Definition

Let $G = (V, E)$ be an n-node undirected, simple graph without loops. A set $I \subseteq V$ is called an *independent set* of G if the nodes of I are pairwise not adjacent. The *maximum independent set* (MIS) problem asks to determine the maximum cardinality $\alpha(G)$ of an independent set of G. MIS is one of the best studied NP-hard problems.

We will need the following notation. The (open) *neighborhood* of a vertex v is $N(v) = \{u \in V : uv \in E\}$, and its *closed neighborhood* is $N[v] = N(v) \cup \{v\}$. The degree $\deg(v)$ of v is $|N(v)|$. For $W \subseteq V$, $G[W] = (W, E \cap \binom{W}{2})$ is the *graph induced* by W. We let $G - W = G[V - W]$.

Key Results

A very simple algorithm solves MIS (exactly) in $O^*(2^n)$ time: it is sufficient to enumerate all the subsets of nodes, check in polynomial time whether each subset is an independent set

or not, and return the maximum cardinality independent set. We recall that the O^* notation suppresses polynomial factors in the input size. However, much faster (though still exponential-time) algorithms are known. In more detail, there exist algorithms that solve MIS in worst-case time $O^*(c^n)$ for some constant $c \in (1, 2)$. In this section, we will illustrate some of the most relevant techniques that have been used in the design and analysis of exact MIS algorithms. Due to space constraints, our description will be slightly informal (please see the references for formal details).

Bounding the Size of the Search Tree

All the nontrivial exact MIS algorithms, starting with [7], are recursive branching algorithms. As an illustration, consider the following simple MIS algorithm `Alg1`. If the graph is empty, output $\alpha(G) = 0$ (base instance). Otherwise, choose any node v of maximum degree, and output

$$\alpha(G) = \max\{\alpha(G - \{v\}), 1 + \alpha(G - N[v])\}.$$

Intuitively, the subgraph $G - \{v\}$ corresponds to the choice of not including v in the independent set (v is *discarded*), while the subgraph $G - N[v]$ to the choice of including v in the independent set (v is *selected*). Observe that, when v is selected, the neighbors of v have to be discarded. We will later refer to this branching as a *standard branching*.

The running time of the above algorithm, and of branching algorithms more in general, can be bounded as follows. The recursive calls induce a *search tree*, where the root is the input instance and the leaves are base instances (that can be solved in polynomial time). Observe that each branching step can be performed in polynomial time (excluding the time needed to solve subproblems). Furthermore, the height of the search tree is bounded by a polynomial. Therefore, the running time of the algorithm is bounded by $O^*(L(n))$, where $L(n)$ is the maximum number of leaves of any search tree that can be generated by the considered algorithm on an input instance with n nodes. Let us assume that $L(n) \leq c^n$ for some constant $c \geq 1$. When we branch at node v, we generate two subproblems containing $n - 1$ and $n - |N[v]|$ nodes, respectively. Therefore, c has to satisfy $c^n \geq c^{n-1} + c^{n-|N[v]|}$. Assuming pessimistically $|N[v]| = 1$, one obtains $c^n \geq 2c^{n-1}$ and therefore $c \geq 2$. We can conclude that the running time of the algorithm is $O^*(2^n)$. Though the running time of `Alg1` does not improve on exhaustive search, much faster algorithms can be obtained by branching in a more careful way and using a similar type of analysis. This will be discussed in the next subsections.

Refined Branching Rules

Several refined branching rules have been developed for MIS. Let us start with some *reduction rules*, which reduce the problem without branching (alternatively, by branching on a single subproblem). An isolated node v can be selected w.l.o.g.:

$$\alpha(G) = 1 + \alpha(G - N[v]).$$

Observe that if $N[u] \subseteq N[v]$, then node v can be discarded w.l.o.g. (*dominance*):

$$\alpha(G) = \alpha(G - \{v\}).$$

This rule implies that nodes of degree 1 can always be selected.

Suppose that we branch at a node v, and in the branch where we discard v we select exactly one of its neighbors, say w. Then by replacing w with v, we obtain a solution of the same cardinality including v: this means that the branch where we select v has to provide the optimal solution. Therefore, we can assume w.l.o.g. that the optimal solution either contains v or at least 2 of its neighbors. This idea is exploited in the *folding* operation [1], which we next illustrate only in the case of degree-2 nodes. Let $N[v] = \{w_1, w_2\}$. Remove $N[v]$. If $w_1w_2 \notin E$, create a node v' and add edges between v' and nodes in $N(w_1) \cup N(w_2) - \{v\}$. Let $G_{\text{fold}}(v)$ be the resulting graph. Then, one has

$$\alpha(G) = 1 + \alpha(G_{\text{fold}}(v)).$$

Intuitively, including node v' in the optimal solution to $G_{\text{fold}}(v)$ corresponds to selecting both w_1 and w_2, while discarding v' corresponds to selecting v.

Let `Alg2` be the algorithm that exhaustively applies the mentioned reduction rules and then performs a standard branching on a node of maximum degree. Reduction rules reduce the number of nodes at least by 1; hence, we have the constraint $c^n \geq c^{n-1}$. If we branch at node v, $\deg(v) \geq 3$. This gives $c^n \geq c^{n-1} + c^{n-4}$, which is satisfied by $c \geq 1.380\ldots$. Hence, the running time is in $O^*(1.381^n)$.

Let us next briefly sketch some other useful ideas that lead to refined branchings. A *mirror* [3] of a node v is a node u at distance 2 from v such that $N(v) - N(u)$ induces a clique. By the above discussion, if we branch by discarding v, we can assume that we select at least two neighbors of v and therefore we have also to discard the mirrors $M(v)$ of v. In other terms, we can use the refined branching

$$\alpha(G) = \max\{\alpha(G - \{v\} - M(v)), 1 + \alpha(G - N[v])\}.$$

A *satellite* [5] of a node v is a node u at distance 2 from v such that there exists a node $u' \in N(v) \cap N(u)$ that satisfies $N[u'] - N[v] = \{u\}$. Observe that if an optimal solution discards u, then we can discard v as well by dominance since $N[u'] \subseteq N[v]$ in $G - \{u\}$. Therefore, we can assume that in the branch where we select v, we also select its satellites $S(v)$. In other terms,

$$\alpha(G) = \max\{\alpha(G - \{v\}), 1 + |S(v)| + \alpha(G - N[v] - \cup_{u \in S(v)} N[u])\}.$$

Another useful trick [4] is to branch on nodes that form a *small separator* (of size 1 or 2 in the graph), hence isolating two or more connected components that can be solved independently (see also [2, 5]).

Measure and Conquer

Above we always used the number n of nodes as a *measure* of the size of subproblems. As observed in [3], much tighter running time bounds can be achieved by using smarted measures. As an illustration, we will present a refined bound on the running time of `Alg2`.

Let us measure the size of subproblems with the number n_3 of nodes of degree at least 3 (*large* nodes). Observe that, when $n_3 = 0$, G is a collection of isolated nodes, paths, and cycles. Therefore, in that case, `Alg2` only applies reduction rules, hence solving the problem in polynomial time. In other terms, $L(n_3) = L(0) = 1$ in this case. If the algorithm applies any reduction rule, the number of large nodes cannot increase and we obtain the trivial inequality $c^{n_3} \geq c^{n_3}$. Suppose next that `Alg2` performs a standard branching at a node v. Note that at this point all nodes in the graph are large. If $\deg(v) \geq 4$, then we obtain the inequality $c^{n_3} \geq c^{n_3-1} + c^{n_3-5}$ which is satisfied by $c \geq 1.324\ldots$. Otherwise ($\deg(v) = 3$), observe that the neighbors of v have degree 3 in G and at most 2 in $G - \{v\}$. Therefore, the number of large nodes is at most $n_3 - 4$ in both subproblems $G - \{v\}$ and $G - N[v]$. This gives the inequality $c^{n_3} \geq 2c^{n_3-4}$ which is satisfied by $c \geq 2^{1/4} < 1.1893$. We can conclude that the running time of the algorithm is in $O^*(1.325^n)$. In [3], each node is assigned a weight which is a growing function of its degree, and the measure is the sum of node weights (a similar measure is used also in [2, 5]).

In [2], it is shown how to use a fast MIS algorithm for graphs of maximum degree Δ to derive faster MIS algorithms for graphs of maximum degree $\Delta + 1$. Here the measure used in the analysis is a combination of the number of nodes and edges.

Memorization

So far we described algorithms with polynomial space complexity. *Memorization* [6] is a technique to speed up exponential-time branching algorithms at the cost of an exponential space complexity. The basic idea is to store the optimal solution to subproblems in a proper (exponential size) data structure. Each time a new subproblem is generated, one first checks (in polynomial time) whether that subproblem was already solved before. This way one avoids to solve the same subproblem several times.

In order to illustrate this technique, it is convenient to consider the variant `Alg3` of `Alg2` where we do not apply folding. This way, each subproblem corresponds to some induced subgraph $G[W]$ of the input graph. We will also use the standard measure though memorization is compatible with measure and conquer. By adapting the analysis of `Alg2`, one obtains the constraint $c^n \geq c^{n-1} + c^{n-3}$ and hence a running time of $O^*(1.466^n)$. Next, consider the variant `Alg3mem` of `Alg3` where we apply memorization. Let $L_k(n)$ be the maximum number of subproblems on k nodes generated by `Alg3mem` starting from an instance with n nodes. A slight adaptation of the standard analysis shows that $L_k(n) \leq 1.466^{n-k}$. However, since there are at most $\binom{n}{k}$ induced subgraphs on k nodes and we never solve the same subproblem twice, one also has $L_k(n) \leq \binom{n}{k}$. Using Stirling's formula, one obtains that the two upper bounds are roughly equal for $k = \alpha n$ and $\alpha = 0.107\ldots$. We can conclude that the running time of `Alg3mem` is in $O^*(\sum_{k=0}^{n} L_k(n)) = O^*(\sum_{k=0}^{n} \min\{1.466^{n-k}, \binom{n}{k}\}) = O^*(\max_{k=0}^{n} \min\{1.466^{n-k}, \binom{n}{k}\}) = O^*(1.466^{(1-0.107)n}) = O^*(1.408^n)$. The analysis can be refined [6] by bounding the number of *connected* induced subgraphs with k nodes in graphs of small maximum degree.

Cross-References

▸ Exact Algorithms for Dominating Set

Recommended Reading

1. Beigel R (1999) Finding maximum independent sets in sparse and general graphs. In: ACM-SIAM symposium on discrete algorithms (SODA), Baltimore, pp 856–857
2. Bourgeois N, Escoffier B, Paschos VT, van Rooij JMM (2012) Fast algorithms for max independent set. Algorithmica 62(1–2):382–415
3. Fomin FV, Grandoni F, Kratsch D (2009) A measure & conquer approach for the analysis of exact algorithms. J ACM 56(5):Article no. 25
4. Fürer M (2006) A faster algorithm for finding maximum independent sets in sparse graphs. In: Latin American theoretical informatics symposium (LATIN), Valdivia, pp 491–501
5. Kneis J, Langer A, Rossmanith P (2009) A fine-grained analysis of a simple independent set algorithm. In: Foundations of software technology and theoretical computer science (FSTTCS), Kanpur, pp 287–298
6. Robson JM (1986) Algorithms for maximum independent sets. J Algorithms 7(3):425–440
7. Tarjan R, Trojanowski A (1977) Finding a maximum independent set. SIAM J Comput 6(3):537–546

E

Exact Algorithms for Maximum Two-Satisfiability

Ryan Williams
Department of Computer Science, Stanford University, Stanford, CA, USA

Keywords

Max 2-SAT

Years and Authors of Summarized Original Work

2004; Williams

Problem Definition

In the maximum 2-satisfiability problem (abbreviated as MAX 2-SAT), one is given a Boolean formula in conjunctive normal form, such that each clause contains at most two literals. The task is to find an assignment to the variables of the formula such that a maximum number of clauses are satisfied.

MAX 2-SAT is a classic optimization problem. Its decision version was proved *NP*-complete by Garey, Johnson, and Stockmeyer [7], in stark contrast with 2-SAT which is solvable in linear time [2]. To get a feeling for the difficulty of the problem, the *NP*-completeness reduction is sketched here. One can transform any 3-SAT instance F into a MAX 2-SAT instance F', by replacing each clause of F such as

$$c_i = (\ell_1 \vee \ell_2 \vee \ell_3),$$

where ℓ_1, ℓ_2, and ℓ_3 are arbitrary literals, with the collection of 2-CNF clauses

$$(\ell_1), (\ell_2), (\ell_3), (c_i), (\neg\ell_1 \vee \neg\ell_2), (\neg\ell_2 \vee \neg\ell_3),$$
$$(\neg\ell_1 \vee \neg\ell_3), (\ell_1 \vee c_i), (\ell_2 \vee c_i), (\ell_3 \vee c_i),$$

where c_i is a new variable. The following are true:

- If an assignment satisfies c_i, then exactly seven of the ten clauses in the 2-CNF collection can be satisfied.
- If an assignment does not satisfy c_i, then exactly six of the ten clauses can be satisfied.

If F is satisfiable then there is an assignment satisfying 7/10 of the clauses in F', and if F is not satisfiable, then no assignment satisfies more than 7/10 of the clauses in F'. Since 3-SAT reduces to MAX 2-SAT, it follows that MAX 2-SAT (as a decision problem) is *NP*-complete.

Notation

A CNF formula is represented as a set of clauses.

The letter ω denotes the smallest real number such that for all $\epsilon > 0$, n by n matrix multiplication over a field can be performed in $O(n^{\omega+\epsilon})$ field operations. Currently, it is known that $\omega < 2.373$ [4, 16]. The field matrix product of two matrices A and B is denoted by $A \times B$.

Let A and B be matrices with entries from $\mathbb{R} \cup \{\infty\}$. The *distance product* of A and B (written in shorthand as $A \circledast B$) is the matrix C defined by the formula

$$C[i,j] = \min_{k=1,\ldots,n} \{A[i,k] + B[k,j]\}.$$

A word on m's and n's: in reference to graphs, m and n denote the number of edges and the number of nodes in the graph, respectively. In reference to CNF formulas, m and n denote the number of clauses and the number of variables, respectively.

Key Result

The primary result of this entry is a procedure solving Max 2-Sat in $O(m \cdot 2^{\omega n/3})$ time. The method can be generalized to *count* the number of solutions to *any* constraint optimization problem with at most two variables per constraint. Indeed, in the same running time, one can find a Boolean assignment that maximizes any given degree-two polynomial in n variables [18, 19]. In this entry, we shall restrict attention to be Max 2-Sat, for simplicity. There are several other known exact algorithms for Max 2-Sat that are more effective in special cases, such as sparse instances [3, 8, 9, 11–13, 15, 17]. The procedure described below is the only one known (to date) that runs in c^n steps for a constant $c < 2$.

Key Idea

The algorithm gives a reduction from MAX 2-SAT to the problem MAX TRIANGLE, in which one is given a graph with integer weights on its nodes and edges, and the goal is to output a 3-cycle of maximum weight. At first, the existence of such a reduction sounds strange, as MAX TRIANGLE can be trivially solved in $O(n^3)$ time by trying all possible 3-cycles. The key is that the reduction exponentially increases the problem size, from a MAX 2-SAT instance with m clauses and n variables to a MAX TRIANGLE instance having $O(2^{2n/3})$ edges, $O(2^{n/3})$ nodes, and weights in the range $\{-m, \ldots, m\}$.

Note that if MAX TRIANGLE required $\Theta(n^3)$ time to solve, then the resulting MAX 2-SAT algorithm would take $\Theta(2^n)$ time, rendering the above reduction pointless. However, it turns out that the brute-force search of $O(n^3)$ for MAX TRIANGLE is not the best one can do: using fast matrix multiplication, there is an algorithm for MAX TRIANGLE that runs in $O(Wn^\omega)$ time on graphs with weights in the range $\{-W, \ldots, W\}$.

Main Algorithm

First, a reduction from MAX 2-SAT to MAX TRIANGLE is described, arguing that each triangle of

weight K in the resulting graph is in one-to-one correspondence with an assignment that satisfies K clauses of the MAX 2-SAT instance. Let a, b be reals, and let $\mathbb{Z}[a,b] := [a,b] \cap \mathbb{Z}$.

Lemma 1 *If* MAX TRIANGLE *on graphs with n nodes and weights in $\mathbb{Z}[-W,W]$ is solvable in $O(f(W)\cdot g(n))$ time, for polynomials f and g, then* MAX 2-SAT *is solvable in $O(f(m)\cdot g(2^{n/3}))$ time, where m is the number of clauses and n is the number of variables.*

Proof Let C be a given 2-CNF formula. Assume without loss of generality that n is divisible by 3. Let F be an instance of MAX 2-SAT. Arbitrarily partition the n variables of F into three sets P_1, P_2, P_3, each having $n/3$ variables. For each P_i, make a list L_i of all $2^{n/3}$ assignments to the variables of P_i.

Define a graph $G = (V,E)$ with $V = L_1 \cup L_2 \cup L_3$ and $E = \{(u,v) | u \in P_i, v \in P_j, i \neq j\}$. That is, G is a complete tripartite graph with $2^{n/3}$ nodes in each part, and each node in G corresponds to an assignment to $n/3$ variables in C. Weights are placed on the nodes and edges of G as follows. For a node v, define $w(v)$ to be the number of clauses that are satisfied by the partial assignment denoted by v. For each edge $\{u,v\}$, define $w(\{u,v\}) = -W_{uv}$, where W_{uv} is the number of clauses that are satisfied by *both* u and v.

Define the weight of a triangle in G to be the total sum of all weights and nodes in the triangle.

Claim 1 There is a one-to-one correspondence between the triangles of weight K in G and the variable assignments satisfying exactly K clauses in F.

Proof Let a be a variable assignment. Then there exist unique nodes $v_1 \in L_1, v_2 \in L_2$, and $v_3 \in L_3$ such that a is precisely the concatenation of v_1, v_2, v_3 as assignments. Moreover, any triple of nodes $v_1 \in L_1, v_2 \in L_2$, and $v_3 \in L_3$ corresponds to an assignment. Thus, there is a one-to-one correspondence between triangles in G and assignments to F.

The number of clauses satisfied by an assignment is exactly the weight of its corresponding triangle. To see this, let $T_a = \{v_1, v_2, v_3\}$ be the triangle in G corresponding to assignment a. Then

$$\begin{aligned} w(T_a) &= w(v_1) + w(v_2) + w(v_3) + w(\{v_1,v_2\}) \\ &\quad + w(\{v_2,v_3\}) + w(\{v_1,v_3\}) \\ &= \sum_{i=1}^{3} |\{c \in F | v_i \text{ satisfies } F\}| \\ &\quad - \sum_{i,j:i\neq j} |\{c \in F | v_i \text{ and } v_j \text{ satisfy } F\}| \\ &= |\{c \in F | a \text{ satisfies } F\}|, \end{aligned}$$

where the last equality follows from the inclusion-exclusion principle.

Notice that the number of nodes in G is $3{\cdot}2^{n/3}$, and the absolute value of any node and edge weight is m. Therefore, running a MAX TRIANGLE algorithm on G, a solution to MAX 2-SAT, is obtained in $O(f(m)\cdot g(3{\cdot}2^{n/3}))$, which is $O(f(m)\cdot g(2^{n/3}))$ since g is a polynomial. This completes the proof of Lemma 1.

Next, a procedure is described for finding a maximum triangle faster than brute-force search, using fast matrix multiplication. Alon, Galil, and Margalit [1] (following Yuval [22]) showed that the distance product for matrices with entries drawn from $\mathbb{Z}[-W,W]$ can be computed using fast matrix multiplication as a subroutine.

Theorem 1 (Alon, Galil, Margalit [1]) *Let A and B be $n \times n$ matrices with entries from $\mathbb{Z}[-W,W] \cup \{\infty\}$. Then $A \circledast B$ can be computed in $O(Wn^{\omega}\log n)$ time.*

Proof (Sketch) One can replace ∞ entries in A and B with $2W+1$ in the following. Define matrices A' and B', where

$$A'[i,j] = x^{3W-A[i,j]},\ B'[i,j] = x^{3W-B[i,j]},$$

and x is a variable. Let $C = A' \times B'$. Then

$$C[i,j] = \sum_{k=1}^{n} x^{6W-A[i,k]-B[k,j]}.$$

The next step is to pick a number x that makes it easy to determine, from the sum of arbitrary powers of x, the largest power of x appearing in the sum; this largest power immediately gives the minimum $A[i,k] + B[k,j]$. Each $C[i,j]$ is a polynomial in x with coefficients from $\mathbb{Z}[0,n]$. Suppose each $C[i,j]$ is evaluated at $x = (n+1)$. Then each entry of $C[i,j]$ can be seen as an $(n+1)$-ary number, and the position of this number's most significant digit gives the minimum $A[i,k] + B[k,j]$.

In summary, $A \otimes_d B$ can be computed by constructing

$$A'[i,j] = (n+1)^{3W - A[i,j]},$$

$$B'[i,j] = (n+1)^{3W - B[i,j]}$$

in $O(W \log n)$ time per entry, computing $C = A' \times B'$ in $O(n^\omega \cdot (W \log n))$ time (as the sizes of the entries are $O(W \log n)$), then extracting the minimum from each entry of C, in $O(n^2 \cdot W \log n)$ time. Note if the minimum for an entry $C[i,j]$ is at least $2W+1$, then $C[i,j] = \infty$.

Using the fast distance product algorithm, one can solve MAX TRIANGLE faster than brute force. The following is based on an algorithm by Itai and Rodeh [10] for detecting if an unweighted graph has a triangle in less than n^3 steps. The result can be generalized to *counting* the number of *k-cliques*, for arbitrary $k \geq 3$. (To keep the presentation simple, the counting result is omitted. Concerning the k-clique result, there is unfortunately no asymptotic runtime benefit from using a k-clique algorithm instead of a triangle algorithm, given the current best algorithms for these problems.)

Theorem 2 MAX TRIANGLE *can be solved in* $O(W n^\omega \log n)$*, for graphs with weights drawn from* $\mathbb{Z}[-W, W]$.

Proof First, it is shown that a weight function on nodes and edges can be converted into an equivalent weight function with weights on only edges. Let w be the weight function of G, and redefine the weights to be:

$$w'(\{u,v\}) = \frac{w(u) + w(v)}{2} + w(\{u,v\}),$$

$$w'(u) = 0.$$

Note the weight of a triangle is unchanged by this reduction.

The next step is to use a fast distance product to find a maximum weight triangle in an edge-weighted graph of n nodes. Construe the vertex set of G as the set $\{1,\ldots,n\}$. Define A to be the $n \times n$ matrix such that $A[i,j] = -w(\{i,j\})$ if there is an edge $\{i,j\}$, and $A[i,j] = \infty$ otherwise. The claim is that there is a triangle through node i of weight at least K if and only if $(A \circledast A \circledast A)[i,i] \leq -K$. This is because $(A \circledast A \circledast A)[i,i] \leq -K$ if and only if there are distinct j and k such that $\{i,j\}, \{j,k\}, \{k,i\}$ are edges and $A[i,j] + A[j,k] + A[k,i] \leq -K$, i.e., $w(\{i,j\}) + w(\{j,k\}) + w(\{k,i\}) \geq K$.

Therefore, by finding an i such that $(A \circledast A \circledast A)[i,i]$ is minimized, one obtains a node i contained in a maximum triangle. To obtain the actual triangle, check all m edges $\{j,k\}$ to see if $\{i,j,k\}$ is a triangle.

Theorem 3 MAX 2-SAT *can be solved in* $O(m \cdot 1.732^n)$ *time.*

Proof Given a set of clauses C, apply the reduction from Lemma 1 to get a graph G with $O(2^{n/3})$ nodes and weights from $\mathbb{Z}[-m,m]$. Apply the algorithm of Theorem 2 to output a max triangle in G in $O(m \cdot 2^{\omega n/3} \log(2^{n/3})) = O(m \cdot 1.732^n)$ time, using the $O(n^{2.376})$ matrix multiplication of Coppersmith and Winograd [4].

Applications

By modifying the graph construction, one can solve other problems in $O(1.732^n)$ time, such as Max Cut, Minimum Bisection, and Sparsest Cut. In general, any constraint optimization problem for which each constraint has at most two variables can be solved faster using the above approach. For more details, see [18] and the survey by Woeginger [21]. Techniques similar to the above algorithm have also been used by Dorn

[6] to speed up dynamic programming for some problems on planar graphs (and in general, graphs of bounded branchwidth).

Open Problems

- Improve the space usage of the above algorithm. Currently, $\Theta(2^{2n/3})$ space is needed. A very interesting open question is if there is a $O(1.99^n)$ time algorithm for MAX 2-SAT that uses only *polynomial* space. This question would have a positive answer if one could find an algorithm for solving the k-CLIQUE problem that uses polylogarithmic space and $n^{k-\delta}$ time for some $\delta > 0$ and $k \geq 3$.
- Find a faster-than-2^n algorithm for MAX 2-SAT that does not require fast matrix multiplication. The fast matrix multiplication algorithms have the unfortunate reputation of being impractical.
- Generalize the above algorithm to work for MAX k-SAT, where k is any positive integer. The current formulation would require one to give an efficient algorithm for finding a small hyperclique in a hypergraph. However, no general results are known for this problem. It is conjectured that for all $k \geq 2$, MAX k-SAT is in $\bar{O}(2^{n(1-\frac{1}{k+1})})$ time, based on the conjecture that matrix multiplication is in $n^{2+o(1)}$ time [17].

Cross-References

▸ All Pairs Shortest Paths via Matrix Multiplication

Recommended Reading

1. Alon N, Galil Z, Margalit O (1997) On the exponent of the all-pairs shortest path problem. J Comput Syst Sci 54:255–262
2. Aspvall B, Plass MF, Tarjan RE (1979) A linear-time algorithm for testing the truth of certain quantified boolean formulas. Inf Proc Lett 8(3):121–123
3. Bansal N, Raman V (1999) Upper bounds for Max Sat: further improved. In: Proceedings of ISAAC, Chennai. LNCS, vol 1741. Springer, Berlin, pp 247–258
4. Coppersmith D, Winograd S (1990) Matrix multiplication via arithmetic progressions. JSC 9(3):251–280
5. Dantsin E, Wolpert A (2006) Max SAT for formulas with constant clause density can be solved faster than in $O(2^n)$ time. In: Proceedings of the 9th international conference on theory and applications of satisfiability testing, Seattle. LNCS, vol 4121. Springer, Berlin, pp 266–276
6. Dorn F (2006) Dynamic programming and fast matrix multiplication. In: Proceedings of 14th annual European symposium on algorithms, Zurich. LNCS, vol 4168. Springer, Berlin, pp 280–291
7. Garey M, Johnson D, Stockmeyer L (1976) Some simplified NP-complete graph problems. Theor Comput Sci 1:237–267
8. Gramm J, Niedermeier R (2000) Faster exact solutions for Max2Sat. In: Proceedings of CIAC. LNCS, vol 1767, Rome. Springer, Berlin, pp 174–186
9. Hirsch EA (2000) A $2^{m/4}$-time algorithm for Max 2-SAT: corrected version. Electronic colloquium on computational complexity report TR99-036
10. Itai A, Rodeh M (1978) Finding a minimum circuit in a graph. SIAM J Comput 7(4):413–423
11. Kneis J, Mölle D, Richter S, Rossmanith P (2005) Algorithms based on the treewidth of sparse graphs. In: Proceedings of workshop on graph theoretic concepts in computer science, Metz. LNCS, vol 3787. Springer, Berlin, pp 385–396
12. Kojevnikov A, Kulikov AS (2006) A new approach to proving upper bounds for Max 2-SAT. In: Proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithms, Miami, pp 11–17
13. Mahajan M, Raman V (1999) Parameterizing above guaranteed values: MAXSAT and MAXCUT. J Algorithms 31(2):335–354
14. Niedermeier R, Rossmanith P (2000) New upper bounds for maximum satisfiability. J Algorithms 26:63–88
15. Scott A, Sorkin G (2003) Faster algorithms for MAX CUT and MAX CSP, with polynomial expected time for sparse instances. In: Proceedings of RANDOM-APPROX 2003, Princeton. LNCS, vol 2764. Springer, Berlin, pp 382–395
16. Vassilevska Williams V (2012) Multiplying matrices faster than Coppersmith-Winograd. In: Proceedings of the 44th annual ACM symposium on theory of computing, New York, pp 887–898
17. Williams R (2004) On computing k-CNF formula properties. In: Theory and applications of satisfiability testing. LNCS, vol 2919. Springer, Berlin, pp 330–340
18. Williams R (2005) A new algorithm for optimal 2-constraint satisfaction and its implications. Theor Comput Sci 348(2–3):357–365
19. Williams R (2007) Algorithms and resource requirements for fundamental problems. PhD thesis, Carnegie Mellon University

20. Woeginger GJ (2003) Exact algorithms for NP-hard problems: a survey. In: Combinatorial optimization – Eureka! You shrink! LNCS, vol 2570. Springer, Berlin, pp 185–207
21. Woeginger GJ (2004) Space and time complexity of exact algorithms: some open problems. In: Proceedings of 1st international workshop on parameterized and exact computation (IWPEC 2004), Bergen. LNCS, vol 3162. Springer, Berlin, pp 281–290
22. Yuval G (1976) An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. Inf Process Lett 4(6):155–156

Exact Algorithms for Treewidth

Ioan Todinca
INSA Centre Val de Loire, Universite d'Orleans, Orléans, France

Keywords

Extremal combinatorics; Potential maximal cliques; Treewidth

Years and Authors of Summarized Original Work

2008; Fomin, Kratsch, Todinca, Villanger
2012; Bodlaender, Fomin, Koster, Kratsch, Thilikos
2012; Fomin, Villanger

Problem Definition

The *treewidth* parameter intuitively measures whether the graph has a "treelike" structure. Given an undirected graph $G = (V, E)$, a *tree decomposition* of G is a pair $(\mathcal{X}, T)$, where $T = (I, F)$ is a tree and $\mathcal{X} = \{X_i \mid i \in I\}$ is a collection of subsets of V called *bags* satisfying:

1. $\bigcup_{i \in I} X_i = V$,
2. For each edge uv of G, there is a bag X_i containing both endpoints,
3. For all $v \in V$, the set $\{i \in I \mid v \in X_i\}$ induces a connected subtree of T.

The *width* of a tree decomposition $(\mathcal{X}, T)$ is the size of its largest bag, minus one. The *treewidth* of G, denoted by $\mathrm{tw}(G)$, is the minimum width over all possible tree decompositions. One can easily observe that n-vertex graphs have treewidth at most $n - 1$ and that the graphs of treewidth at most one are exactly the forests.

Given a graph G and a number k, the TREEWIDTH problem consists in deciding if $\mathrm{tw}(G) \leq k$. Arnborg, Corneil, and Proskurowski show that the problem is NP-hard [1]. On the positive side, Bodlaender [2] gives an algorithm solving the problem in time $2^{\mathcal{O}(k^3)}n$. Bouchitté and Todinca [4,5] prove that the problem is polynomial on classes of graphs with polynomially many minimal separators, with an algorithm based on the notion of *potential maximal clique*. This latter technique is also employed by several exact, moderately exponential algorithms for TREEWIDTH.

Key Results

TREEWIDTH can be solved in $\mathcal{O}^*(2^n)$ time by adapting the $\mathcal{O}(n^k)$ algorithm of Arnborg et al. [1] or the Held-Karp technique initially designed for the TRAVELING SALESMAN problem [12]. (We use here the $\mathcal{O}^*$ notation that suppresses polynomial factors.) Fomin et al. [9] break this "natural" 2^n barrier with an algorithm running in time $\mathcal{O}^*(1.8135^n)$, using the same space complexity. Bodlaender et al. [3] present a polynomial-space algorithm running in $\mathcal{O}^*(2.9512^n)$ time. A major improvement for both results is due to Fomin and Villanger [8].

Theorem 1 ([8]) *The* TREEWIDTH *problem can be solved in* $\mathcal{O}^*(1.7549^n)$ *time using exponential space and in* $\mathcal{O}^*(2.6151^n)$ *time using polynomial space.*

These algorithms use an alternative definition for treewidth. A graph $H = (V, E)$ is *chordal* or *triangulated* if it has no induced cycle with four

or more vertices. It is well-known that a chordal graph has tree decompositions whose bags are exactly its maximal cliques. Given an arbitrary graph $G = (V, E)$, a chordal graph $H = (V, F)$ on the same vertex set is called a *minimal triangulation* of G if H contains G as a subgraph and no chordal subgraph of H contains G. The treewidth of G can be defined as the minimum clique size of H minus one, over all minimal triangulations H of G.

A vertex subset S of graph G is a *minimal separator* if there are two distinct components $G[C]$ and $G[D]$ of the graph $G[V \setminus S]$ such that $N_G(C) = N_G(D) = S$ ($N_G(C)$ denotes the neighborhood of C in graph G).

A vertex subset Ω of G is a *potential maximal clique* if there exists some minimal triangulation H of G such that Ω induces a maximal clique in H. Potential maximal cliques are characterized as follows [4]: Ω is a potential maximal clique of G if and only if (i) for each pair of vertices $u, v \in \Omega$, u and v are adjacent or see a same component of $G[V \setminus \Omega]$, and (ii) no component of $G[V \setminus \Omega]$ sees the whole set Ω. As an example, when G is a cycle, its minimal separators are exactly the pairs of nonadjacent vertices, and the potential maximal cliques are exactly the triples of vertices.

A *block* is pair (S, C) such that S is a minimal separator of G and $G[C]$ is a component of $G[V \setminus S]$. Denote by $R_G(S, C)$ the graph obtained from $G[S \cup C]$ by turning S into a clique, i.e., by adding all missing edges with both endpoints in S. The treewidth of G can be obtained as follows:

$$\mathrm{tw}(G) = \min_S \left(\max_C \mathrm{tw}(R(S, C)) \right) \quad (1)$$

where the minimum is taken over all minimal separators S and the maximum is taken over all connected components $G[C]$ of $G[V \setminus S]$.

All quantities $\mathrm{tw}(R_G(S, C))$ can be computed by dynamic programming over blocks (S, C), by increasing the size of $S \cup C$. We only consider here blocks (S, C) such that $S = N_G(C)$ (see [4] for more details).

$$\mathrm{tw}(R_G(S, C)) = \min_{S \subset \Omega \subseteq S \cup C} \left(\max_{1 \le i \le p} (|\Omega| - 1, \mathrm{tw}(R_G(S_i, C_i))) \right) \quad (2)$$

where the minimum is taken over all potential maximal cliques Ω with $S \subset \Omega \subseteq S \cup C$ and the maximum is taken over all pairs (S_i, C_i), where $G[C_i]$ is a component of $G[C \setminus \Omega]$ and $S_i = N_G(C_i)$. Let Π_G denote the set of all potential maximal cliques of graph G. It was pointed in [9] that the number of triples (S, Ω, C) like in Eq. 2 is at most $n|\Pi_G|$, which proves that TREEWIDTH can be computed in $\mathcal{O}^*(|\Pi_G|)$ time and space, if Π_G is given in the input.

Therefore, it remains to give a good upper bound for the number $|\Pi_G|$ of potential maximal cliques of G, together with efficient algorithms for listing these objects. Based on the previously mentioned characterization of potential maximal cliques, Kratsch et al. provide an algorithm listing them in time $\mathcal{O}^*(1.8135^n)$. Fomin and Villanger [8] improve this result, thanks to the following combinatorial theorem:

Theorem 2 ([8]) *Let $G = (V, E)$ be an n-vertex graph, let v be a vertex of G, and b, f be two integers. The number of vertex subsets B containing v such that $G[B]$ is connected, $|B| = b + 1$, and $|N_G(B)| = f$ is at most $\binom{b+f}{f}$.*

The elegant inductive proof also leads to an $\mathcal{O}^*(\binom{b+f}{f})$ time algorithm listing all such sets B. Eventually, the potential maximal cliques of an input graph G can be listed in $\mathcal{O}^*(1.7549^n)$ time [8]. This bound was further improved to $\mathcal{O}^*(1.7347^n)$ in [7].

In order to obtain polynomial-space algorithms for TREEWIDTH, Bodlaender et al. [3] provide a relatively simple divide-and-conquer algorithm, based on the Held-Karp approach, running in $\mathcal{O}^*(4^n)$ time. They also observe that Eq. 1 can be used for recursive, polynomial-space algorithms, by replacing the minimal separators S by *balanced* separators, in the sense that each component of $G[V \setminus S]$ contains at most $n/2$ vertices. This leads to polynomial-space algorithm with $\mathcal{O}^*(2.9512^n)$ running time.

Fomin and Villanger [8] restrict the balanced separators to a subset of the potential maximal cliques, and based on Theorem 2 they obtain, still using polynomial space, a running time of $\mathcal{O}^*(2.6151^n)$.

We refer to the book of Fomin and Kratsch [6] for more details on the TREEWIDTH problem and more generally on exact algorithms.

Applications

Exact algorithms based on potential maximal cliques have been extended to many other problems like FEEDBACK VERTEX SET, LONGEST INDUCED PATH, or MAXIMUM INDUCED SUBGRAPH WITH A FORBIDDEN PLANAR MINOR. More generally, for any constant t and any property $\mathcal{P}$ definable in counting monadic second-order logic, consider the problem of finding, in an arbitrary graph G, a maximum-size induced subgraph $G[F]$ of treewidth at most t and with property $\mathcal{P}$. This generic problem can be solved in $\mathcal{O}^*(|\Pi_G|)$ time, if Π_G is part of the input [7, 10]. Therefore, there is an algorithm in $\mathcal{O}^*(1.7347^n)$ time for the problem, significantly improving the $\mathcal{O}^*(2^n)$ time for exhaustive search.

Open Problems

Currently, the best known upper bound on the number of potential maximal cliques in n-vertex graphs is of $\mathcal{O}^*(1.7347^n)$ and does not seem to be tight [7]. Simple examples show that this bound is of at least $3^{n/3} \sim 1.4425^n$. A challenging question is to find a tight upper bound and efficient algorithms enumerating all potential maximal cliques of arbitrary graphs.

Experimental Results

Several experimental results are reported in [3], especially on an "engineered" version of the $\mathcal{O}^*(2^n)$ time and space algorithm based on the Held-Karp approach. This dynamic programming algorithm is compared with the branch and bound approach of Gogate and Dechter [11] on instances of up to 50 vertices. The results are relatively similar. Bodlaender et al. [3] also observe that the polynomial-space algorithms become too slow even for small instances.

Cross-References

▸ Kernelization, Preprocessing for Treewidth

Recommended Reading

1. Arnborg S, Corneil DG, Proskurowski A (1987) Complexity of finding embeddings in a k-tree. SIAM J Algebr Discret Methods 8(2):277–284
2. Bodlaender HL (1996) A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J Comput 25(6):1305–1317
3. Bodlaender HL, Fomin FV, Koster AMCA, Kratsch D, Thilikos DM (2012) On exact algorithms for treewidth. ACM Trans Algorithms 9(1):12
4. Bouchitté V, Todinca I (2001) Treewidth and minimum fill-in: grouping the minimal separators. SIAM J Comput 31(1):212–232
5. Bouchitté V, Todinca I (2002) Listing all potential maximal cliques of a graph. Theor Comput Sci 276(1–2):17–32
6. Fomin FV, Kratsch D (2010) Exact exponential algorithms, 1st edn. Springer, New York
7. Fomin FV, Villanger Y (2010) Finding induced subgraphs via minimal triangulations. In: Marion JY, Schwentick T (eds) STACS, Nancy. LIPIcs, vol 5. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, pp 383–394
8. Fomin FV, Villanger Y (2012) Treewidth computation and extremal combinatorics. Combinatorica 32(3):289–308
9. Fomin FV, Kratsch D, Todinca I, Villanger Y (2008) Exact algorithms for treewidth and minimum fill-in. SIAM J Comput 38(3):1058–1079
10. Fomin FV, Todinca I, Villanger Y (2014) Large induced subgraphs via triangulations and cmso. In: Chekuri C (ed) SODA, Portland. SIAM, pp 582–583
11. Gogate V, Dechter R (2004) A complete anytime algorithm for treewidth. In: Chickering DM, Halpern JY (eds) UAI, Banff. AUAI Press, pp 201–208
12. Held M, Karp RM (1962) A dynamic programming approach to the sequencing problem. J Soc Ind Appl Math 10(1):196–210

Exact Algorithms on Graphs of Bounded Average Degree

Marcin Pilipczuk
Institute of Informatics, University of Bergen, Bergen, Norway
Institute of Informatics, University of Warsaw, Warsaw, Poland

Keywords

Bounded average degree graphs; Bounded degree graphs; Chromatic number; Counting perfect matchings; Traveling salesman problem; TSP

Years and Authors of Summarized Original Work

2010; Björklund, Husfeldt, Kaski, Koivisto
2012; Björklund, Husfeldt, Kaski, Koivisto
2013; Cygan, Pilipczuk
2014; Golovnev, Kulikov, Mihajlin

Problem Definition

We focus on the following question: how an assumption on the sparsity of an input graph, such as bounded (average) degree, can help in designing exact (exponential-time) algorithms for NP-hard problems. The following classic problems are studied:

Traveling Salesman Problem Find a minimum-length Hamiltonian cycle in an input graph with edge weights.

Chromatic Number Find a minimum number k for which the vertices of an input graph can be colored with k colors such that no two adjacent vertices receive the same color.

Counting Perfect Matchings Find the number of perfect matchings in an input graph.

Key Results

The classic algorithms of Bellman [1] and Held and Karp [10] for traveling salesman problem run in $2^n n^{\mathcal{O}(1)}$ time for n-vertex graphs. Using the inclusion-exclusion principle, the chromatic number of an input graph can be determined within the same running time bound [4]. Finally, as long as counting perfect matchings is concerned, a half-century-old $2^{n/2} n^{\mathcal{O}(1)}$-time algorithm of Ryser for bipartite graphs [12] has only recently been transferred to arbitrary graphs by Björklund [2].

In all three aforementioned cases, it is widely open whether the 2^n or $2^{n/2}$ factor in the running time bound can be improved. In 2008, Björklund, Husfeldt, Kaski, and Koivisto [5,6] observed that such an improvement can be made if we restrict ourselves to bounded degree graphs. Further work of Cygan and Pilipczuk [8] and Golovnev, Kulikov, and Mihajlin [9] extended these results to graphs of bounded average degree.

Bounded Degree Graphs

Traveling Salesman Problem

Let us present the approach of Björklund, Husfeldt, Kaski, and Koivisto on the example of traveling salesman problem. Assume we are given an n-vertex edge-weighted graph G. The classic dynamic programming algorithm picks a root vertex r and then, for every vertex $v \in V(G)$ and every set $X \subseteq V(G)$ containing v and r, computes $T[X, v]$: the minimum possible length of a path in G with vertex set X that starts in r and ends in v. The running time bound $2^n n^{\mathcal{O}(1)}$ is dominated by the number of choices of the set X.

The simple, but crucial, observation is as follows: if a set X satisfies $X \cap N_G[u] = \{u\}$ for some $u \in V(G) \setminus \{r\}$, then the values $T[X, v]$ are essentially useless, as no path starting in r can visit the vertex u without visiting any neighbor of u (here $N_G[u] = N_G(u) \cup \{u\}$ stands for the closed neighborhood of u). Let us call a set $X \subseteq V(G)$ *useful* if $X \cap N_G[u] \neq \{u\}$ for every $u \in V(G) \setminus \{r\}$. The argumentation so far proved that we may skip the computation of $T[X, v]$ for all sets X that are not useful. The natural question is how many different useful sets may exist in an n-vertex graph?

Consider the following greedy procedure: initiate $A = \emptyset$ and, as long as there exists a vertex $u \in V(G)$ such that $N_G[u] \cap N_G[A] = \emptyset$,

add an arbitrarily chosen vertex u to the set A. By construction, the set A satisfies the following property: for every $u_1, u_2 \in A$, we have $N_G[u_1] \cap N_G[u_2] = \emptyset$. An interesting fact is that $|A| = \Omega(n)$ for graphs of bounded degree: whenever we insert a vertex u into the set A, we cannot later insert into A any neighbor of u nor any neighbor of a neighbor of u. However, if the maximum degree of G is bounded by d, then there are at most d neighbors of u, and every such neighbor has at most $d-1$ further neighbors. Consequently, when we insert a vertex u into A, we prohibit at most $d + d(d-1) = d^2$ other vertices from being inserted into A, and $|A| \geq n/(1+d^2)$.

It is easy to adjust the above procedure such that the root vertex r does not belong to A. Observe that for every useful set X and every $u \in A$, we have $X \cap N_G[u] \neq \{u\}$ and, furthermore, the sets $N_G[u]$ for $u \in A$ are pairwise disjoint. We can think of choosing a useful set X as follows: first, for every $u \in A$, we choose the intersection $X \cap N_G[u]$ (there are $2^{|N_G[u]|} - 1$ choices, as the choice $\{u\}$ is forbidden), and, second, we choose the set $X \setminus N_G[A]$. Hence, the number of useful sets is bounded by

$$\begin{aligned}\left(\prod_{u\in A} 2^{|N_G[u]|} - 1\right) \cdot 2^{n-|N_G[A]|} &= 2^n \cdot \prod_{u\in A}\left(1 - 2^{-|N_G[u]|}\right) \leq 2^n \cdot \prod_{u\in A}(1-2^{-d-1}) \\ &= 2^n \cdot (1-2^{-d-1})^{|A|} \leq 2^n \cdot (1-2^{-d-1})^{\frac{n}{1+d^2}} \\ &= \left(2 \cdot \sqrt[1+d^2]{1-2^{-d-1}}\right)^n.\end{aligned}$$

Thus, for every degree bound d, there exists a constant $\varepsilon_d > 0$ such that the number of useful sets in an n-vertex graph of maximum degree d is bounded by $(2-\varepsilon_d)^n$, yielding a $(2-\varepsilon_d)^n n^{\mathcal{O}(1)}$-time algorithm for traveling salesman problem. A better dependency on d in the formula for ε_d can be obtained using a projection theorem of Chung, Frankl, Graham, and Shearer [7] (see [5]).

Chromatic Number

A similar reasoning can be performed for the problem of determining the chromatic number of an input graph. Here, it is useful to rephrase the problem as follows: find a minimum number k such that the vertex set of an input graph can be covered by k maximal independent sets; note that we do not insist that the independent sets are disjoint. Observe that if X is a set of vertices covered by one or more such maximal independent sets, we have $X \cap N_G[u] \neq \emptyset$ for every $u \in V(G)$, as otherwise the vertex u should have been included into one of the covering sets. Hence, we can call a set $X \subseteq V(G)$ *useful* if it intersects every closed neighborhood in G, and we obtain again a $(2-\varepsilon_d)^n$ bound on the number of useful sets. An important contribution of Björklund, Husfeldt, Kaski, and Koivisto [5] can be summarized as follows: using the fact that the useful sets are upward-closed (any superset of a useful set is useful as well), we can trim the fast subset convolution algorithm of [3] to consider useful sets only. Consequently, we obtain a $(2-\varepsilon_d)^n n^{\mathcal{O}(1)}$-time algorithm for computing the chromatic number of an input graph of maximum degree bounded by d.

Bounded Average Degree

Generalizing Algorithms for Bounded Degree Graphs

The above approach for traveling salesman problem has been generalized to graphs of bounded average degree by Cygan and Pilipczuk [8] using the following observation. Assume a graph G has n vertices and average degree bounded by d. Then, a simple Markov-type inequality implies that for every $\zeta > 1$ there are at most n/ζ vertices of degree larger than ζd. However, this bound

cannot be tight for all values of ζ at once, and one can prove the following: if we want at most $n/(\alpha\zeta)$ vertices of degree larger than ζd for some $\alpha > 1$, then we can always find such a constant ζ of order roughly exponential in α.

An appropriate choice of α and the corresponding value of ζ allow us to partition the vertex set of an input graph into a large part of bounded degree and a very small part of unbounded degree. The extra multiplicative gap of α in the size bound allows us to hide the cost of extensive branching on the part with unbounded degree in the gains obtained by considering only (appropriately defined) useful sets in the bounded degree part.

With this line of reasoning, Cygan and Pilipczuk [8] showed that for every degree bound d, there exists a constant $\varepsilon_d > 0$ such that traveling salesman problem in graphs of bounded average degree by d can be solved in $(2 - \varepsilon_d)^n n^{\mathcal{O}(1)}$ time. It should be noted that the constant ε_d depends here doubly exponentially on d, as opposed to single-exponential dependency in the works for bounded degree graphs.

Furthermore, Cygan and Pilipczuk showed how to express the problem of counting perfect matchings in an n-vertex graph as a specific variant of a problem of counting Hamiltonian cycles in an $n/2$-vertex graph. This reduction not only gives a simpler $2^{n/2}n^{\mathcal{O}(1)}$-time algorithm for counting perfect matchings, as compared to the original algorithm of Björklund [2], but since the reduction does not increase the number of edges in a graph, it also provides a $(2 - \varepsilon_d)^{n/2}n^{\mathcal{O}(1)}$-time algorithm in the case of bounded average degree.

In a subsequent work, Golovnev, Kulikov, and Mihajlin [9] showed how to use the aforementioned multiplicative gap of α to obtain a $(2 - \varepsilon_d)^n n^{\mathcal{O}(1)}$-time algorithm for computing the chromatic number of a graph with average degree bounded by d. Furthermore, they expressed all previous algorithms as the task of determining one coefficient in a carefully chosen polynomial, obtaining polynomial space complexity without any significant loss in time complexity.

Counting Perfect Matchings in Bipartite Graphs

A somewhat different line of research concerns counting perfect matchings in bipartite graphs. Here, a $2^{n/2}n^{\mathcal{O}(1)}$-time algorithm is known for several decades [12]. Cygan and Pilipczuk presented a very simple $2^{(1-1/(3.55d))n/2}n^{\mathcal{O}(1)}$-time algorithm for this problem in graphs of average degree at most d, improving upon the previous works of Servedio and Wan [13] and Izumi and Wadayama [11]. Furthermore, this result generalizes to the problem of computing the permanent of a matrix over an arbitrary commutative ring with the number of nonzero entries linear in the dimension of the matrix.

Cross-References

- Exact Graph Coloring Using Inclusion-Exclusion
- Fast Subset Convolution

Recommended Reading

1. Bellman R (1962) Dynamic programming treatment of the travelling salesman problem. J ACM 9:61–63
2. Björklund A (2012) Counting perfect matchings as fast as ryser. In: Rabani Y (ed) SODA, Kyoto. SIAM, pp 914–921
3. Björklund A, Husfeldt T, Kaski P, Koivisto M (2007) Fourier meets möbius: fast subset convolution. In: Johnson DS, Feige U (eds) STOC, San Diego. ACM, pp 67–74
4. Björklund A, Husfeldt T, Koivisto M (2009) Set partitioning via inclusion-exclusion. SIAM J Comput 39(2):546–563
5. Björklund A, Husfeldt T, Kaski P, Koivisto M (2010) Trimmed moebius inversion and graphs of bounded degree. Theory Comput Syst 47(3):637–654
6. Björklund A, Husfeldt T, Kaski P, Koivisto M (2012) The traveling salesman problem in bounded degree graphs. ACM Trans Algorithms 8(2):18
7. Chung FRK, Frankl P, Graham RL, Shearer JB (1986) Some intersection theorems for ordered sets and graphs. J Comb Theory Ser A 43(1):23–37
8. Cygan M, Pilipczuk M (2013) Faster exponential-time algorithms in graphs of bounded average degree. In: Fomin FV, Freivalds R, Kwiatkowska MZ, Peleg D (eds) ICALP (1). Lecture notes in computer science, vol 7965. Springer, Berlin/Heidelberg, pp 364–375

9. Golovnev A, Kulikov AS, Mihajlin I (2014) Families with infants: a general approach to solve hard partition problems. In: ICALP (1). Lecture notes in computer science. Springer, Berlin/Heidelberg, pp 551–562. Available at http://arxiv.org/abs/1311.2456
10. Held M, Karp RM (1962) A dynamic programming approach to sequencing problems. J Soc Ind Appl Math 10:196–210
11. Izumi T, Wadayama T (2012) A new direction for counting perfect matchings. In: FOCS, New Brunswick. IEEE Computer Society, pp 591–598
12. Ryser H (1963) Combinatorial mathematics. The Carus mathematical monographs. Mathematical Association of America, Buffalo
13. Servedio RA, Wan A (2005) Computing sparse permanents faster. Inf Process Lett 96(3):89–92

Exact Graph Coloring Using Inclusion-Exclusion

Andreas Björklund and Thore Husfeldt
Department of Computer Science, Lund University, Lund, Sweden

Keywords

Vertex coloring

Years and Authors of Summarized Original Work

2006; Björklund, Husfeldt

Problem Definition

A *k-coloring* of a graph $G = (V, E)$ assigns one of k colors to each vertex such that neighboring vertices have different colors. This is sometimes called *vertex coloring*.

The smallest integer k for which the graph G admits a k-coloring is denoted $\chi(G)$ and called the *chromatic number*. The number of k-colorings of G is denoted $P(G;k)$ and called the *chromatic polynomial*.

Key Results

The central observation is that $\chi(G)$ and $P(G;k)$ can be expressed by an inclusion-exclusion formula whose terms are determined by the number of independent sets of induced subgraphs of G. For $X \subseteq V$, let $s(X)$ denote the number of nonempty independent vertex subsets disjoint from X, and let $s_r(X)$ denote the number of ways to choose r nonempty independent vertex subsets $S_1, \ldots, S_r$ (possibly overlapping and with repetitions), all disjoint from X, such that $|S_1| + \cdots + |S_r| = |V|$.

Theorem 1 ([1]) *Let G be a graph on n vertices.*

1.

$$\chi(G) = \min_{k \in \{1,\ldots,n\}} \left\{ k : \sum_{X \subseteq V} (-1)^{|X|} s(X)^k > 0 \right\}.$$

2. For $k = 1, \ldots, n$,

$$P(G;k) = \sum_{r=1}^{k} \binom{k}{r} \left(\sum_{X \subseteq V} (-1)^{|X|} s_r(X) \right).$$

The time needed to evaluate these expressions is dominated by the 2^n evaluations of $s(X)$ and $s_r(X)$, respectively. These values can be precomputed in time and space within a polynomial factor of 2^n because they satisfy

$$s(X) = \begin{cases} 0, & \text{if } X = V, \\ s(X \cup \{v\}) + s(X \cup \{v\} \cup N(v)) + 1, & \text{for } v \notin X, \end{cases}$$

where $N(v)$ are the neighbors of v in G. Alternatively, the values can be computed using exponential-time, polynomial-space algorithms from the literature.

This leads to the following bounds:

Theorem 2 ([3]) *For a graph G on n vertices, $\chi(G)$ and $P(G;k)$ can be computed in*

1. *Time and space $2^n n^{O(1)}$.*
2. *Time $O(2.2461^n)$ and polynomial space*

The space requirement can be reduced to $O(1.292^n)$ *[4].*

The techniques generalize to arbitrary families of subsets over a universe of size n, provided membership in the family can be decided in polynomial time [3, 4], and to the Tutte polynomial and the Potts model [2].

Applications

In addition to being a fundamental problem in combinatorial optimization, graph coloring also arises in many applications, including register allocation and scheduling.

Recommended Reading

1. Björklund A, Husfeldt T (2008) Exact algorithms for exact satisfiability and number of perfect matchings. Algorithmica 52(2):226–249
2. Björklund A, Husfeldt T, Kaski P, Koivisto M (2007) Fourier meets Möbius: fast subset convolution. In: Proceedings of the 39th annual ACM symposium on theory of computing (STOC), San Diego, 11–13 June 2007. Association for Computing Machinery, New York, pp 67–74
3. Björklund A, Husfeldt T, Koivisto M (2009) Set partitioning via inclusion-exclusion. SIAM J Comput 39(2):546–563
4. Björklund A, Husfeldt T, Kaski P, Koivisto M (2011) Covering and packing in linear space. Inf Process Lett 111(21–22):1033–1036

Exact Quantum Algorithms

Ashley Montanaro
Department of Computer Science, University of Bristol, Bristol, UK

Keywords

Exact algorithms; Quantum algorithms; Quantum query complexity

Years and Authors of Summarized Original Work

2013; Ambainis

Problem Definition

Many of the most important known quantum algorithms operate in the query complexity model. In the simplest variant of this model, the goal is to compute some Boolean function of n input bits by making the minimal number of queries to the bits. All other resources (such as time and space) are considered to be free. In the model of *exact* quantum query complexity, one insists that the algorithm succeeds with certainty on every allowed input. The aim is then to find quantum algorithms which satisfy this constraint and still outperform any possible classical algorithm. This can be a challenging task, as achieving a probability of error equal to zero requires delicate cancellations between the amplitudes in the quantum algorithm. Nevertheless, efficient exact quantum algorithms are now known for certain functions.

Some basic Boolean functions which we will consider below are:

- Parity^n: $f(x_1, \ldots, x_n) = x_1 \oplus x_2 \oplus \cdots \oplus x_n$.
- Threshold^n_k: $f(x_1, \ldots, x_n) = 1$ if $|x| \geq k$, and $f(x) = 0$ otherwise, where $|x| := \sum_i x_i$

is the Hamming weight of x. The special case $k = n/2$ is called the majority function.
- Exact_k^n: $f(x_1, \ldots, x_n) = 1$ if $|x| = k$, and $f(x) = 0$ otherwise.
- NE ("not-all-equal") on 3 bits: $f(x_1, x_2, x_3) = 0$ if $x_1 = x_2 = x_3$, and $f(x_1, x_2, x_3) = 1$ otherwise.

Key Results

Early Results

One of the earliest results in quantum computation was that the parity of 2 bits can be computed with certainty using only 1 quantum query [6], implying that Parity^n can be computed using $\lceil n/2 \rceil$ quantum queries. By contrast, any classical algorithm which computes this function must make n queries. The quantum algorithm for Parity^n can be used as a subroutine to obtain speedups over classical computation for other problems. For example, based on this algorithm the majority function on n bits can be computed exactly using $n + 1 - w(n)$ quantum queries, where $w(n)$ is the number of 1s in the binary expansion of n [8]; this result has recently been improved (see below).

If the function to be computed is partial, i.e., some possible inputs are disallowed, the separation between exact quantum and classical query complexity can be exponential. For example, in the Deutsch-Jozsa problem we are given query access to an n-bit string x (with n even) such that either all the bits of x are equal or exactly half of them are equal to 1. Our task is to determine which is the case. Any exact classical algorithm must make at least $n/2 + 1$ queries to bits of x to solve this problem, but it can be solved with only one quantum query [7]. An exponential separation is even known between exact quantum and *bounded-error* classical query complexity for a different partial function [5].

Recent Developments

For some years, the best known separation between exact quantum and classical query complexity of a total Boolean function (i.e., a function $f : \{0,1\}^n \to \{0,1\}$ with all possible n-bit strings allowed as input) was the factor of 2 discussed above. However, recently the first example has been presented of an exact quantum algorithm for a family of total Boolean functions which achieves a lower asymptotic query complexity than the best possible classical algorithm [1].

The family of functions used can be summarized as a "not-all-equal tree of depth d." It is based around the recursive use of the NE function. Define the function $\text{NE}^0(x_1) = x_1$ and then for $d > 0$

$$\text{NE}^d(x_1, \ldots, x_{3^d}) = \text{NE}(\text{NE}^{d-1}(x_1, \ldots, x_{3^{d-1}}), \text{NE}^{d-1}(x_{3^{d-1}+1}, \ldots, x_{2\cdot 3^{d-1}}), \text{NE}^{d-1}(x_{2\cdot 3^{d-1}+1}, \ldots, x_{3^d})).$$

Then the following separation is known:

Theorem 1 (Ambainis [1]) *There is an exact quantum algorithm which computes* NE^d *using* $O(2.593\ldots^d)$ *queries. Any classical algorithm which computes* NE^d *must make* $\Omega(3^d)$ *queries, even if it is allowed probability of failure* $1/3$.

In addition, Theorem 1 implies the first known asymptotic separation between exact quantum and classical communication complexity for a total function. Improvements over the best possible classical algorithms are also known for the other basic Boolean functions previously mentioned.

Theorem 2 (Ambainis, Iraids, and Smotrovs [2]) *There is an exact quantum algorithm which computes* Exact_k^n *using* $\max\{k, n-k\}$ *queries and an exact quantum algorithm which computes* Threshold_k^n *using* $\max\{k, n-k+1\}$ *queries. Both of these complexities are optimal.*

By contrast, it is easy to see that any exact classical algorithm for these functions must make

n queries. An optimal exact quantum algorithm for the special case Exact_2^4 had already been found prior to this, in work which also gave optimal exact quantum query algorithms for all Boolean functions on up to 3 bits [9].

Methods

We briefly describe the main ingredients of the efficient quantum algorithm for NE^d [1]. The basic idea is to fix some small d_0, start with an exact quantum algorithm which computes NE^{d_0} using fewer queries than the best possible classical algorithm, and then amplify the separation by using the algorithm recursively. A difficulty with this approach is that the standard approach for using a quantum algorithm recursively incurs a factor of 2 penalty in the number of queries with each recursive call. This factor of 2 is required to "uncompute" information left over after the algorithm has completed. Therefore, a query complexity separation by a factor of 2 or less does not immediately give an asymptotic separation.

This problem can be addressed by introducing the notion of p-computation. Let $p \in [-1, 1]$. A quantum algorithm $\mathcal{A}$ is said to p-compute a function $f(x_1, \ldots, x_n)$ if, for some state $|\psi_{\text{start}}\rangle$:

- Whenever $f(x_1, \ldots, x_n) = 0$, $\mathcal{A}|\psi_{\text{start}}\rangle = |\psi_{\text{start}}\rangle$.
- Whenever $f(x_1, \ldots, x_n) = 1$, $\mathcal{A}|\psi_{\text{start}}\rangle = p|\psi_{\text{start}}\rangle + \sqrt{1-p^2}|\psi\rangle$ for some $|\psi\rangle$, which may depend on x, such that $\langle\psi|\psi_{\text{start}}\rangle = 0$.

It can be shown that if there exists an algorithm which p-computes some function f for some $p \leq 0$, there exists an exact quantum algorithm which computes f using the same number of queries. Further, if an algorithm (-1)-computes some function f, the same algorithm can immediately be used recursively, without needing any additional queries at each level of recursion. Thus, to obtain an asymptotic quantum-classical separation for NE^d, it suffices to obtain an algorithm which (-1)-computes NE^{d_0} using strictly fewer than 3^{d_0} queries, for some d_0.

The NE^d problem also behaves particularly well with respect to p-computation for general values of p:

Lemma 1 *If there is an algorithm $\mathcal{A}$ which p-computes NE^{d-1} using k queries, there is an algorithm $\mathcal{A}'$ which p'-computes NE^d with $2k$ queries, for $p' = 1 - 4(1-p)^2/9$.*

This lemma allows algorithms for NE^{d-1} to be lifted to algorithms for NE^d, at the expense of making the value of p worse. Nevertheless, given that it is easy to write down an algorithm which (-1)-computes NE^0 using one query, the lemma is sufficient to obtain an exact quantum algorithm for NE^2 using 4 queries. This is already enough to prove an asymptotic quantum-classical separation, but this separation can be improved using the following lemma (a corollary of a variant of amplitude amplification):

Lemma 2 *If there is an algorithm $\mathcal{A}$ which p-computes NE^d using k queries, there is an algorithm $\mathcal{A}'$ which p'-computes NE^d with $2k$ queries, for $p' = 2p^2 - 1$.*

Interleaving Lemmas 1 and 2 allows one to derive an algorithm which (-1)-computes NE^8 using 2,048 queries, which implies an exact quantum algorithm for NE^d using $O(2{,}048^{d/8}) = O(2.593\ldots^d)$ queries.

Experimental Results

It is a difficult task to design exact quantum query algorithms, even for small functions, as these algorithms require precise cancellations between amplitudes. One way to gain numerical evidence for what the exact quantum query complexity of a function should be is to use the formulation of quantum query complexity as a semidefinite programming (SDP) problem [4]. This allows one to estimate the optimal success probability of any quantum algorithm using a given number of queries to compute a given function. If this success probability is very close to 1, this gives numerical evidence that there exists an exact quantum algorithm using that number of queries.

This approach has been applied for all Boolean functions on up to 4 bits, giving strong evidence that the only function on 4 bits which requires 4 quantum queries is the AND function and functions equivalent to it [9]. This has led to the conjecture that, for any n, the only function on n bits which requires n quantum queries to be computed exactly is the AND function and functions equivalent to it. This would be an interesting contrast with the classical case where most functions on n bits require n queries. This conjecture has recently been proven for various special cases: symmetric functions, monotone functions, and functions with formula size n [3].

Cross-References

▸ Quantum Algorithm for the Parity Problem
▸ Quantum Search

Recommended Reading

1. Ambainis A (2013) Superlinear advantage for exact quantum algorithms. In: Proceedings of the 45th annual ACM symposium on theory of computing, pp 891–900. arXiv:1211.0721
2. Ambainis A, Iraids J, Smotrovs J (2013) Exact quantum query complexity of EXACT and THRESHOLD. In: Proceedings of the 8th conference on the theory of quantum computation, communication, and cryptography (TQC'13), pp 263–269. arXiv:1302.1235
3. Ambainis A, Gruska J, Zheng S (2014) Exact query complexity of some special classes of Boolean functions. arXiv:1404.1684
4. Barnum H, Saks M, Szegedy M (2003) Quantum query complexity and semi-definite programming. In: Proceedings of the 18th annual IEEE conference on computational complexity, Aarhus, pp 179–193
5. Brassard G, Høyer P (1997) An exact quantum polynomial-time algorithm for Simon's problem. In: Proceedings of the fifth Israeli symposium on theory of computing and systems, Aarhus, Denmark pp 12–23. quant-ph/9704027
6. Cleve R, Ekert A, Macchiavello C, Mosca M (1998) Quantum algorithms revisited. Proc R Soc Lond A 454(1969):339–354. quant-ph/9708016
7. Deutsch D, Jozsa R (1992) Rapid solution of problems by quantum computation. Proc R Soc Lond Ser A 439(1907):553–558
8. Hayes T, Kutin S, van Melkebeek D (2002) The quantum black-box complexity of majority. Algorithmica 34(4):480–501. quant-ph/0109101
9. Montanaro A, Jozsa R, Mitchison G (2013) On exact quantum query complexity. Algorithmica 71(4):775–796

Experimental Implementation of Tile Assembly

Constantine G. Evans
Division of Biology and Bioengineering, California Institute of Technology, Pasadena, CA, USA

Keywords

DNA tiles; Experimental tile self-assembly

Years and Authors of Summarized Original Work

2007; Schulman, Winfree
2008; Fujibayashi, Hariadi, Park, Winfree, Murata
2009; Barish, Schulman, Rothemund, Winfree
2012; Schulman, Yurke, Winfree

Problem Definition

From the earliest works on tile self-assembly, abstract theoretical models and experimental implementations have been linked. In 1998, in addition to developing the abstract and kinetic Tile Assembly Models (aTAM and kTAM) [14], Winfree et al. demonstrated the use of DNA tiles to construct a simple, periodic lattice [16]. Periodic lattices and "uniquely addressed" assemblies, where each tile type appears once in each assembly, have been widely studied, with systems employing up to a thousand unique tiles in three dimensions [8, 13]. While these systems provide insight into the behavior of DNA tile systems, *algorithmic* tile systems of more theoretical interest

pose specific challenges for experimental implementation.

In the aTAM, abstract tiles attach individually to empty lattice sites if bonds of a sufficient total strength b (at least abstract "temperature" τ) can be made, and once attached, never detach. Experimentally, free tiles and assemblies of bound tiles are in solution. Tiles have short single-stranded "sticky ends" regions that form bonds with complementary regions on other tiles. Tiles attach to assemblies at rates dependent only upon their concentrations, regardless of the strength of bonds that can be made. Once attached, tiles can detach and do so at a rate that is exponentially dependent upon the total strength of the bonds [6]. Thus, for a tile t_i with concentration $[t_i]$ binding by a total abstract bond strength b, we have attachment and detachment rates of

$$r_f = k_f[t_i] \qquad r_b = k_f e^{-b\Delta G^\circ_{se}/RT+\alpha} \tag{1}$$

where k_f is an experimentally determined rate constant, α is a constant binding free energy change (e.g., from entropic considerations), ΔG°_{se} is the free energy change of a single-strength bond, and T is the (physical) temperature. Using the substitutions $[t_i] = e^{-G_{mc}+\alpha}$, $G_{se} = -\Delta G^\circ_{se}/RT$, and $\hat{k}_f = k_f e^{\alpha}$, these can be simplified to

$$r_f = \hat{k}_f e^{-G_{mc}} \qquad r_b = \hat{k}_f e^{-bG_{se}} \tag{2}$$

where G_{se} is a (positive) unitless free energy for a single-strength bond (larger values correspond to stronger bonds), G_{mc} is a free energy analogue of concentration (larger values correspond to lower concentrations), and $\hat{k}_f$ is an adjusted rate constant.

These rates are the basis of the kinetic Tile Assembly Model (kTAM), which is widely used as a physical model of tile assembly [14]. Tiles that attach faster than they detach will tend to remain attached and allow further growth: for example, if $G_{mc} < 2G_{se}$, tile attachments by $b \geq 2$ will be favorable. Tiles that detach faster than they attach will tend to remain detached and not allow further growth. Since G_{mc} is dependent upon tile concentration, and G_{se} is dependent upon physical temperature (lower temperatures result in larger G_{se} values), the attachment and detachment rates can be tuned such that attachment is slightly more favorable than detachment for tiles attaching by a certain total bond strength and less favorable for less strongly bound tiles. In this way, in the limit of low concentrations and slow growth, the kTAM approximates the aTAM at a given abstract temperature τ. When moving away from this limit and toward experimentally feasible conditions, however, the kTAM provides insight into many of the challenges faced in experimental implementation of algorithmic tile assembly:

Growth errors: While tile assembly in the aTAM is error-free, tiles can attach in erroneous locations in experiments. Even ignoring the possibility of lattice defects, malformed tiles, and other experimental peculiarities, errors can arise in the kTAM via tiles that attach by less than the required bond strength (e.g., one single-strength bond for a $\tau = 2$ system) and are then "frozen" in place by further attachments [4]. As the further growth of algorithmic systems depends on the tiles already present in an assembly, a single erroneously incorporated tile can propagate undesired growth via further, valid attachments. These errors can arise both in growth sites where another tile could attach correctly ("growth errors") and lattice sites where no correct tile could attach ("facet nucleation errors") [3, 14].

Seeding: Tile assembly in the aTAM is usually initiated from a designated "seed" tile. In solution, however, tiles are free to attach to all other tiles and can form assemblies without starting from a seed, even if this requires several unfavorable attachments to form a stable structure that can allow further growth. Depending upon the tile system, these "spuriously nucleated" structures can potentially form easily. For example, a $T = 2$ system with boundaries of identical tiles that attach by double bonds on both sides can readily form long strings of boundary tiles [10, 11].

Tile depletion: As free tiles in solution are incorporated into assemblies, their concentrations are

correspondingly reduced. This depletion lowers the attachment rates for those tiles and in turn changes the favorability of growth. If different tile types are incorporated in different quantities, their attachment rates will become unequal, and at some point in assembly, attachment by two single-strength bonds may be favorable for one tile type and unfavorable for another.

Tile design: While theoretical constructions may employ an arbitrary number of sticky ends types, this number is limited by tile designs in practice. Most tiles use short single-stranded DNA regions of 5–10 nucleotides (nt), limiting the number of possible sticky ends to 4^5–4^{10} at best. However, since partial bonds can form between subsequences of the sticky ends, sequences with sufficient orthogonality are required, and since DNA binding strength is sequence dependent, sequences with similar binding energies are required [5]. Both of these effects place considerably more stringent limits on the number of sticky ends and change the behavior of experimental systems.

Key Results

Winfree and Bekbolatov developed a tileset transformation, "uniform proofreading," that reduced per-site *growth* error rates from $r_{\text{err}} \approx me^{-G_{se}}$ (where m is the number of possible errors) to $\approx me^{-KG_{se}}$ by scaling each tile into a $K \times K$ block of individually attaching tiles with unique internal bonds [15]. However, this transformation did not reduce facet nucleation errors. Chen and Goel later created a modified transformation, "snaked proofreading," that reduced both growth and facet nucleation errors by changing the strengths of the internal bonds used [3]. These and other proofreading methods have the potential to drastically reduce error rates in experimental systems.

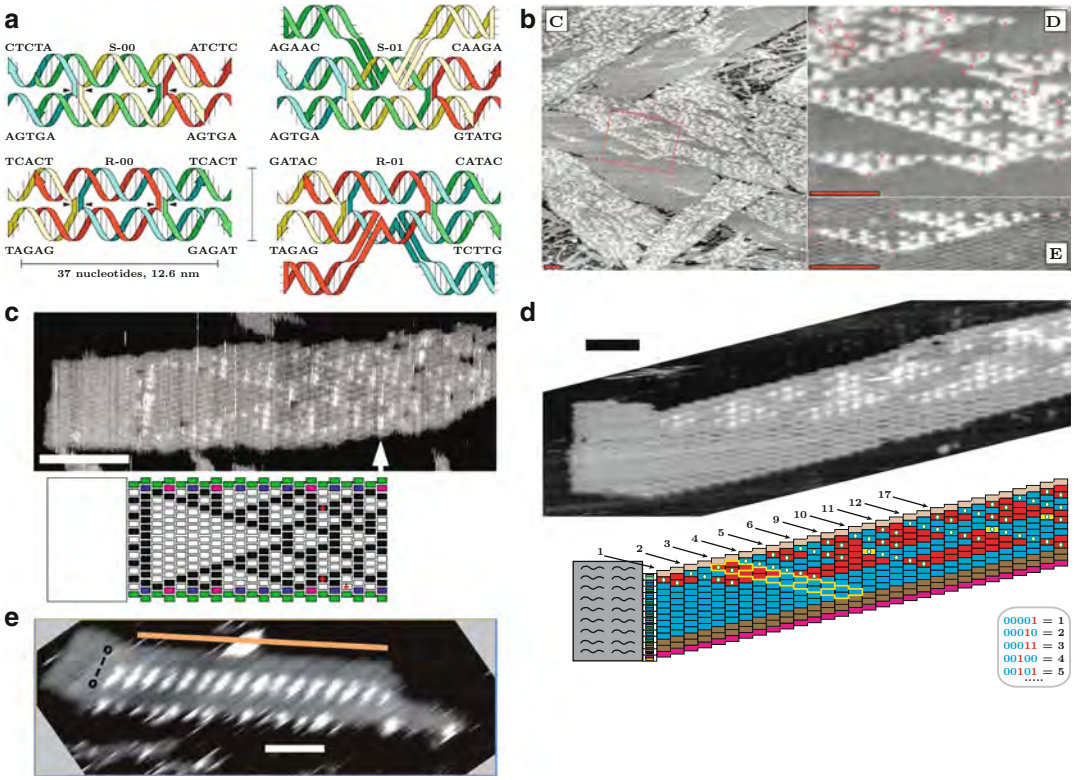
Schulman et al. analyzed tile system nucleation through the consideration of "critical nuclei," tile assemblies where melting and further growth are equally favorable, and showed that by ensuring a sufficient number of unfavorable attachments would be required for a critical nucleus to form, the rate of spurious nucleation can be kept arbitrarily low [11]. Using this analysis, Schulman et al. constructed the "zigzag" ribbon system, which forms a ribbon where each row must assemble completely before the next can begin growth, as an example of a system where spurious nucleation can be made arbitrarily low by increasing ribbon width. To nucleate desired structures, this system makes use of a large, preformed seed structure to allow the growth of the first ribbon row.

Schulman et al. also devised a "constant-temperature" growth technique where the concentrations of assemblies, controlled by the concentration of initial seeds in a nucleation-controlled system, are kept small enough in comparison to the concentrations of free tiles that growth does not significantly deplete tile concentrations, which thus remain approximately constant [12]. After growth is completed, the remaining free tiles are "deactivated" by adding an excess of DNA strands complementary to specific sticky ends sequences.

In analyzing the effects of DNA sequences on tile assembly, Evans and Winfree showed an exponential increase of error rates in the kTAM for partial binding between different sticky ends sequences and for differing sequence-dependent binding energies and developed algorithms for sequence design and assignment to reduce these effects [5]. With reasonable design constraints, their algorithms suggested limits of around 80 sticky ends types for tiles using 5 nt sticky ends and around 360 for tiles using 10 nt sticky ends before significant sequence effects begin to become unavoidable and must be incorporated into tile system design.

Experimental Results

While numerous designs exist for tile structures, experimental implementations have usually used either double-crossover (DX) tiles with 5 or 6 nt sticky ends [16] or single-stranded tiles (SST) with 10 and 11 nt sticky ends [17]. SSTs potentially offer a significantly larger sequence



Experimental Implementation of Tile Assembly, Fig. 1 Experimental results for algorithmic tile assembly. (**a**) and (**b**) show the Rothemund et al. XOR system's DX tiles and resulting structures, with (b) illustrating the high error rates and seeding problems of the system [9]. (**c**) shows the Fujibayashi et al. fixed-width XOR ribbon [7], while (**d**) shows the Barish et al. binary counter ribbon with partial 2×2 proofreading [2]; the rectangular structures on the left of both systems are preformed DNA origami seeds. (**e**) shows an example bit-copying ribbon from Schulman et al. [12]

space and have been employed in large, non-algorithmic systems [8, 13] but have not yet been used for complex algorithmic systems.

Early experiments in algorithmic tile assembly using DX tiles did not employ any of the key results discussed above. Rothemund et al. implemented a simple XOR system of four logical tiles (eight tiles were needed owing to structural considerations), using DNA hairpins on "one-valued" tiles as labels [9] and flexible, one-dimensional seeds (Fig. 1a,b). While assemblies grew, and Sierpinski triangle patterns were visible, error rates were between 1 and 10 % per tile. Barish et al. implemented more complex bit-copying and binary counting systems in a similar way, finding per-tile error rates of around 10 % [1].

More recently, Fujibayashi et al. used rigid DNA origami structures to serve as seeds for the growth of a fixed-width XOR ribbon system and, in doing so, reduced error rates to 1.4 % per tile without incorporating proofreading [7] (Fig. 1c). This seeding mechanism was also used by Barish et al. to seed zigzag bit-copying and binary counting ribbon systems that implemented 2×2 uniform proofreading [2]. With nucleation control and proofreading, these systems resulted in dramatically reduced error rates of 0.26 % per proofreading block for copying and 4.1 % for the more algorithmically complex binary counting, which only partially implemented uniform proofreading (Fig. 1d).

A similar bit-copying ribbon was later implemented by Schulman et al., with the

addition of the constant-temperature, constant-concentration growth method and the use of biotin-streptavidin labels rather than DNA hairpins. The result was a decrease in error rates by almost a factor of ten to 0.034 % per block [12] (Fig. 1e). At this error rate, structures of around 2,500 error-free blocks, or 10,000 individual tiles, could be grown with reasonable yields, suggesting that with the incorporation of proofreading, nucleation control and constant-concentration growth methods, low-error experimental implementations of increasingly complex algorithmic tile systems may be feasible up to sequence space limitations.

Cross-References

▸ Robustness in Self-Assembly

Recommended Reading

1. Barish RD, Rothemund PWK, Winfree E (2005) Two computational primitives for algorithmic self-assembly: copying and counting. Nano Lett 5(12):2586–2592. doi:10.1021/nl052038l
2. Barish RD, Schulman R, Rothemund PWK, Winfree E (2009) An information-bearing seed for nucleating algorithmic self-assembly. PNAS 106:6054–6059. doi:10.1073/pnas.0808736106
3. Chen HL, Goel A (2005) Error free self-assembly using error prone tiles. In: DNA 10, Milan. LNCS, vol 3384. Springer, pp 702–707
4. Doty D (2012) Theory of algorithmic self-assembly. Commun ACM 55(12):78–88. doi:10.1145/2380656.2380675
5. Evans CG, Winfree E (2013) DNA sticky end design and assignment for robust algorithmic self-assembly. In: DNA 19, Tempe. LNCS, vol 8141. Springer, pp 61–75. doi:10.1007/978-3-319-01928-4_5
6. Evans CG, Hariadi RF, Winfree E (2012) Direct atomic force microscopy observation of DNA tile crystal growth at the single-molecule level. J Am Chem Soc 134:10,485–10,492. doi:10.1021/ja301026z
7. Fujibayashi K, Hariadi R, Park SH, Winfree E, Murata S (2008) Toward reliable algorithmic self-assembly of DNA tiles: a fixed-width cellular automaton pattern. Nano Lett 8(7):1791–1797. doi:10.1021/nl0722830
8. Ke Y, Ong LL, Shih WM, Yin P (2012) Three-dimensional structures self-assembled from DNA bricks. Science 338(6111):1177–1183. doi:10.1126/science.1227268
9. Rothemund PWK, Papadakis N, Winfree E (2004) Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol 2(12):e424. doi:10.1371/journal.pbio.0020424
10. Schulman R, Winfree E (2007) Synthesis of crystals with a programmable kinetic barrier to nucleation. PNAS 104(39):15,236–15,241. doi:10.1073/pnas.0701467104
11. Schulman R, Winfree E (2010) Programmable control of nucleation for algorithmic self-assembly. SIAM J Comput 39(4):1581–1616. doi:10.1137/070680266
12. Schulman R, Yurke B, Winfree E (2012) Robust self-replication of combinatorial information via crystal growth and scission. PNAS 109(17):6405–6410. doi:10.1073/pnas.1117813109
13. Wei B, Dai M, Yin P (2012) Complex shapes self-assembled from single-stranded DNA tiles. Nature 485(7400):623–626
14. Winfree E (1998) Simulations of computing by self-assembly. Technical report CaltechCSTR:1998.22, Pasadena
15. Winfree E, Bekbolatov R (2004) Proofreading tile sets: error correction for algorithmic self-assembly. In: DNA 9, Madison. Wisconson, LNCS, vol 2943. Springer, pp 126–144
16. Winfree E, Liu F, Wenzler LA, Seeman NC (1998) Design and self-assembly of two-dimensional DNA crystals. Nature 394(6693):539–544
17. Yin P, Hariadi RF, Sahu S, Choi HMT, Park SH, LaBean TH, Reif JH (2008) Programming DNA tube circumferences. Science 321(5890):824–826. doi:10.1126/science.1157312

Experimental Methods for Algorithm Analysis

Catherine C. McGeoch
Department of Mathematics and Computer Science, Amherst College, Amherst, MA, USA

Keywords

Algorithm engineering; Empirical algorithmics; Empirical analysis of algorithms; Experimental algorithmics

Years and Authors of Summarized Original Work

2001; McGeoch

Problem Definition

Experimental analysis of algorithms describes not a specific algorithmic problem, but rather an approach to algorithm design and analysis. It complements, and forms a bridge between, traditional *theoretical analysis*, and the application-driven methodology used in *empirical analysis*.

The traditional theoretical approach to algorithm analysis defines algorithm efficiency in terms of counts of dominant operations, under some abstract model of computation such as a RAM; the input model is typically either worst-case or average-case. Theoretical results are usually expressed in terms of asymptotic bounds on the function relating input size to number of dominant operations performed.

This contrasts with the tradition of empirical analysis that has developed primarily in fields such as operations research, scientific computing, and artificial intelligence. In this tradition, the efficiency of implemented programs is typically evaluated according to CPU or wall-clock times; inputs are drawn from real-world applications or collections of benchmark test sets, and experimental results are usually expressed in comparative terms using tables and charts.

Experimental analysis of algorithms spans these two approaches by combining the sensibilities of the theoretician with the tools of the empiricist. Algorithm and program performance can be measured experimentally according to a wide variety of *performance indicators*, including the dominant cost traditional to theory, bottleneck operations that tend to dominate running time, data structure updates, instruction counts, and memory access costs. A researcher in experimental analysis selects performance indicators most appropriate to the scale and scope of the specific research question at hand. (Of course time is not the only metric of interest in algorithm studies; this approach can be used to analyze other properties such as solution quality or space use.)

Input instances for experimental algorithm analysis may be randomly generated or derived from application instances. In either case, they typically are described in terms of a small- to medium-sized collection of *controlled parameters*. A primary goal of experimentation is to investigate the cause-and-effect relationship between input parameters and algorithm/program performance indicators.

Research goals of experimental algorithmics may include discovering functions (not necessarily asymptotic) that describe the relationship between input and performance, assessing the strengths and weaknesses of different algorithm/data structures/programming strategies, and finding best algorithmic strategies for different input categories. Results are typically presented and illustrated with graphs showing comparisons and trends discovered in the data.

The two terms "empirical" and "experimental", are often used interchangeably in the literature. Sometimes the terms "old style" and "new style" are used to describe, respectively, the empirical and experimental approaches to this type of research. The related term "algorithm engineering" refers to a systematic design process that takes an abstract algorithm all the way to an implemented program, with an emphasis on program efficiency. Experimental and empirical analysis is often used to guide the algorithm engineering process. The general term *algorithmics* can refer to both design and analysis in algorithm research.

Key Results

None

Applications

Experimental analysis of algorithms has been used to investigate research problems originating in theoretical computer science. One example arises in the average-case analysis of algorithms for the One-Dimensional Bin Packing problem. Experimental analyses have led to new theorems about the performance of the optimal algorithm; new asymptotic bounds on average-case performance of approximation algorithms; extensions

of theoretical results to new models of inputs; and to new algorithms with tighter approximation guarantees. Another example is the experimental discovery of a type of phase-transition behavior for random instances of the 3CNF-Satisfiabilty problem, which has led to new ways to characterize the difficulty of problem instances.

A second application of experimental algorithmics is to find more realistic models of computation, and to design new algorithms that perform better on these models. One example is found in the development of new memory-based models of computation that give more accurate time predictions than traditional unit-cost models. Using these models, researchers have found new cache-efficient and I/O-efficient algorithms that exploit properties of the memory hierarchy to achieve significant reductions in running time.

Experimental analysis is also used to design and select algorithms that work best in practice, algorithms that work best on specific categories of inputs, and algorithms that are most robust with respect to bad inputs.

Data Sets

Many repositories for data sets and instance generators to support experimental research are available on the Internet. They are usually organized according to specific combinatorial problems or classes of problems.

URL to Code

Many code repositories to support experimental research are available on the Internet. They are usually organized according to specific combinatorial problems or classes of problems. Skiena's *Stony Brook Algorithm Repository* (www.cs.sunysb.edu/~algorith/) provides a comprehensive collection of problem definitions and algorithm descriptions, with numerous links to implemented algorithms.

Recommended Reading

The algorithmic literature containing examples of experimental research is much too large to list here. Some articles containing advice and commentary on experimental methodology in the context of algorithm research appear in the list below.

The workshops and journals listed below are specifically intended to support research in experimental analysis of algorithms. Experimental work also appears in more general algorithm research venues such as SODA (ACM/IEEE Symposium on Data Structures and Algorithms), *Algorithmica*, and *ACM Transactions on Algorithms*.

1. ACM Journal of Experimental Algorithmics. Launched in 1996, this journal publishes contributed articles as well as special sections containing selected papers from ALENEX and WEA. Visit www.jea.acm.org, or visit portal.acm.org and click on ACM Digital Library/Journals/Journal of Experimental Algorithmics
2. ALENEX. Beginning in 1999, the annual workshop on Algorithm Engineering and Experimentation is sponsored by SIAM and ACM. It is co-located with SODA, the SIAM Symposium on Data Structures and Algorithms. Workshop proceedings are published in the Springer LNCS series. Visit www.siam.org/meetings/ for more information
3. Barr RS, Golden BL, Kelly JP, Resende MGC, Stewart WR (1995) Designing and reporting on computational experiments with heuristic methods. J Heuristics 1(1):9–32
4. Cohen PR (1995) Empirical methods for artificial intelligence. MIT, Cambridge
5. DIMACS Implementation Challenges. Each DIMACS Implementation Challenge is a year-long cooperative research event in which researchers cooperate to find the most efficient algorithms and strategies for selected algorithmic problems. The DIMACS Challenges since 1991 have targeted a variety of optimization problems on graphs; advanced data structures; and scientific application areas involving computational biology and parallel computation. The DIMACS Challenge proceedings are published by AMS as part of the DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Visit dimacs.rutgers.edu/Challenges for more information
6. Johnson DS (2002) A theoretician's guide to the experimental analysis of algorithms. In: Goodrich MH, Johnson DS, McGeoch CC (eds) Data structures, near neighbors searches, and methodology: fifth and sixth DIMACS implementation challenges, vol 59, DIMACS series in discrete mathematics and theoretical

computer science. American Mathematical Society, Providence
7. McGeoch CC (1996) Toward an experimental method for algorithm simulation. INFORMS J Comput 1(1):1–15
8. WEA. Beginning in 2001, the annual Workshop on Experimental and Efficient Algorithms is sponsored by EATCS. Workshop proceedings are published in the Springer LNCS series

Exponential Lower Bounds for *k*-SAT Algorithms

Dominik Scheder
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
Institute for Computer Science, Shanghai Jiaotong University, Shanghai, China

Keywords

Exponential algorithms; k-SAT; Lower bounds; PPSZ algorithm; Proof complexity

Years and Authors of Summarized Original Work

2013; Shiteng Chen, Dominik Scheder, Navid Talebanfard, Bangsheng Tang

Problem Definition

Given a propositional formula in conjunctive normal form, such as $(x \vee y) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{z})$, one wants to find an assignment of truth values to the variables that makes the formula evaluate to true. Here, $[x \mapsto 1, y \mapsto 0, z \mapsto 0]$ does the job. We call such formulas *CNF formulas* and such assignments *satisfying assignments*. SAT is the problem of deciding whether a given CNF formula is satisfiable. If every clause (such as $(\bar{x} \vee \bar{y} \vee z)$ above) has at most k literals, we call this a k-CNF formula. The above example is a 3-CNF formula. The problem of deciding whether a given k-CNF formula is satisfiable is called k-SAT. This is one of the most fundamental NP-complete problems.

Several clever algorithms have been developed for k-SAT. In this note we are mostly concerned with the PPSZ algorithm [3]. This is itself an improved version of the older PPZ algorithm [4]. Another prominent SAT algorithm is Schöning's random walk algorithm [6], which is slower than PPSZ, but has the benefit that it can be turned into a deterministic algorithm [5].

Given that we currently cannot prove P $\neq$ NP, all super-polynomial lower bounds on the running time of k-SAT algorithms must be either conditional, that is, rest on widely believed but yet unproven assumptions, or must be for a particular family of algorithms. In this note we sketch exponential lower bounds for the PPSZ algorithm, which is the currently fastest algorithm for k-SAT. We measure the running time of a SAT algorithm in terms of n, the number of variables. Often probabilistic algorithms for k-SAT (like PPSZ) have polynomial running time and success probability p^n for some $p < 1$. One can turn this into a Monte Carlo algorithm with success probability at least $1/2$ by repeating it $(1/p)^n$ times. We prefer the formulation of PPSZ as having polynomial running time, and we are interested in the worst-case success probability p^n.

Key Results

The worst-case behavior of PPSZ is exponential. That is, there are satisfiable k-CNF formulas on n variables, for which PPSZ finds a satisfying assignment with probability at most $2^{-\Omega(n)}$. More precisely, there is a constant C and a sequence $\epsilon_k \leq \frac{C \log^2 k}{k}$ such that the worst-case success probability of PPSZ for k-SAT is at most $2^{-(1-\epsilon_k)n}$. See Theorem 3 below for a formal statement.

The PPSZ Algorithm

The PPSZ algorithm, named after its inventors Paturi, Pudlák, Saks, and Zane [3], is the fastest

known algorithm for k-SAT. We now give a brief description of it: Choose a random ordering σ of the n variables $x_1, \ldots, x_n$ of F. Choose random truth values $b = (b_1, \ldots, b_n) \in \{0,1\}$. Iterate through the variables in the ordering given by σ. When processing x_i check whether it is "obvious" what the correct value of x_i should be. If so, fix x_i to that value. Otherwise, fix x_i to b_i. By fixing we mean replacing each occurrence of x_i in F by that value (and each occurrence of $\bar{x}_i$ by the negation of that value). After all variables have been processed, the algorithm returns the satisfying assignment it has found or returns `failure` if it has run into a contradiction.

It remains to specify what "obvious" means: Given a CNF formula F and a variable x_i, we say that the correct value of x_i is *obviously* b if the statement $x_i = b$ can be derived from F by width-w resolution, where w is some large constant (think of $w = 1{,}000$). This can be checked in time $O(n^w)$, which is polynomial.

Let $\mathrm{ppsz}(F, \sigma, b)$ be the return value of ppsz. That is, $\mathrm{ppsz}(F, \sigma, b) \in \mathrm{sat}(F) \cup \{\texttt{failure}\}$, where $\mathrm{sat}(F)$ is the set of satisfying assignments of F.

A Very Brief Sketch of the Analysis of PPSZ

Let σ be a permutation of $x_1, \ldots, x_n$ and let $b = (b_1, \ldots, b_n) \in \{0,1\}^n$. Suppose we run PPSZ on F using this permutation σ and the truth values b. For $1 \le i \le n$, define Z_i to be 1 if PPSZ did not find it obvious what the correct value of x_i should be. Let $Z = Z_1 + \cdots + Z_n$. To underline the dependence on F, σ, and b, we sometimes write $Z(F, \sigma, b)$. It is not difficult to show the following lemma.

Lemma 1 ([3]) *Let F be a satisfiable CNF formula over n variables. Let σ be a random permutation of its variables and let $a \in \{0,1\}^n$ be satisfying assignment of F. Then*

$$\Pr_{\sigma,b}[\mathrm{ppsz}(F, \sigma, b) = a] = \mathbb{E}_\sigma[2^{-Z(F,\sigma,a)}] \quad (1)$$

Since $x \mapsto 2^x$ is a convex function, Jensen's inequality implies that $\mathbb{E}_\sigma[2^{-Z}] \ge 2^{-\mathbb{E}[Z]}$, and by linearity of expectation, it holds that $\mathbb{E}[Z] = \sum_{i=1}^n \mathbb{E}[Z_i]$.

Lemma 2 ([3]) *There are numbers $c_k \in [0,1]$ such that the following holds: If F is a k-CNF formula over n variables with a unique satisfying assignment a, then $\mathbb{E}_\sigma[Z_i(F, \sigma, a)] \le c_k$ for all $1 \le i \le n$. Furthermore, for large k we have $c_k \approx 1 - \frac{\pi^2}{6k}$, and in particular $c_3 = 2\ln(2) - 1 \approx 0.38$.*

Combining everything, Paturi, Pudlák, Saks, and Zane obtain their main result:

Theorem 1 ([3]) *Let F be a k-CNF formula with a unique satisfying assignment a. Then PPSZ finds this satisfying assignment with probability at least $2^{-c_k n}$.*

It takes a considerable additional effort to show that the same bound holds also if F has multiple satisfying assignments:

Theorem 2 ([2]) *Let F be a satisfiable k-CNF formula. Then PPSZ finds a satisfying assignment with probability at least $2^{-c_k n}$.*

We sketch the intuition behind the proof of Lemma 2. It turns out that in the worst case the event $Z_i = 1$ can be described by the following random experiment: Let $T = (V, E)$ be the infinite rooted $(k-1)$-ary tree. For each node $v \in V$ choose $\tau(v) \in [0,1]$ randomly and independently. Call a node v *alive* if $\tau(v) \ge \tau(\mathrm{root})$. Then $\Pr[Z_i = 1]$ is (roughly) equal to the probability that T contains an infinite path of alive vertices, starting with the root. Call this probability c_k. A simple calculation shows that $c_3 = 2\ln(2) - 1$. For larger values of c_k, there is not necessarily a closed form, but Paturi, Pudlák, Saks, and Zane show that $c_k \approx 1 - \frac{\pi^2}{6k}$ for large k.

Hard Instances for the PPSZ Algorithm

One can construct instances on which the success probability of PPSZ is exponentially small. The construction is probabilistic and rather simple. Its analysis is quite technical, so we can only sketch it here. We start with some easy estimates. By Lemma 1 we can write the success probability of PPSZ as

$$\Pr_{\sigma,b}[\mathrm{ppsz}(F,\sigma,b) \in \mathrm{sat}(F)] = \sum_{a \in \mathrm{sat}(F)} \mathbb{E}_\sigma[2^{-Z(F,\sigma,a)}]. \quad (2)$$

Above we used Jensen's inequality to prove $\mathbb{E}[2^{-Z}] \geq 2^{-\mathbb{E}[Z]}$. In this section we want to construct *hard instances*, that is, instances on which the success probability (2) is exponentially small. Thus, we cannot use Jensen's inequality, as it gives a lower bound, not an upper. Instead, we use the following trivial estimate:

$$\sum_{a \in \mathrm{sat}(F)} \mathbb{E}_\sigma[2^{-Z(F,\sigma,a)}] \leq \sum_{a \in \mathrm{sat}(F)} \max_\sigma 2^{-Z(F,\sigma,a)} \leq |\mathrm{sat}(F)| \cdot \max_{a \in \mathrm{sat}(F),\sigma} 2^{-Z(F,\sigma,a)}. \quad (3)$$

We would like to construct a satisfiable k-CNF formula F for which (i) $|\mathrm{sat}(F)|$ is small, i.e., F has few satisfying assignments, and (ii) $Z(F,\sigma,a)$ is large for every permutation and every satisfying assignment a. It turns out there are formulas satisfying both requirements:

Theorem 3 *There are numbers ϵ_k converging to 0 such that the following holds: For every k, there is a family $(F_n)_{n\geq 1}$, where each F_n is a satisfiable k-CNF formula over n variables such that*

1. $|\mathrm{sat}(F_n)| \leq 2^{\epsilon_k n}$.
2. $Z(F,\sigma,a) \geq (1-\epsilon_k)n$ *for all* σ *and all* $a \in \mathrm{sat}(F_n)$.

Thus, the probability of PPSZ finding a satisfying assignment of F_n is at most $2^{-(1-2\epsilon_k)n}$. Furthermore, $\epsilon_k \leq \frac{C\log^2(k)}{k}$ for some universal constant C.

This theorem shows that PPSZ has exponentially small success probability. Also, it shows that the *strong exponential time hypothesis* (SETH) holds for PPSZ: As k grows, the advantage over the trivial success probability 2^{-n} becomes negligible.

The Probabilistic Construction

Let $A \in \mathbb{F}_2^{n\times n}$. The system $Ax = 0$ defines a Boolean function $f_A : \{0,1\}^n \to \{0,1\}$ as follows: $f_A(x) = 1$ if and only if $A \cdot x = 0$. Say A is *k-sparse* if every row of A has at most k nonzero entries. If A is k-sparse, then f_A can be written as a k-CNF formula with n variables and $2^{k-1}n$ clauses. Our construction will be probabilistic. For this, we define a distribution over k-sparse matrices in $\mathbb{F}_2^{n\times n}$. Our distribution will have the form $\mathcal{D}^n$, where $\mathcal{D}$ is a distribution over row vectors from $\mathbb{F}_2^n$. That is, we sample each row of A independently from $\mathcal{D}$. Let us describe $\mathcal{D}$. Define $\mathbf{e}_i \in \mathbb{F}_2^n$ to be the vector with a 1 at the i^{th} position and 0 elsewhere. Sample $i_1,\ldots,i_k \in \{1,\ldots,n\}$ uniformly and independently and let $X = \mathbf{e}_{i_1} + \cdots + \mathbf{e}_{i_k}$. Clearly, $X \in \mathbb{F}_2^n$ has at most k nonzero entries. This is our distribution $\mathcal{D}$.

Let A be a random matrix sampled as described, and write f_A as a k-CNF formula F. Note that $\mathrm{sat}(F) = \ker A$. The challenge is to show that F satisfies the two conditions of Theorem 3.

Lemma 3 (A has high rank) *With probability $1-o(1)$, $|\ker(A)| \leq 2^{\epsilon_k n}$.*

This shows that F satisfies the first condition of the theorem, i.e., it has few satisfying assignments. Lemma 3 is quite straightforward to prove, though not trivial. The next lemma shows that $Z(F,\sigma,a)$ is large.

Lemma 4 *With probability $1-o(1)$, it holds that $Z(F,\sigma,a) \geq (1-\epsilon_k)n$ for all permutations σ and all $a \in \mathrm{sat}(F)$.*

Proving this lemma is the main technical challenge. The proof uses ideas from proof complexity (indeed, the above construction is inspired by constructions in proof complexity).

Open Problems

Suppose the true worst-case success probability of PPSZ on k-CNF formulas is $2^{-r_k n}$. Paturi, Pudlák, Saks, and Zane have proved that $r_k \leq 1 - \Omega(1k)$. Chen, Scheder, Talebanfard, and

Tang showed that $r_k \geq 1 - O\left(\frac{\log^2 k}{k}\right)$. Can one close this gap by construction harder instances or maybe even improve the analysis of PPSZ?

What is the average-case success probability of PPSZ on F when we sample A from $\mathcal{D}^n$? Note that F is exponentially hard with probability $1 - o(1)$, but this might leave a $1/n$ probability that F is very easy for PPSZ.

The construction of [1] is probabilistic. Can one make it explicit? The proof of Lemma 4 uses (implicit in [1]) a nonstandard notion of expansion. We do not know of explicit construction of those expanders.

Cross-References

- Backtracking Based k-SAT Algorithms
- Derandomization of k-SAT Algorithm
- Exact Algorithms and Strong Exponential Time Hypothesis
- Exact Algorithms and Time/Space Tradeoffs
- Exact Algorithms for General CNF SAT
- Exact Algorithms for k-SAT Based on Local Search
- Exact Algorithms for Maximum Two-Satisfiability
- Exact Graph Coloring Using Inclusion-Exclusion
- Random Planted 3-SAT
- Thresholds of Random k-Sat
- Unique k-SAT and General k-SAT

Recommended Reading

1. Chen S, Scheder D, Tang B, Talebanfard N (2013) Exponential lower bounds for the PPSZ k-SAT algorithm. In Khanna S (ed) SODA, New Orleans. SIAM, pp 1253–1263
2. Hertli T (2011) 3-SAT faster and simpler – unique-SAT bounds for PPSZ hold in general. In Ostrovsky R (ed) FOCS, Palm Springs. IEEE, pp 277–284
3. Paturi R , Pudlák P, Saks ME, Zane F (2005) An improved exponential-time algorithm for k-SAT. J ACM 52(3):337–364
4. Paturi R, Pudlák P, Zane F (1999) Satisfiability coding lemma. Chicago J Theor Comput Sci 11(19). (electronic)
5. Robin RA, Scheder D (2011) A full derandomization of Schöning's k-SAT algorithm. In Fortnow L Salil P, Vadhan (eds) STOC, San Jose. ACM, pp 245–252
6. Schöning U (1999) A probabilistic algorithm for k-SAT and constraint satisfaction problems. In FOCS '99: Proceedings of the 40th annual symposium on foundations of computer science, New York. IEEE Computer Society, Washington, DC, p 410

External Sorting and Permuting

Jeffrey Scott Vitter
University of Kansas, Lawrence, KS, USA

Keywords

Disk; External memory; I/O; Out-of-core; Permuting; Secondary storage; Sorting

Synonyms

Out-of-core sorting

Years and Authors of Summarized Original Work

1988; Aggarwal, Vitter

Problem Definition

Notations The main properties of magnetic disks and multiple disk systems can be captured by the commonly used *parallel disk model* (PDM), which is summarized below in its current form as developed by Vitter and Shriver [22]:

N = problem size (in units of data items);

M = internal memory size(inunitsofdata items);

B = block transfer size (in units of data items);

D = number of independent disk drives;

P = number of CPUs,

where $M < N$, and $1 \leq DB \leq M/2$. The data items are assumed to be of fixed length. In a single I/O, each of the D disks can simultaneously transfer a block of B contiguous data items. (In the original 1988 article [2], the D blocks per I/O were allowed to come from the same disk, which is not realistic.) If $P \leq D$, each of the P processors can drive about D/P disks; if $D < P$, each disk is shared by about P/D processors. The internal memory size is M/P per processor, and the P processors are connected by an interconnection network.

It is convenient to refer to some of the above PDM parameters in units of disk blocks rather than in units of data items; the resulting formulas are often simplified. We define the lowercase notation

$$n = \frac{N}{B}, \quad m = \frac{M}{B}, \quad q = \frac{Q}{B}, \quad z = \frac{Z}{B} \tag{1}$$

to be the problem input size, internal memory size, query specification size, and query output size, respectively, in units of disk blocks.

The primary measures of performance in PDM are:

1. The number of I/O operations performed
2. The amount of disk space used
3. The internal (sequential or parallel) computation time

For reasons of brevity in this survey, focus is restricted onto only the first two measures. Most of the algorithms run in $O(N \log N)$ CPU time with one processor, which is optimal in the comparison model, and in many cases are optimal for parallel CPUs. In the word-based RAM model, sorting can be done more quickly in $O(N \log \log N)$ CPU time. Arge and Thorup [5] provide sorting algorithms that are theoretically optimal in terms of both I/Os and time in the word-based RAM model. In terms of auxiliary storage in external memory, algorithms and data structures should ideally use linear space, which means $O(N/B) = O(n)$ disk blocks of storage. Vitter [20] gives further details about the PDM model and provides optimal algorithms and data structures for a variety of problems. The content of this chapter comes largely from an abbreviated form of [19].

Problem 1 External sorting
INPUT: The input data records $R_0, R_1, R_2, \ldots$ are initially "striped" across the D disks, in units of blocks, so that record R_i is in block $\lfloor i/B \rfloor$ and block j is stored on disk $j \bmod D$.
OUTPUT: A striped representation of a permuted ordering $R_{\sigma(0)}, R_{\sigma(1)}, R_{\sigma(2)}, \ldots$ of the input records with the property that $key(R_{\sigma(i)}) \leq key(R_{\sigma(i+1)})$ for all $i \geq 0$.

Permuting is the special case of sorting in which the permutation that describes the final position of the records is given explicitly and does not have to be discovered, for example, by comparing keys.

Problem 2 Permuting
INPUT: Same input assumptions as in external sorting. In addition, a permutation σ of the integers $\{0, 1, 2, \ldots, N-1\}$ is specified.
OUTPUT: A striped representation of a permuted ordering $R_{\sigma(0)}, R_{\sigma(1)}, R_{\sigma(2)}, \ldots$ of the input records.

Key Results

Theorem 1 ([2, 15]) *The average-case and worst-case number of I/Os required for sorting $N = nB$ data items using D disks is*

$$Sort(N) = \Theta\left(\frac{n}{D} \log_m n\right). \tag{2}$$

Theorem 2 ([2]) *The average-case and worst-case number of I/Os required for permuting N data items using D disks is*

$$\Theta\left(\min\left\{\frac{N}{D}, Sort(N)\right\}\right). \tag{3}$$

A more detailed lower bound is provided in (9) in section "Lower Bounds on I/O."

Matrix transposition is the special case of permuting in which the permutation can be represented as a transposition of a matrix from row-major order into column-major order.

Theorem 3 ([2]) *With D disks, the number of I/Os required to transpose a $p \times q$ matrix from row-major order to column-major order is*

$$\Theta\left(\frac{n}{D}\log_m \min\{M, p, q, n\}\right), \qquad (4)$$

where $N = pq$ and $n = N/B$.

Matrix transposition is a special case of a more general class of permutations called *bit-permute/complement* (BPC) permutations, which in turn is a subset of the class of *bit-matrix-multiply/complement* (BMMC) permutations. BMMC permutations are defined by a $\log N \times \log N$ nonsingular 0-1 matrix A and a $(\log N)$-length 0-1 vector c. An item with binary address x is mapped by the permutation to the binary address given by $Ax \oplus c$, where $\oplus$ denotes bitwise exclusive-or. BPC permutations are the special case of BMMC permutations in which A is a permutation matrix, that is, each row and each column of A contain a single 1. BPC permutations include matrix transposition, bit-reversal permutations (which arise in the FFT), vector-reversal permutations, hypercube permutations, and matrix re-blocking. Cormen et al. [8] characterize the optimal number of I/Os needed to perform any given BMMC permutation solely as a function of the associated matrix A, and they give an optimal algorithm for implementing it.

Theorem 4 ([8]) *With D disks, the number of I/Os required to perform the BMMC permutation defined by matrix A and vector c is*

$$\Theta\left(\frac{n}{D}\left(1 + \frac{\text{rank}(\gamma)}{\log m}\right)\right), \qquad (5)$$

where γ is the lower-left $\log n \times \log B$ submatrix of A.

The two main paradigms for external sorting are *distribution* and *merging*, which are discussed in the following sections for the PDM model.

Sorting by Distribution

Distribution sort [12] is a recursive process that uses a set of $S - 1$ partitioning elements to partition the items into S disjoint buckets. All the items in one bucket precede all the items in the next bucket. The sort is completed by recursively sorting the individual buckets and concatenating them together to form a single fully sorted list.

One requirement is to choose the $S - 1$ partitioning elements so that the buckets are of roughly equal size. When that is the case, the bucket sizes decrease from one level of recursion to the next by a relative factor of $\Theta(S)$, and thus there are $O(\log_S n)$ levels of recursion. During each level of recursion, the data are scanned. As the items stream through internal memory, they are partitioned into S buckets in an online manner. When a buffer of size B fills for one of the buckets, its block is written to the disks in the next I/O, and another buffer is used to store the next set of incoming items for the bucket. Therefore, the maximum number of buckets (and partitioning elements) is $S = \Theta(M/B) = \Theta(m)$, and the resulting number of levels of recursion is $\Theta(\log_m n)$. How to perform each level of recursion in a linear number I/Os is discussed in [2, 14, 22].

An even better way to do distribution sort, and deterministically at that, is the BalanceSort method developed by Nodine and Vitter [14]. During the partitioning process, the algorithm keeps track of how evenly each bucket has been distributed so far among the disks. It maintains an invariant that guarantees good distribution across the disks for each bucket.

The distribution sort methods mentioned above for parallel disks perform write operations in complete stripes, which make it easy to write parity information for use in error correction and recovery. But since the blocks written in each stripe typically belong to multiple buckets, the buckets themselves will not be striped on the disks, and thus the disks must be used independently during read operations. In the write phase, each bucket must therefore keep track of the last block written to each disk so that the blocks for the bucket can be linked together.

An orthogonal approach is to stripe the contents of each bucket across the disks so that read operations can be done in a striped manner. As a result, the write operations must use disks independently, since during each write, multiple buckets will be writing to multiple stripes. Error correction and recovery can still be handled efficiently by devoting to each bucket one block-sized buffer in internal memory. The buffer is continuously updated to contain the exclusive-or (parity) of the blocks written to the current stripe, and after $D - 1$ blocks have been written, the parity information in the buffer can be written to the final (Dth) block in the stripe.

Under this new scenario, the basic loop of the distribution sort algorithm is, as before, to read one memory load at a time and partition the items into S buckets. However, unlike before, the blocks for each individual bucket will reside on the disks in contiguous stripes. Each block therefore has a predefined place where it must be written. With the normal round-robin ordering for the stripes (namely, $\ldots, 1, 2, 3, \ldots, D, 1, 2, 3, \ldots, D, \ldots$), the blocks of different buckets may "collide," meaning that they need to be written to the same disk, and subsequent blocks in those same buckets will also tend to collide. Vitter and Hutchinson [21] solve this problem by the technique of *randomized cycling*. For each of the S buckets, they determine the ordering of the disks in the stripe for that bucket via a random permutation of $\{1, 2, \ldots, D\}$. The S random permutations are chosen independently. If two blocks (from different buckets) happen to collide during a write to the same disk, one block is written to the disk and the other is kept on a write queue. With high probability, subsequent blocks in those two buckets will be written to different disks and thus will not collide. As long as there is a small pool of available buffer space to temporarily cache the blocks in the write queues, Vitter and Hutchinson [21] show that with high probability the writing proceeds optimally.

The randomized cycling method or the related merge sort methods discussed at the end of section "Sorting by Merging" are the methods of choice for sorting with parallel disks. Distribution sort algorithms may have an advantage over the merge approaches presented in section "Sorting by Merging" in that they typically make better use of lower levels of cache in the memory hierarchy of real systems, based upon analysis of distribution sort and merge sort algorithms on models of hierarchical memory.

Sorting by Merging

The *merge* paradigm is somewhat orthogonal to the distribution paradigm of the previous section. A typical merge sort algorithm works as follows [12]: In the "run formation" phase, the n blocks of data are scanned, one memory load at a time; each memory load is sorted into a single "run," which is then output onto a series of stripes on the disks. At the end of the run formation phase, there are $N/M = n/m$ (sorted) runs, each striped across the disks. (In actual implementations, "replacement selection" can be used to get runs of $2M$ data items, on the average, when $M \gg B$ [12].) After the initial runs are formed, the merging phase begins. In each pass of the merging phase, R runs are merged at a time. For each merge, the R runs are scanned and its items merged in an online manner as they stream through internal memory. Double buffering is used to overlap I/O and computation. At most $R = \Theta(m)$ runs can be merged at a time, and the resulting number of passes is $O(\log_m n)$.

To achieve the optimal sorting bound (2), each merging pass must be done in $O(n/D)$ I/Os, which is easy to do for the single-disk case. In the more general multiple-disk case, each parallel read operation during the merging must on the average bring in the next $\Theta(D)$ blocks needed for the merging. The challenge is to ensure that those blocks reside on different disks so that they can be read in a single I/O (or a small constant number of I/Os). The difficulty lies in the fact that the runs being merged were themselves formed during the previous merge pass. Their blocks were written to the disks in the previous pass without knowledge of how they would interact with other runs in later merges.

The Greed Sort method of Nodine and Vitter [15] was the first optimal deterministic EM algorithm for sorting with multiple disks. It works

by relaxing the merging process with a final pass to fix the merging. Aggarwal and Plaxton [1] developed an optimal deterministic merge sort based upon the Sharesort hypercube parallel sorting algorithm. To guarantee even distribution during the merging, it employs two high-level merging schemes in which the scheduling is almost oblivious. Like Greed Sort, the Sharesort algorithm is theoretically optimal (i.e., within a constant factor of optimal), but the constant factor is larger than the distribution sort methods.

One of the most practical methods for sorting is based upon the *simple randomized merge sort* (SRM) algorithm of Barve et al. [7], referred to as "randomized striping" by Knuth [12]. Each run is striped across the disks, but with a random starting point (the only place in the algorithm where randomness is utilized). During the merging process, the next block needed from each disk is read into memory, and if there is not enough room, the least needed blocks are "flushed" (without any I/Os required) to free up space.

Further improvements in merge sort are possible by a more careful prefetching schedule for the runs. Barve et al. [6], Kallahalla and Varman [11], Shah et al. [17], and others have developed competitive and optimal methods for prefetching blocks in parallel I/O systems.

Hutchinson et al. [10] have demonstrated a powerful duality between parallel writing and parallel prefetching, which gives an easy way to compute optimal prefetching and caching schedules for multiple disks. More significantly, they show that the same duality exists between distribution and merging, which they exploit to get a provably optimal and very practical parallel disk merge sort. Rather than use random starting points and round-robin stripes as in SRM, Hutchinson et al. [10] order the stripes for each run independently, based upon the randomized cycling strategy discussed in section "Sorting by Distribution" for distribution sort. These approaches have led to successfully faster external memory sorting algorithms [9]. Clever algorithm engineering optimizations on multicore architectures have won recent big data sorting competitions [16].

Handling Duplicates: Bundle Sorting

For the problem of *duplicate removal*, in which there are a total of K distinct items among the N items, Arge et al. [4] use a modification of merge sort to solve the problem in $O\left(n \max\left\{1, \log_m(K/B)\right\}\right)$ I/Os, which is optimal in the comparison model. When duplicates get grouped together during a merge, they are replaced by a single copy of the item and a count of the occurrences. The algorithm can be used to sort the file, assuming that a group of equal items can be represented by a single item and a count.

A harder instance of sorting called *bundle sorting* arises when there are K distinct key values among the N items, but all the items have different secondary information that must be maintained, and therefore items cannot be aggregated with a count. Matias et al. [13] develop optimal distribution sort algorithms for bundle sorting using

$$O\left(n \max\left\{1, \log_m \min\{K, n\}\right\}\right) \tag{6}$$

I/Os and prove the matching lower bound. They also show how to do bundle sorting (and sorting in general) *in place* (i.e., without extra disk space).

Permuting and Transposition

Permuting is the special case of sorting in which the key values of the N data items form a permutation of $\{1, 2, \ldots, N\}$. The I/O bound (3) for permuting can be realized by one of the optimal sorting algorithms except in the extreme case $B \log m = o(\log n)$, where it is faster to move the data items one by one in a nonblocked way. The one-by-one method is trivial if $D = 1$, but with multiple disks, there may be bottlenecks on individual disks; one solution for doing the permuting in $O(N/D)$ I/Os is to apply the randomized balancing strategies of [22].

Matrix transposition can be as hard as general permuting when B is relatively large (say, $\frac{1}{2}M$) and N is $O(M^2)$, but for smaller B, the special structure of the transposition permutation makes transposition easier. In particular, the matrix can be broken up into square submatrices of B^2 elements such that each submatrix contains B

blocks of the matrix in row-major order and also B blocks of the matrix in column-major order. Thus, if $B^2 < M$, the transpositions can be done in a simple one-pass operation by transposing the submatrices one at a time in internal memory. Thonangi and Yang [18] discuss other types of permutations realizable with fewer I/Os than sorting.

Fast Fourier Transform and Permutation Networks

Computing the fast Fourier transform (FFT) in external memory consists of a series of I/Os that permit each computation implied by the FFT directed graph (or butterfly) to be done while its arguments are in internal memory. A permutation network computation consists of an oblivious (fixed) pattern of I/Os such that any of the $N!$ possible permutations can be realized; data items can only be reordered when they are in internal memory. A permutation network can be realized by a series of three FFTs.

The algorithms for FFT are faster and simpler than for sorting because the computation is nonadaptive in nature, and thus the communication pattern is fixed in advance [22].

Lower Bounds on I/O

The following proof of the permutation lower bound (3) of Theorem 2 is due to Aggarwal and Vitter [2]. The idea of the proof is to calculate, for each $t \geq 0$, the number of distinct orderings that are realizable by sequences of t I/Os. The value of t for which the number of distinct orderings first exceeds $N!/2$ is a lower bound on the average number of I/Os (and hence the worst-case number of I/Os) needed for permuting.

Assuming for the moment that there is only one disk, $D = 1$, consider how the number of realizable orderings can change as a result of an I/O. In terms of increasing the number of realizable orderings, the effect of reading a disk block is considerably more than that of writing a disk block, so it suffices to consider only the effect of read operations. During a read operation, there are at most B data items in the read block, and they can be interspersed among the M items in internal memory in at most $\binom{M}{B}$ ways, so the number of realizable orderings increases by a factor of $\binom{M}{B}$. If the block has never before resided in internal memory, the number of realizable orderings increases by an extra $B!$ factor, since the items in the block can be permuted among themselves. (This extra contribution of $B!$ can only happen once for each of the N/B original blocks.) There are at most $n + t \leq N \log N$ ways to choose which disk block is involved in the tth I/O (allowing an arbitrary amount of disk space). Hence, the number of distinct orderings that can be realized by all possible sequences of t I/Os is at most

$$(B!)^{N/B} \left(N(\log N) \binom{M}{B} \right)^t. \tag{7}$$

Setting the expression in (7) to be at least $N!/2$, and simplifying by taking the logarithm, the result is

$$N \log B + t \left(\log N + B \log \frac{M}{B} \right) = \Omega(N \log N). \tag{8}$$

Solving for t gives the matching lower bound $\Omega(n \log_m n)$ for permuting for the case $D = 1$. The general lower bound (3) of Theorem 2 follows by dividing by D.

Hutchinson et al. [10] derive an asymptotic lower bound (i.e., one that accounts for constant factors) from a more refined argument that analyzes both input operations and output operations. Assuming that $m = M/B$ is an increasing function, the number of I/Os required to sort or permute n indivisible items, up to lower-order terms, is at least

$$\frac{2N}{D} \frac{\log n}{B \log m + 2 \log N} \sim \begin{cases} \frac{2n}{D} \log_m n & \text{if } B \log m = \omega(\log N); \\ \frac{N}{D} & \text{if } B \log m = o(\log N). \end{cases} \tag{9}$$

For the typical case in which $B \log m = \omega(\log N)$, the lower bound, up to lower order terms, is $2n \log_m n$ I/Os. For the pathological in which $B \log m = o(\log N)$, the I/O lower bound is asymptotically N/D.

Permuting is a special case of sorting, and hence the permuting lower bound applies also to sorting. In the unlikely case that $B \log m = o(\log n)$, the permuting bound is only $\Omega(N/D)$, and in that case the comparison model must be used to get the full lower bound (2) of Theorem 1 [2]. In the typical case in which $B \log m = \Omega(\log n)$, the comparison model is not needed to prove the sorting lower bound; the difficulty of sorting in that case arises not from determining the order of the data but from permuting (or routing) the data.

The proof used above for permuting also works for permutation networks, in which the communication pattern is oblivious (fixed). Since the choice of disk block is fixed for each t, there is no $N \log N$ term as there is in (7), and correspondingly there is no additive $\log N$ term in the inner expression as there is in (8). Hence, solving for t gives the lower bound (2) rather than (3). The lower bound follows directly from the counting argument; unlike the sorting derivation, it does not require the comparison model for the case $B \log m = o(\log n)$. The lower bound also applies directly to FFT, since permutation networks can be formed from three FFTs in sequence. The transposition lower bound involves a potential argument based upon a togetherness relation [2].

For the problem of bundle sorting, in which the N items have a total of K distinct key values (but the secondary information of each item is different), Matias et al. [13] derive the matching lower bound.

The lower bounds mentioned above assume that the data items are in some sense "indivisible," in that they are not split up and reassembled in some magic way to get the desired output. It is conjectured that the sorting lower bound (2) remains valid even if the indivisibility assumption is lifted. However, for an artificial problem related to transposition, removing the indivisibility assumption can lead to faster algorithms. Whether the conjecture is true is a challenging theoretical open problem.

Applications

Sorting and sorting-like operations account for a significant percentage of computer use [12], with numerous database applications. In addition, sorting is an important paradigm in the design of efficient EM algorithms, as shown in [20], where several applications can be found. With some technical qualifications, many problems that can be solved easily in linear time in internal memory, such as permuting, list ranking, expression tree evaluation, and finding connected components in a sparse graph, require the same number of I/Os in PDM as does sorting.

Open Problems

Several interesting challenges remain. One difficult theoretical problem is to prove lower bounds for permuting and sorting without the indivisibility assumption. Another question is to determine the I/O cost for each individual permutation, as a function of some simple characterization of the permutation, such as number of inversions. A continuing goal is to develop optimal EM algorithms and to translate theoretical gains into observable improvements in practice.

Many interesting challenges and opportunities in algorithm design and analysis arise from new architectures being developed. For example, Arge et al. [3] propose the *parallel external memory (PEM)* model for the design of efficient algorithms for chip multiprocessors, in which each processor has a private cache and shares a larger main memory with the other processors. The paradigms described earlier form the basis for efficient algorithms for sorting, selection, and prefix sums. Further architectures to explore include other forms of multicore architectures, networks of workstations, hierarchical storage devices, disk drives with processing capabilities, and storage devices based upon microelectrome-

chanical systems (MEMS). Active (or intelligent) disks, in which disk drives have some processing capability and can filter information sent to the host, have been proposed to further reduce the I/O bottleneck, especially in large database applications. MEMS-based nonvolatile storage has the potential to serve as an intermediate level in the memory hierarchy between DRAM and disks. It could ultimately provide better latency and bandwidth than disks, at less cost per bit than DRAM.

URL to Code

Two systems for developing external memory algorithms are TPIE and STXXL, which can be downloaded from http://www.madalgo.au.dk/tpie/ and http://stxxl.sourceforge.net/, respectively. Both systems include subroutines for sorting and permuting and facilitate development of more advanced algorithms.

Cross-References

▸ I/O-Model

Recommended Reading

1. Aggarwal A, Plaxton CG (1994) Optimal parallel sorting in multi-level storage. In: Proceedings of the 5th ACM-SIAM symposium on discrete algorithms, Arlington, vol 5, pp 659–668
2. Aggarwal A, Vitter JS (1988) The input/output complexity of sorting and related problems. Commun ACM 31:1116–1127
3. Arge L, Goodrich MT, Nelson M, Sitchinava N (2008) Fundamental parallel algorithms for private-cache chip multiprocessors. In: Proceedings of the 20th symposium on parallelism in algorithms and architectures, Munich, pp 197–206
4. Arge L, Knudsen M, Larsen K (1993) A general lower bound on the I/O-complexity of comparison-based algorithms. In: Proceedings of the workshop on algorithms and data structures, Montréal. Lecture notes in computer science, vol 709, pp 83–94
5. Arge L, Thorup M (2013) RAM-efficient external memory sorting. In: Proceedings of the 24th international symposium on algorithms and computation, Hong Kong. Lecture notes in computer science, vol 8283, pp 491–501
6. Barve RD, Kallahalla M, Varman PJ, Vitter JS (2000) Competitive analysis of buffer management algorithms. J Algorithms 36:152–181
7. Barve RD, Vitter JS (2002) A simple and efficient parallel disk mergesort. ACM Trans Comput Syst 35:189–215
8. Cormen TH, Sundquist T, Wisniewski LF (1999) Asymptotically tight bounds for performing BMMC permutations on parallel disk systems. SIAM J Comput 28:105–136
9. Dementiev R, Sanders P (2003) Asynchronous parallel disk sorting. In: Proceedings of the 15th ACM symposium on parallelism in algorithms and architectures, San Diego, pp 138–148
10. Hutchinson DA, Sanders P, Vitter JS (2005) Duality between prefetching and queued writing with parallel disks. SIAM J Comput 34:1443–1463
11. Kallahalla M, Varman PJ (2005) Optimal read-once parallel disk scheduling. Algorithmica 43:309–343
12. Knuth DE (1998) Sorting and searching. The art of computer programming, vol 3, 2nd edn. Addison-Wesley, Reading
13. Matias Y, Segal E, Vitter JS (2006) Efficient bundle sorting. SIAM J Comput 36(2):394–410
14. Nodine MH, Vitter JS (1993) Deterministic distribution sort in shared and distributed memory multiprocessors. In: Proceedings of the 5th ACM symposium on parallel algorithms and architectures, Velen, vol 5. ACM, pp 120–129
15. Nodine MH, Vitter JS (1995) Greed sort: an optimal sorting algorithm for multiple disks. J ACM 42:919–933
16. Rahn M, Sanders P, Singler J (2010) Scalable distributed-memory external sorting. In: Proceedings of the 26th IEEE international conference on data engineering, Long Beach, pp 685–688
17. Shah R, Varman PJ, Vitter JS (2004) Online algorithms for prefetching and caching on parallel disks. In: Proceedings of the 16th ACM symposium on parallel algorithms and architectures, Barcelona, pp 255–264
18. Thonangi R, Yang J (2013) Permuting data on random-access block storage. Proc VLDB Endow 6(9):721–732
19. Vitter JS (2001) External memory algorithms and data structures: dealing with massive data. ACM Comput Surv 33(2):209–271
20. Vitter JS (2008) Algorithms and data structures for external memory. Series on foundations and trends in theoretical computer science. Now Publishers, Hanover. (Also referenced as Volume 2, Issue 4 of Foundations and trends in theoretical computer science, Now Publishers)
21. Vitter JS, Hutchinson DA (2006) Distribution sort with randomized cycling. J ACM 53:656–680
22. Vitter JS, Shriver EAM (1994) Algorithms for parallel memory I: two-level memories. Algorithmica 12:110–147

F

Facility Location

Karen Aardal[1,2], Jaroslaw Byrka[1,2], and Mohammad Mahdian[3]
[1]Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands
[2]Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
[3]Yahoo! Research, Santa Clara, CA, USA

Keywords

Plant location; Warehouse location

Years and Authors of Summarized Original Work

1997; Shmoys, Tardos, Aardal

Problem Definition

Facility location problems concern situations where a planner needs to determine the location of facilities intended to serve a given set of clients. The objective is usually to minimize the sum of the cost of opening the facilities and the cost of serving the clients by the facilities, subject to various constraints, such as the number and the type of clients a facility can serve. There are many variants of the facility location problem, depending on the structure of the cost function and the constraints imposed on the solution. Early references on facility location problems include Kuehn and Hamburger [35], Balinski and Wolfe [8], Manne [40], and Balinski [7]. Review works include Krarup and Pruzan [34] and Mirchandani and Francis [42]. It is interesting to notice that the algorithm that is probably one of the most effective ones to solve the uncapacitated facility location problem to optimality is the primal-dual algorithm combined with branch-and-bound due to Erlenkotter [16] dating back to 1978. His primal-dual scheme is similar to techniques used in the modern literature on approximation algorithms.

More recently, extensive research into approximation algorithms for facility location problems has been carried out. Review articles on this topic include Shmoys [49, 50] and Vygen [55]. Besides its theoretical and practical importance, facility location problems provide a showcase of common techniques in the field of approximation algorithms, as many of these techniques such as linear programming rounding, primal-dual methods, and local search have been applied successfully to this family of problems. This entry defines several facility location problems, gives a few historical pointers, and lists approximation algorithms with an emphasis on the results derived in the paper by Shmoys, Tardos, and Aardal [51]. The techniques applied to the *uncapacitated facility location* (UFL) problem are discussed in some more detail.

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

In the UFL problem, a set $\mathcal{F}$ of n_f *facilities* and a set C of n_c *clients* (also known as *cities*, or *demand points*) are given. For every facility $i \in \mathcal{F}$, the *facility opening* cost is equal to f_i. Furthermore, for every facility $i \in \mathcal{F}$ and client $j \in C$, there is a *connection cost* c_{ij}. The objective is to open a subset of the facilities and connect each client to an open facility so that the total cost is minimized. Notice that once the set of open facilities is specified, it is optimal to connect each client to the open facility that yields smallest connection cost. Therefore, the objective is to find a set $S \subseteq \mathcal{F}$ that minimizes $\sum_{i \in S} f_i + \sum_{j \in C} \min_{i \in S}\{c_{ij}\}$. This definition and the definitions of other variants of the facility location problem in this entry assume unit demand at each client. It is straightforward to generalize these definitions to the case where each client has a given demand. The UFL problem can be formulated as the following integer program due to Balinski [7]. Let y_i, $i \in \mathcal{F}$ be equal to 1 if facility i is open, and equal to 0 otherwise. Let x_{ij}, $i \in \mathcal{F}$, $j \in C$ be the fraction of client j assigned to facility i.

$$\min \sum_{i \in \mathcal{F}} f_i y_i + \sum_{i \in \mathcal{F}} \sum_{j \in C} c_{ij} x_{ij} \tag{1}$$

$$\text{subject to} \sum_{i \in \mathcal{F}} x_{ij} = 1, \quad \text{for all } j \in C, \tag{2}$$

$$x_{ij} - y_i \leq 0, \quad \text{for all } i \in \mathcal{F},\ j \in C \tag{3}$$

$$x \geq 0,\ y \in \{0,1\}^{n_f} \tag{4}$$

In the *linear programming (LP) relaxation* of UFL the constraint $y \in \{0,1\}^{n_f}$ is substituted by the constraint $y \in [0,1]^{n_f}$. Notice that in the uncapacitated case, it is not necessary to require $x_{ij} \in \{0,1\}$, $i \in \mathcal{F}$, $j \in C$ if each client has to be serviced by precisely one facility, as $0 \leq x_{ij} \leq 1$ by constraints (2) and (4). Moreover, if x_{ij} is not integer, then it is always possible to create an integer solution with the same cost by assigning client j completely to one of the facilities currently servicing j.

A γ-approximation algorithm is a polynomial algorithm that, in case of minimization, is guaranteed to produce a feasible solution having value *at most* γz^*, where z^* is the value of an optimal solution, and $\gamma \geq 1$. If $\gamma = 1$ the algorithm produces an optimal solution. In case of maximization, the algorithm produces a solution having value *at least* γz^*, where $0 \leq \gamma \leq 1$.

Hochbaum [25] developed an $O(\log n)$-approximation algorithm for UFL. By a straightforward reduction from the Set Cover problem, it can be shown that this cannot be improved unless $NP \subseteq DTIME[n^{O(\log \log n)}]$ due to a result by Feige [17]. However, if the connection costs are restricted to come from distances in a metric space, namely $c_{ij} = c_{ji} \geq 0$ for all $i \in \mathcal{F}, j \in C$ (nonnegativity and symmetry) and $c_{ij} + c_{ji'} + c_{i'j'} \geq c_{ij'}$ for all $i, i' \in \mathcal{F}, j, j' \in C$ (triangle inequality), then constant approximation guarantees can be obtained. In all results mentioned below, except for the maximization objectives, it is assumed that the costs satisfy these restrictions. If the distances between facilities and clients are Euclidean, then for some location problems approximation schemes have been obtained [5].

Variants and Related Problems

A variant of the uncapacitated facility location problem is obtained by considering the objective coefficients c_{ij} as the per unit profit of servicing client j from facility i. The *maximization version* of UFL, max-UFL is obtained by maximizing the profit minus the facility opening cost, i.e., $\max \sum_{i \in \mathcal{F}} \sum_{j \in C} c_{ij} x_{ij} - \sum_{i \in \mathcal{F}} f_i y_i$. This variant was introduced by Cornuéjols, Fisher, and Nemhauser [15].

In the *k-median problem* the facility opening cost is removed from the objective function (1) to obtain $\min \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij}$, and the constraint that no more than k facilities may be opened, $\sum_{i \in M} y_i \leq k$, is added. In the *k-center problem* the constraint $\sum_{i \in M} y_i \leq k$ is again included, and the objective function here is to minimize the maximum distance used on a link between an open facility and a client.

In the *capacitated facility location problem* a capacity constraint $\sum_{j \in C} x_{ij} \leq u_i y_i$ is added for all $i \in \mathcal{F}$. Here it is important to distinguish between the *splittable* and the *unsplittable*

case, and also between *hard capacities* and *soft capacities*. In the splittable case one has $x \geq 0$, allowing for a client to be serviced by multiple depots, and in the unsplittable case one requires $x \in \{0,1\}^{n_f \times n_c}$. If each facility can be opened at most once (i.e., $y_i \in \{0,1\}$), the capacities are called hard; otherwise, if the problem allows a facility i to be opened any number r of times to serve ru_i clients, the capacities are called soft.

In the k-level facility location problem, the following are given: a set C of clients, k disjoint sets $\mathcal{F}_1, \ldots, \mathcal{F}_k$ of facilities, an opening cost for each facility, and connection costs between clients and facilities. The goal is to connect each client j through a path $i_1, \ldots, i_k$ of open facilities, with $i_\ell \in \mathcal{F}_\ell$. The connection cost for this client is $c_{ji_1} + c_{i_1 i_2} + \cdots + c_{i_{k-1} i_k}$. The goal is to minimize the sum of connection costs and facility opening costs.

The problems mentioned above have all been considered by Shmoys, Tardos, and Aardal [51], with the exceptions of max-UFL, and the k-center and k-median problems. The max-UFL variant is included for historical reasons, and k-center and k-median are included since they have a rich history and since they are closely related to UFL. Results on the capacitated facility location problem with hard capacities are mentioned as this, at least from the application point of view, is a more realistic model than the soft capacity version, which was treated in [51]. For k-level facility location, Shmoys et al. considered the case $k = 2$. Here, the problem for general k is considered.

There are many other variants of the facility location problem that are not discussed here. Examples include K-facility location [33], universal facility location [24, 38], online facility location [3, 18, 41], fault tolerant facility location [28, 30, 54], facility location with outliers [12, 28], multicommodity facility location [48], priority facility location [37, 48], facility location with hierarchical facility costs [52], stochastic facility location [23, 37, 46], connected facility location [53], load-balanced facility location [22, 32, 37], concave-cost facility location [24], and capacitated-cable facility location [37, 47].

Key Results

Many algorithms have been proposed for location problems. To begin with, a brief description of the algorithms of Shmoys, Tardos, and Aardal [51] is given. Then, a quick overview of some key results is presented. Some of the algorithms giving the best values of the approximation guarantee γ are based on solving the LP-relaxation by a polynomial algorithm, which can actually be quite time consuming, whereas some authors have suggested fast combinatorial algorithms for facility location problems with less competitive γ-values. Due to space restrictions the focus of this entry is on the algorithms that yield the best approximation guarantees. For more references the survey papers by Shmoys [49, 50] and by Vygen [55] are recommended.

The Algorithms of Shmoys, Tardos, and Aardal

First the algorithm for UFL is described, and then the results that can be obtained by adaptations of the algorithm to other problems are mentioned.

The algorithm solves the LP relaxation and then, in two stages, modifies the obtained fractional solution. The first stage is called *filtering* and it is designed to bound the connection cost of each client to the most distant facility fractionally serving him. To do so, the facility opening variables y_i are scaled up by a constant and then the connection variables x_{ij} are adjusted to use the closest possible facilities.

To describe the second stage, the notion of *clustering*, formalized later by Chudak and Shmoys [13] is used. Based on the fractional solution, the instance is cut into pieces called *clusters*. Each cluster has a distinct client called the *cluster center*. This is done by iteratively choosing a client, not covered by the previous clusters, as the next cluster center, and adding to this cluster the facilities that serve the cluster center in the fractional solution, along with other clients served by these facilities. This construction of clusters guarantees that the facilities in each cluster are open to a total extent of one, and therefore after opening the facility with the smallest opening cost in each cluster, the total

facility opening cost that is paid does not exceed the facility opening cost of the fractional solution. Moreover, by choosing clients for the cluster centers in a greedy fashion, the algorithm makes each cluster center the minimizer of a certain cost function among the clients in the cluster. The remaining clients in the cluster are also connected to the opened facility. The triangle inequality for connection costs is now used to bound the cost of this connection. For UFL, this filtering and rounding algorithm is a 4-approximation algorithm. Shmoys et al. also show that if the filtering step is substituted by *randomized filtering*, an approximation guarantee of 3.16 is obtained.

In the same paper, adaptations of the algorithm, with and without randomized filtering, was made to yield approximation algorithms for the soft-capacitated facility location problem, and for the 2-level uncapacitated problem. Here, the results obtained using randomized filtering are discussed.

For the problem with soft capacities two versions of the problem were considered. Both have equal capacities, i.e., $u_i = u$ for all $i \in \mathcal{F}$. In the first version, a solution is "feasible" if the y-variables either take value 0, or a value between 1 and $\gamma' \geq 1$. Note that γ' is not required to be integer, so the constructed solution is not necessarily integer. This can be interpreted as allowing for each facility i to expand to have capacity $\gamma' u$ at a cost of $\gamma' f_i$. A (γ, γ')-approximation algorithm is a polynomial algorithm that produces such a feasible solution having a total cost within a factor of γ of the true optimal cost, i.e., with $y \in \{0, 1\}^{n_f}$. Shmoys et al. developed a $(5.69, 4.24)$-approximation algorithm for the splittable case of this problem, and a $(7.62, 4.29)$-approximation algorithm for the unsplittable case.

In the second soft-capacitated model, the original problem is changed to allow for the y-variables to take nonnegative integer values, which can be interpreted as allowing multiple facilities of capacity u to be opened at each location. The approximation algorithms in this case produces a solution that is feasible with respect to this modified model. It is easy to show that the approximation guarantees obtained for the previous model also hold in this case, i.e., Shmoys et al. obtained a 5.69-approximation algorithm for splittable demands and a 7.62-approximation algorithm for unsplittable demands. This latter model is the one considered in most later papers, so this is the model that is referred to in the paragraph on soft capacity results below.

UFL

The first algorithm with constant performance guarantee was the 3.16-approximation algorithm by Shmoys, Tardos, and Aardal, see above. Since then numerous improvements have been made. Guha and Khuller [19, 20] proved a lower bound on approximability of 1.463, and introduced a *greedy augmentation procedure*. A series of approximation algorithms based on LP-rounding was then developed (see e.g., [10, 13]). There are also greedy algorithms that only use the LP-relaxation implicitly to obtain a lower bound for a primal-dual analysis. An example is the JMS 1.61-approximation algorithm developed by Jain, Mahdian, and Saberi [29]. Some algorithms combine several techniques, like the 1.52-approximation algorithm of Mahdian, Ye, and Zhang [39], which uses the JMS algorithm and the greedy augmentation procedure. Currently, the best known approximation guarantee is 1.5 reported by Byrka [10]. It is obtained by combining a randomized LP-rounding algorithm with the greedy JMS algorithm.

max-UFL

The first constant factor approximation algorithm was derived in 1977 by Cornuéjols et al. [15] for max-UFL. They showed that opening one facility at a time in a greedy fashion, choosing the facility to open as the one with highest marginal profit, until no facility with positive marginal profit can be found, yields a $(1 - 1/e) \approx 0.632$-approximation algorithm. The current best approximation factor is 0.828 by Ageev and Sviridenko [2].

k-Median, *k*-Center

The first constant factor approximation algorithm for the k-median problem is due to Charikar,

Guha, Tardos, and Shmoys [11]. This LP-rounding algorithm has the approximation ratio of $6\frac{2}{3}$. The currently best known approximation ratio is $3 + \epsilon$ achieved by a local search heuristic of Arya, et al. [6] (see also a separate entry *k-median and Facility Location*).

The first constant factor approximation algorithm for the *k*-center problem was given by Hochbaum and Shmoys [26], who developed a 2-approximation algorithm. This performance guarantee is the best possible unless $P = NP$.

Capacitated Facility Location

For the soft-capacitated problem with equal capacities, the first constant factor approximation algorithms are due to Shmoys et al. [51] for both the splittable and unsplittable demand cases, see above. Recently, a 2-approximation algorithm for the soft capacitated facility location problem with unsplittable unit demands was proposed by Mahdian et al. [39]. The integrality gap of the LP relaxation for the problem is also 2. Hence, to improve the approximation guarantee one would have to develop a better lower bound on the optimal solution.

In the hard capacities version it is important to allow for splitting the demands, as otherwise even the feasibility problem becomes difficult. Suppose demands are splittable, then we may to distinguish between the equal capacity case, where $u_i = u$ for all $i \in \mathcal{F}$, and the general case. For the problem with equal capacities, a 5.83-approximation algorithm was given by Chudak and Wiliamson [14]. The first constant factor approximation algorithm, with $\gamma = 8.53 + \epsilon$, for general capacities was given by Pál, Tardos, and Wexler [44]. This was later improved by Zhang, Chen, and Ye [57] who obtained a 5.83-approximation algorithm also for general capacities.

k-Level Problem

The first constant factor approximation algorithm for $k = 2$ is due to Shmoys et al. [51], with $\gamma = 3.16$. For general k, the first algorithm, having $\gamma = 3$, was proposed by Aardal, Chudak, and Shmoys [1]. For $k = 2$, Zhang [56] developed a 1.77-approximation algorithm. He also showed that the problem for $k = 3$ and $k = 4$ can be approximated by $\gamma = 2.523$ (This value of γ deviates slightly from the value 2.51 given in the paper. The original argument contained a minor calculation error.) and $\gamma = 2.81$ respectively.

Applications

Facility location has numerous applications in the field of operations research. See the book edited by Mirchandani and Francis [42] or the book by Nemhauser and Wolsey [43] for a survey and a description of applications of facility location in problems such as plant location and locating bank accounts. Recently, the problem has found new applications in network design problems such as placement of routers and caches [22, 36], agglomeration of traffic or data [4, 21], and web server replications in a content distribution network [31, 45].

Open Problems

A major open question is to determine the exact approximability threshold of UFL and close the gap between the upper bound of 1.5 [10] and the lower bound of 1.463 [20]. Another important question is to find better approximation algorithms for *k*-median. In particular, it would be interesting to find an LP-based 2-approximation algorithm for *k*-median. Such an algorithm would determine the integrality gap of the natural LP relaxation of this problem, as there are simple examples that show that this gap is at least 2.

Experimental Results

Jain et al. [28] published experimental results comparing various primal-dual algorithms. A more comprehensive experimental study of several primal-dual, local search, and heuristic algorithms is performed by Hoefer [27]. A collection of data sets for UFL and several other location problems can be found in the OR-library maintained by Beasley [9].

Cross-References

- ▸ Assignment Problem
- ▸ Bin Packing (hardness of Capacitated Facility Location with unsplittable demands)
- ▸ Circuit Placement
- ▸ Greedy Set-Cover Algorithms (hardness of a variant of UFL, where facilities may be built at all locations with the same cost)
- ▸ Local Approximation of Covering and Packing Problems
- ▸ Local Search for *K*-medians and Facility Location

Recommended Reading

1. Aardal K, Chudak FA, Shmoys DB (1999) A 3-approximation algorithm for the *k*-level uncapacitated facility location problem. Inf Process Lett 72:161–167
2. Ageev AA, Sviridenko MI (1999) An 0.828-approximation algorithm for the uncapacitated facility location problem. Discret Appl Math 93:149–156
3. Anagnostopoulos A, Bent R, Upfal E, van Hentenryck P (2004) A simple and deterministic competitive algorithm for online facility location. Inf Comput 194(2):175–202
4. Andrews M, Zhang L (1998) The access network design problem. In: Proceedings of the 39th annual IEEE symposium on foundations of computer science (FOCS). IEEE Computer Society, Los Alamitos, pp 40–49
5. Arora S, Raghavan P, Rao S (1998) Approximation schemes for Euclidean *k*-medians and related problems. In: Proceedings of the 30th annual ACM symposium on theory of computing (STOC). ACM, New York, pp 106–113
6. Arya V, Garg N, Khandekar R, Meyerson A, Munagala K, Pandit V (2001) Local search heuristics for k-median and facility location problems. In: Proceedings of the 33rd annual ACM symposium on theory of computing (STOC). ACM, New York, pp 21–29
7. Balinski ML (1966) On finding integer solutions to linear programs. In: Proceedings of the IBM scientific computing symposium on combinatorial problems. IBM, White Plains, pp 225–248
8. Balinski ML, Wolfe P (1963) On Benders decomposition and a plant location problem. In: ARO-27. Mathematica Inc., Princeton
9. Beasley JE (2008) Operations research library. http://people.brunel.ac.uk/~mastjjb/jeb/info.html. Accessed 2008
10. Byrka J (2007) An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. In: Proceedings of the 10th international workshop on approximation algorithms for combinatorial optimization problems (APPROX). Lecture notes in computer science, vol 4627. Springer, Berlin, pp 29–43
11. Charikar M, Guha S, Tardos E, Shmoys DB (1999) A constant factor approximation algorithm for the k-median problem. In: Proceedings of the 31st annual ACM symposium on theory of computing (STOC). ACM, New York, pp 1–10
12. Charikar M, Khuller S, Mount D, Narasimhan G (2001) Facility location with outliers. In: Proceedings of the 12th annual ACM-SIAM symposium on discrete algorithms (SODA). SIAM, Philadelphia, pp 642–651
13. Chudak FA, Shmoys DB (2003) Improved approximation algorithms for the uncapacitated facility location problem. SIAM J Comput 33(1):1–25
14. Chudak FA, Wiliamson DP (1999) Improved approximation algorithms for capacitated facility location problems. In: Proceedings of the 7th conference on integer programing and combinatorial optimization (IPCO). Lecture notes in computer science, vol 1610. Springer, Berlin, pp 99–113
15. Cornuéjols G, Fisher ML, Nemhauser GL (1977) Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms. Manag Sci 8:789–810
16. Erlenkotter D (1978) A dual-based procedure for uncapacitated facility location problems. Oper Res 26:992–1009
17. Feige U (1998) A threshold of ln n for approximating set cover. J ACM 45:634–652
18. Fotakis D (2003) On the competitive ratio for online facility location. In: Proceedings of the 30th international colloquium on automata, languages and programming (ICALP). Lecture notes in computer science, vol 2719. Springer, Berlin, pp 637–652
19. Guha S, Khuller S (1998) Greedy strikes back: improved facility location algorithms. In: Proceedings of the 9th ACM-SIAM symposium on discrete algorithms (SODA). SIAM, Philadelphia, pp 228–248
20. Guha S, Khuller S (1999) Greedy strikes back: improved facility location algorithms. J Algorithm 31:228–248
21. Guha S, Meyerson A, Munagala K (2001) A constant factor approximation for the single sink edge installation problem. In: Proceedings of the 33rd annual ACM symposium on theory of computing (STOC). ACM, New York, pp 383–388
22. Guha S, Meyerson A, Munagala K (2000) Hierarchical placement and network design problems. In: Proceedings of the 41st annual IEEE symposium on foundations of computer science (FOCS). IEEE Computer Society, Los Alamitos, pp 603–612
23. Gupta A, Pál M, Ravi R, Sinha A (2004) Boosted sampling: approximation algorithms for stochastic optimization. In: Proceedings of the 36st annual ACM symposium on theory of computing (STOC). ACM, New York, pp 417–426

24. Hajiaghayi M, Mahdian M, Mirrokni VS (2003) The facility location problem with general cost functions. Networks 42(1):42–47
25. Hochbaum DS (1982) Heuristics for the fixed cost median problem. Math Program 22(2):148–162
26. Hochbaum DS, Shmoys DB (1985) A best possible approximation algorithm for the *k*-center problem. Math Oper Res 10:180–184
27. Hoefer M (2003) Experimental comparison of heuristic and approximation algorithms for uncapacitated facility location. In: Proceedings of the 2nd international workshop on experimental and efficient algorithms (WEA). Lecture notes in computer science, vol 2647. Springer, Berlin, pp 165–178
28. Jain K, Mahdian M, Markakis E, Saberi A, Vazirani VV (2003) Approximation algorithms for facility location via dual fitting with factor-revealing LP. J ACM 50(6):795–824
29. Jain K, Mahdian M, Saberi A (2002) A new greedy approach for facility location problems. In: Proceedings of the 34st annual ACM symposium on theory of computing (STOC). ACM, New York, pp 731–740
30. Jain K, Vazirani VV (2000) An approximation algorithm for the fault tolerant metric facility location problem. In: Approximation algorithms for combinatorial optimization, proceedings of APPROX. Lecture notes in computer science, vol 1913. Springer, Berlin, pp 177–183
31. Jamin S, Jin C, Jin Y, Raz D, Shavitt Y, Zhang L (2000) On the placement of internet instrumentations. In: Proceedings of the 19th annual joint conference of the IEEE computer and communications Societies (INFOCOM), vol 1. IEEE Computer Society, Los Alamitos, pp 295–304
32. Karger D, Minkoff M (2000) Building Steiner trees with incomplete global knowledge. In: Proceedings of the 41st annual IEEE symposium on foundations of computer science (FOCS). IEEE Computer Society, Los Alamitos, pp 613–623
33. Krarup J, Pruzan PM (1990) Ingredients of locational analysis. In: Mirchandani P, Francis R (eds) Discrete location theory. Wiley, New York, pp 1–54
34. Krarup J, Pruzan PM (1983) The simple plant location problem: survey and synthesis. Eur J Oper Res 12:38–81
35. Kuehn AA, Hamburger MJ (1963) A heuristic program for locating warehouses. Manag Sci 9:643–666
36. Li B, Golin M, Italiano G, Deng X, Sohraby K (1999) On the optimal placement of web proxies in the internet. In: Proceedings of the 18th annual joint conference of the IEEE computer and communications societies (INFOCOM). IEEE Computer Society, Los Alamitos, pp 1282–1290
37. Mahdian M (2004) Facility location and the analysis of algorithms through factor-revealing programs. PhD thesis, MIT, Cambridge
38. Mahdian M, Pál M (2003) Universal facility location. In: Proceedings of the 11th annual european symposium on algorithms (ESA). Lecture notes in computer science, vol 2832. Springer, Berlin, pp 409–421
39. Mahdian M, Ye Y, Zhang J (2006) Approximation algorithms for metric facility location problems. SIAM J Comput 36(2):411–432
40. Manne AS (1964) Plant location under economies-of-scale – decentralization and computation. Manag Sci 11:213–235
41. Meyerson A (2001) Online facility location. In: Proceedings of the 42nd annual IEEE symposium on foundations of computer science (FOCS). IEEE Computer Society, Los Alamitos, pp 426–431
42. Mirchandani PB, Francis RL (1990) Discrete location theory. Wiley, New York
43. Nemhauser GL, Wolsey LA (1990) Integer and combinatorial optimization. Wiley, New York
44. Pál M, Tardos E, Wexler T (2001) Facility location with nonuniform hard capacities. In: Proceedings of the 42nd annual IEEE symposium on foundations of computer science (FOCS). IEEE Computer Society, Los Alamitos, pp 329–338
45. Qiu L, Padmanabhan VN, Voelker G (2001) On the placement of web server replicas. In: Proceedings of the 20th annual joint conference of the IEEE computer and communications societies (INFOCOM). IEEE Computer Society, Los Alamitos, pp 1587–1596
46. Ravi R, Sinha A (2006) Hedging uncertainty: approximation algorithms for stochastic optimization problems. Math Program 108(1):97–114
47. Ravi R, Sinha A (2002) Integrated logistics: approximation algorithms combining facility location and network design. In: Proceedings of the 9th conference on integer programming and combinatorial optimization (IPCO). Lecture notes in computer science, vol 2337. Springer, Berlin, pp 212–229
48. Ravi R, Sinha A (2004) Multicommodity facility location. In: Proceedings of the 15th annual ACM-SIAM symposium on discrete algorithms (SODA). SIAM, Philadelphia, pp 342–349
49. Shmoys DB (2000) Approximation algorithms for facility location problems. In: Jansen K, Khuller S (eds) Approximation algorithms for combinatorial optimization, vol 1913, Lecture notes in computer science. Springer, Berlin, pp 27–33
50. Shmoys DB (2004) The design and analysis of approximation algorithms: facility location as a case study. In: Thomas RR, Hosten S, Lee J (eds) Proceedings of symposia in applied mathematics, vol 61. AMS, Providence, pp 85–97
51. Shmoys DB, Tardos E, Aardal K (1997) Approximation algorithms for facility location problems. In: Proceedings of the 29th annual ACM symposium on theory of computing (STOC). ACM, New York, pp 265–274
52. Svitkina Z, Tardos E (2006) Facility location with hierarchical facility costs. In: Proceedings of the 17th annual ACM-SIAM symposium on discrete algorithm (SODA). SIAM, Philadelphia, pp 153–161
53. Swamy C, Kumar A (2004) Primal-dual algorithms for connected facility location problems. Algorithmica 40(4):245–269

54. Swamy C, Shmoys DB (2003) Fault-tolerant facility location. In: Proceedings of the 14th annual ACM-SIAM symposium on discrete algorithms (SODA). SIAM, Philadelphia, pp 735–736
55. Vygen J (2005) Approximation algorithms for facility location problems (lecture notes). Technical report No. 05950-OR, Research Institute for Discrete Mathematics, University of Bonn. http://www.or.uni-bonn.de/~vygen/fl.pdf
56. Zhang J (2004) Approximating the two-level facility location problem via a quasi-greedy approach. In: Proceedings of the 15th annual ACM-SIAM symposium on discrete algorithms (SODA). SIAM, Philadelphia, pp 808–817. Also, Math Program 108:159–176 (2006)
57. Zhang J, Chen B, Ye Y (2005) A multiexchange local search algorithm for the capacitated facility location problem. Math Oper Res 30(2):389–403

Failure Detectors

Rachid Guerraoui
School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland

Keywords

Distributed oracles; Failure information; Partial synchrony; Time-outs

Years and Authors of Summarized Original Work

1996; Chandra, Toueg

Problem Definition

A distributed system is comprised of a collection of processes. The processes typically seek to achieve some common task by communicating through message passing or shared memory. Most interesting tasks require, at least at certain points of the computation, some form of *agreement* between the processes. An abstract form of such agreement is *consensus* where processes need to agree on a single value among a set of proposed values. Solving this seemingly elementary problem is at the heart of reliable distributed computing and, in particular, of distributed database commitment, total ordering of messages, and emulations of many shared object types.

Fischer, Lynch, and Paterson's seminal result in the theory of distributed computing [13] says that consensus cannot be deterministically solved in an *asynchronous* distributed system that is prone to process failures. This impossibility holds consequently for all distributed computing problems which themselves rely on consensus.

Failures and *asynchrony* are fundamental ingredients in the consensus impossibility. The impossibility holds even if only *one* process *fails*, and it does so only by *crashing*, i.e., stopping its activities. Tolerating crashes is the least one would expect from a distributed system for the goal of distribution is in general to avoid single points of failures in centralized architectures. Usually, actual distributed applications exhibit more severe failures where processes could deviate arbitrarily from the protocol assigned to them.

Asynchrony refers to the absence of assumptions on process speeds and communication delays. This absence prevents any process from distinguishing a crashed process from a correct one and this inability is precisely what leads to the consensus impossibility. In practice, however, distributed systems are not completely asynchronous: some timing assumptions can typically be made. In the best case, if precise lower and upper bounds on communication delays and process speeds are assumed, then it is easy to show that consensus and related impossibilities can be circumvented despite the crash of any number of processes [20].

Intuitively, the way that such timing assumptions circumvent asynchronous impossibilities is by providing processes with *information about failures*, typically through *time-out* (or *heart-beat*) mechanisms, usually underlying actual distributed applications. Whereas certain information about failures can indeed be obtained in distributed systems, the accuracy of such information might vary from a system to another, depending on the underlying network, the load

of the application, and the mechanisms used to detect failures. A crucial problem in this context is to characterize such information, in an abstract and precise way.

Key Results

The Failure Detector Abstraction

Chandra and Toueg [5] defined the *failure detector* abstraction as a simple way to capture failure information that is needed to circumvent asynchronous impossibilities, in particular the consensus impossibility. The model considered in [5] is a message passing one where processes can fail by *crashing*. Processes that crash stop their activities and do not recover. Processes that do not crash are said to be *correct*. At least one process is supposed to be correct in every execution of the system.

Roughly speaking, a failure detector is an oracle that provides processes with information about failures. The oracle is accessed in each computation step of a process and it provides the process with a value conveying some failure information. The value is picked from some set of values, called the *range* of the failure detector. For instance, the range could be the set of subsets of processes in the system, and each subset could depict the set of processes detected to have crashed, or considered to be correct. This would correspond to the situation where the failure detector is implemented using a time-out: every process q that does not communicate within some time period with some process p, would be included in subset of processes suspected of having crashed by p.

More specifically, a failure detector is a function, D, that associates to each *failure pattern*, F, a set of *failure detector histories* $\{H_i\} = D(F)$. Both the failure pattern and the failure detector history are themselves functions.

- A failure pattern F is a function that associates to each time t, the set of processes $F(t)$ that have indeed crashed by time t. This notion assumes the existence of a global clock, outside the control of the processes, as well as a specific concept of *crash* event associated with time. A set of failure pattern is called an *environment*.
- A failure detector history H is also a function, which associates to each process p and time t, some value v from the range of failure detector values. (The range of a failure detector D is denoted R_D.) This value v is said to be output by the failure detector D at process p and time t.

Two observations are in order.

- By construction, the output of a failure detector does not depend on the computation, i.e., on the actual steps performed by the processes, on their algorithm or the input of such algorithm. The output of the failure detector depends solely on the failure pattern, namely on whether and when processes crashed.
- A failure detector might associate several histories to each failure pattern. Each history represents a suite of possible combinations of outputs for the same given failure pattern. This captures the inherent non-determinism of a failure detection mechanism. Such a mechanism is typically itself implemented as a distributed algorithm and the variations in communication delays for instance could lead the same mechanism to output (even slightly) different information for the same failure pattern.

To illustrate these concepts, consider two classical examples of failure detectors.

1. The *perfect* failure detector outputs a subset of processes, i.e., the range of the failure detector is the set of subsets of processes in the system. When a process q is output at some time t at a process p, then q is said to be *detected* (of having crashed) by p. The *perfect* failure detector guarantees the two following properties:
   - Every process that crashes is eventually permanently detected;
   - No correct process is ever detected.

2. The *eventually strong* failure detector outputs a subset of processes: when a process q is output at some time t at a process p, then q is said to be *suspected* (of having crashed) by p. An *eventually strong* failure detector ensures the two following properties:
   - Every process that crashes is eventually suspected;
   - Eventually, some correct process is never suspected.

The *perfect* failure detector is *reliable*: if a process q is detected, then q has crashed. An *eventually strong* failure detector is *unreliable*: there never is any guarantee that the information that is output is accurate. The use of the term *suspected* conveys that idea. The distinction between *unreliability* and *reliability* was precisely captured in [14] for the general context where the range of the failure detector can be arbitrary.

Consensus Algorithms

Two important results were established in [5].

Theorem 1 (Chandra-Toueg [5]) *There is an algorithm that solves consensus with a perfect failure detector.*

The theorem above implicitly says that if the distributed system provides means to implement perfect failure detection, then the consensus impossibility can be circumvented, even if all but one process crashes. In fact, the result holds for any failure pattern, i.e., in any environment.

The second theorem below relates the existence of a consensus algorithm to a resilience assumption. More specifically, the theorem holds in the *majority* environment, which is the set of failure patterns where more than half of the processes are correct.

Theorem 2 (Chandra-Toueg [5]) *There is an algorithm that implements consensus with an eventually strong failure detector in the majority environment.*

The algorithm underlying the result above is similar to *eventually synchronous* consensus algorithms [10] and share also some similarities with the *Paxos* algorithm [18]. It is shown in [5] that no algorithm using solely the *eventually strong* failure detector can solve consensus without the majority assumption. (This result is generalized to any unreliable failure detector in [14].) This resilience lower bound is intuitively due to the possibility of partitions in a message passing system where at least half of the processes can crash and failure detection is unreliable. In shared memory for example, no such possibility exists and consensus can be solved with the *eventually strong* failure [19].

Failure Detector Reductions

Failure detectors can be compared. A failure detector D_2 is said to be *weaker* than a failure detector D_1 if there is an asynchronous algorithm, called a *reduction* algorithm, which, using D_1, can emulate D_2. Three remarks are important here.

- The fact that the reduction algorithm is asynchronous means that it does not use any other source of failure information, besides D_1.
- Emulating failure detector D_2 means implementing a distributed variable that mimics the output that could be provided by D_2.
- The existence of a reduction algorithm depends on environment. Hence, strictly speaking, the fact that a failure detector is weaker than another one depends on the environment under consideration.

If failure detector D_1 is weaker than D_2, and vice et versa, then D_1 and D_2 are said to be *equivalent*. Else, if D_1 is weaker than D_2 and D_2 is not weaker than D_1, then D_1 is said to be *strictly weaker* than D_2. Again, strictly speaking, these notions depend on the considered environment.

The ability to compare failure detectors help define a notion of *weakest* failure detector to solve a problem. Basically, a failure detector D is the weakest to solve a problem P if the two following properties are satisfied:

- There is an algorithm that solves P using D.
- If there is an algorithm that solves P using some failure detector D', then D is weaker than D'.

Theorem 3 (Chandra-Hadzilacos-Toueg [4]) *The eventually strong failure detector is the weakest to solve consensus in the majority environment.*

The weakest failure detector to implement consensus in any environment was later established in [8].

Applications

A Practical Perspective

The identification of the failure detector concept had an impact on the design of reliable distributed architectures. Basically, a failure detector can be viewed as a first class service of a distributed system, at the same level as a name service or a file service. Time-out and heartbeat mechanisms can thus be hidden under the failure detector abstraction, which can then export a unified interface to higher level applications, including consensus and state machine replication algorithms [2, 11, 21].

Maybe more importantly, a failure detector service can encapsulate synchrony assumptions: these can be changed without impact on the rest of the applications. Minimal synchrony assumptions to devise specific failure detectors could be explored leading to interesting theoretical results [1, 7, 12].

A Theoretical Perspective

A second application of the failure detector concept is a theory of distributed computability. Failure detectors enable to classify problems. A problem A is *harder* (resp. *strictly harder*) than problem B if the weakest failure detector to solve B is weaker (resp. strictly weaker) than the weakest failure detector to solve A. (This notion is of course parametrized by a specific environment.)

Maybe surprisingly, the induced failure detection reduction between problems does not exactly match the classical *black-box* reduction notion. For instance, it is well known that there is no asynchronous distributed algorithm that can use a *Queue* abstraction to implement a *Compare-Swap* abstraction in a system of $n > 2$ processes where $n - 1$ can fail by crashing [15]. In this sense, a *Compare-Swap* abstraction is strictly more powerful (in a *black-box* sense) than a *Queue* abstraction. It turns out that:

Theorem 4 (Delporte-Fauconnier-Guerraoui [9]) *The weakest failure detector to solve the Queue problem is also the weakest to solve the Compare-Swap problem in a system of $n > 2$ processes where $n - 1$ can fail by crashing.*

In a sense, this theorem indicates that reducibility as induced by the failure detector notion is different from the traditional *black-box* reduction.

Open Problems

Several issues underlying the failure detector notion are still open. One such issue consists in identifying the weakest failure detector to solve the seminal *set-agreement* problem [6]: a decision task where processes need to agree on up to k values, instead of a single value as in consensus. Three independent groups of researchers [3, 16, 22] proved the impossibility of solving this problem in an asynchronous system with k failures, generalizing the consensus impossibility [13]. Determining the weakest failure detector to circumvent this impossibility would clearly help understand the fundamentals of failure detection reducibility.

Another interesting research direction is to relate the complexity of distributed algorithm with the underlying failure detector [17]. Clearly, failure detectors circumvents asynchronous impossibilities, but to what extend do they boost the complexity of distributed algorithms? One would of course expect the complexity of a solution to a problem to be higher if the failure detector is weaker. But to what extend?

Cross-References

▸ Asynchronous Consensus Impossibility
▸ Atomic Broadcast

- Causal Order, Logical Clocks, State Machine Replication
- Linearizability

Recommended Reading

1. Aguilera MK, Delporte-Gallet C, Fauconnier H, Toueg S (2003) On implementing omega with weak reliability and synchrony assumptions. In: 22th ACM symposium on principles of distributed computing, pp 306–314
2. Bertier M, Marin O, Sens P (2003) Performance analysis of a hierarchical failure detector. In: Proceedings 2003 international conference on dependable systems and networks (DSN 2003), San Francisco, 22–25 June 2003, pp 635–644
3. Boroswsky E, Gafni E (n.d.) Generalized FLP impossibility result for *t*-resilient asynchronous computations. In: Proceedings of the 25th ACM symposium on theory of computing. ACM, pp 91–100
4. Chandra TD, Hadzilacos V, Toueg S (1996) The weakest failure detector for solving consensus. J ACM 43(4):685–722
5. Chandra TD, Toueg S (1996) Unreliable failure detectors for reliable distributed systems. J ACM 43(2):225–267
6. Chauduri S (1993) More choices allow more faults: set consensus problems in totally asynchronous systems. Inf Comput 105(1):132–158
7. Chen W, Toueg S, Aguilera MK (2002) On the quality of service of failure detectors. IEEE Trans Comput 51(1):13–32
8. Delporte-Gallet C, Fauconnier H, Guerraoui R (2002) Failure detection lower bounds on registers and consensus. In: Proceedings of the 16th international symposium on distributed computing, LNCS, vol 2508
9. Delporte-Gallet C, Fauconnier H, Guerraoui R (2005) Implementing atomic objects in a message passing system. Technical report, EPFL Lausanne
10. Dwork C, Lynch NA, Stockmeyer L (1988) Consensus in the presence of partial synchrony. J ACM 35(2):288–323
11. Felber P, Guerraoui R, Fayad M (1999) Putting oo distributed programming to work. Commun ACM 42(11):97–101
12. Fernández A, Jiménez E, Raynal M (2006) Eventual leader election with weak assumptions on initial knowledge, communication reliability and synchrony. In: Proceedings of the international symposium on dependable systems and networks (DSN), pp 166–178
13. Fischer MJ, Lynch NA, Paterson MS (1985) Impossibility of distributed consensus with one faulty process. J ACM 32(2):374–382
14. Guerraoui R (2000) Indulgent algorithms. In: Proceedings of the 19th annual ACM symposium on principles of distributed computing, ACM, Portland, pp 289–297
15. Herlihy M (1991) Wait-free synchronization. ACM Trans Program Lang Syst 13(1):123–149
16. Herlihy M, Shavit N (1993) The asynchronous computability theorem for *t*-resilient tasks. In: Proceedings of the 25th ACM symposium on theory of computing, pp 111–120
17. Keidar I, Rajsbaum S (2002) On the cost of fault-tolerant consensus when there are no faults-a tutorial. In: Tutorial 21st ACM symposium on principles of distributed computing
18. Lamport L (1998) The part-time parliament. ACM Trans Comput Syst 16(2):133–169
19. Lo W-K, Hadzilacos V (1994) Using failure detectors to solve consensus in asynchronous shared memory systems. In: Proceedings of the 8th international workshop on distributed algorithms. LNCS, vol 857, pp 280–295
20. Lynch N (1996) Distributed algorithms. Morgan Kauffman
21. Michel R, Corentin T (2006) In search of the holy grail: looking for the weakest failure detector for wait-free set agreement. Technical Report TR 06-1811, INRIA
22. Saks M, Zaharoglou F (1993) Wait-free *k*-set agreement is impossible: the topology of public knowledge. In: Proceedings of the 25th ACM symposium on theory of computing, ACM, pp 101–110

False-Name-Proof Auction

Makoto Yokoo
Department of Information Science and Electrical Engineering, Kyushu University, Nishi-ku, Fukuoka, Japan

Keywords

False-name-proof auctions; Pseudonymous bidding; Robustness against false-name bids

Years and Authors of Summarized Original Work

2004; Yokoo, Sakurai, Matsubara

Problem Definition

In Internet auctions, it is easy for a bidder to submit multiple bids under multiple identifiers (e.g., multiple e-mail addresses). If only one item/good

is sold, a bidder cannot make any additional profit by using multiple bids. However, in combinatorial auctions, where multiple items/goods are sold simultaneously, submitting multiple bids under fictitious names can be profitable. A bid made under a fictitious name is called a *false-name bid*.

Here, use the same model as the GVA section. In addition, false-name bids are modeled as follows.

- Each bidder can use multiple identifiers.
- Each identifier is unique and cannot be impersonated.
- Nobody (except the owner) knows whether two identifiers belongs to the same bidder or not.

The goal is to design a *false-name-proof protocol*, i.e., a protocol in which using false-names is useless, thus bidders voluntarily refrain from using false-names.

The problems resulting from collusion have been discussed by many researchers. Compared with collusion, a false-name bid is easier to execute on the Internet since obtaining additional identifiers, such as another e-mail address, is cheap. False-name bids can be considered as a very restricted subclass of collusion.

Key Results

The Generalized Vickrey Auction (GVA) protocol is (dominant strategy) incentive compatible, i.e., for each bidder, truth-telling is a dominant strategy (a best strategy regardless of the action of other bidders) if there exists no false-name bids. However, when false-name bids are possible, truth-telling is no longer a dominant strategy, i.e., the GVA is not false-name-proof.

Here is an example, which is identical to Example 1 in the GVA section.

Example 1 Assume there are two goods a and b, and three bidders, bidder 1, 2, and 3, whose types are θ_1, θ_2, and θ_3, respectively. The evaluation value for a bundle $v(B, \theta_i)$ is determined as follows.

	$\{a\}$	$\{b\}$	$\{a, b\}$
θ_1	\$6	\$0	\$6
θ_2	\$0	\$0	\$8
θ_3	\$0	\$5	\$5

As shown in the GVA section, good a is allocated to bidder 1, and b is allocated to bidder 3. Bidder 1 pays \$3 and bidder 3 pays \$2.

Now consider another example.

Example 2 Assume there are only two bidders, bidder 1 and 2, whose types are θ_1 and θ_2, respectively. The evaluation value for a bundle $v(B, \theta_i)$ is determined as follows.

	$\{a\}$	$\{b\}$	$\{a, b\}$
θ_1	\$6	\$5	\$11
θ_2	\$0	\$0	\$8

In this case, the bidder 1 can obtains both goods, but he/she requires to pay \$8, since if bidder 1 does not participate, the social surplus would have been \$8. When bidder 1 does participate, bidder 1 takes everything and the social surplus except bidder 1 becomes 0. Thus, bidder 1 needs to pay the decreased amount of the social surplus, i.e., \$8.

However, bidder 1 can use another identifier, namely, bidder 3 and creates a situation identical to Example 1. Then, good a is allocated to bidder 1, and b is allocated to bidder 3. Bidder 1 pays \$3 and bidder 3 pays \$2. Since bidder 3 is a false-name of bidder 1, bidder 1 can obtain both goods by paying \$3 + \$2 = \$5. Thus, using a false-name is profitable for bidder 1.

The effects of false-name bids on combinatorial auctions are analyzed in [4]. The obtained results can be summarized as follows.

- As shown in the above example, the GVA protocol is not false-name-proof.
- There exists no false-name-proof combinatorial auction protocol that satisfies Pareto efficiency.
- If a surplus function of bidders satisfies a condition called *concavity*, then the GVA is guaranteed to be false-name-proof.

Also, a series of protocols that are false-name-proof in various settings have been developed: combinatorial auction protocols [2, 3], multi-unit auction protocols [1], and double auction protocols [5].

Furthermore, in [2], a distinctive class of combinatorial auction protocols called a Price-oriented, Rationing-free (PORF) protocol is identified. The description of a PORF protocol can be used as a guideline for developing strategy/false-name proof protocols.

The outline of a PORF protocol is as follows:

1. For each bidder, the price of each bundle of goods is determined independently of his/her own declaration, while it depends on the declarations of other bidders. More specifically, the price of bundle (a set of goods) B for bidder i is determined by a function $p(B, \Theta_X)$, where Θ_X is a set of declared types by other bidders X.
2. Each bidder is allocated a bundle that maximizes his/her utility independently of the allocations of other bidders (i.e., rationing-free). The prices of bundles must be determined so that *allocation feasibility* is satisfied, i.e., no two bidders want the same item.

Although a PORF protocol appears to be quite different from traditional protocol descriptions, surprisingly, it is a sufficient and necessary condition for a protocol to be strategy-proof. Furthermore, if a PORF protocol satisfies the following additional condition, it is guaranteed to be false-name-proof.

Definition 1 (No Super-Additive price increase (NSA)) For any subset of bidders $S \subseteq N$ and $N' = N \setminus S$, and for $i \in S$, denote B_i as a bundle that maximizes i's utility, then $\sum_{i \in S} p(B_i, \bigcup_{j \in S \setminus \{i\}} \{\theta_j\} \cup \Theta_{N'}) \geq p(\bigcup_{i \in S} B_i, \Theta_{N'})$.

An intuitive description of this condition is that the price of buying a combination of bundles (the right side of the inequality) must be smaller than or equal to the sum of the prices for buying these bundles separately (the left side). This condition is also a necessary condition for a protocol to be false-name-proof, i.e., any false-name-proof protocol can be described as a PORF protocol that satisfies the NSA condition.

Here is a simple example of a PORF protocol that is false-name-proof. This protocol is called the Max Minimal-Bundle (M-MB) protocol [2]. To simplify the protocol description, a concept called a *minimal* bundle is introduced.

Definition 2 (minimal bundle) Bundle B is called minimal for bidder i, if for all $B' \subset B$ and $B' \neq B$, $v(B', \theta_i) < v(B, \theta_i)$ holds.

In this new protocol, the price of bundle B for bidder i is defined as follows:

- $p(B, \Theta_X) = \max_{B_j \subseteq M, j \in X} v(B_j, \theta_j)$, where $B \cap B_j \neq \emptyset$ and B_j is minimal for bidder j.

How this protocol works using Example 1 is described here. The prices for each bidder is determined as follows.

	$\{a\}$	$\{b\}$	$\{a, b\}$
bidder 1	\$8	\$8	\$8
bidder 2	\$6	\$5	\$6
bidder 3	\$8	\$8	\$8

The minimal bundle for bidder 1 is $\{a\}$, the minimal bundle for bidder 2 is $\{a, b\}$, and the minimal bundle for bidder 3 is $\{b\}$. The price of bundle $\{a\}$ for bidder 1 is equal to the largest evaluation value of conflicting bundles. In this case, the price is \$8, i.e., the evaluation value of bidder 2 for bundle $\{a, b\}$. Similarly, the price of bidder 2 for bundle $\{a, b\}$ is 6, i.e., the evaluation value of bidder 1 for bundle $\{a\}$. As a result, bundle $\{a, b\}$ is allocated to bidder 2.

It is clear that this protocol satisfies the allocation feasibility. For each good l, choose bidder j^* and bundle B_j^* that maximize $v(B_j, \theta_j)$ where $l \in B_j$ and B_j is minimal for bidder j. Then, only bidder j^* is willing to obtain a bundle that contains good l. For all other bidders, the price of a bundle that contains l is higher than (or equal to) his/her evaluation value.

Furthermore, it is clear that this protocol satisfies the NSA condition. In this pricing scheme, $p(B \cup B', \Theta_X) = \max(p(B, \Theta_X), p(B', \Theta_X))$ holds for all B, B', and Θ_X. Therefore, the following formula holds

$$p\left(\bigcup_{i \in S} B_i, \Theta_X\right) = \max_{i \in S} p(B_i, \Theta_X) \leq \sum_{i \in S} p(B_i, \Theta_X).$$

Furthermore, in this pricing scheme, prices increase monotonically by adding opponents, i.e., for all $X' \supseteq X$, $p(B, \Theta_{X'}) \geq p(B, \Theta_X)$ holds. Therefore, for each i, $p(B_i, \bigcup_{j \in S \setminus \{i\}} \{\theta_j\} \cup \Theta_{N'}) \geq p(B_i, \Theta_{N'})$ holds. Therefore, the NSA condition, i.e., $\sum_{i \in S} p(B_i, \bigcup_{j \in S \setminus \{i\}} \{\theta_j\} \cup \Theta_{N'}) \geq p(\bigcup_{i \in S} B_i, \Theta_{N'})$ holds.

Applications

In Internet auctions, using multiple identifiers (e.g., multiple e-mail addresses) is quite easy and identifying each participant on the Internet is virtually impossible. Combinatorial auctions have lately attracted considerable attention. When combinatorial auctions become widely used in Internet auctions, false-name-bids could be a serious problem.

Open Problems

It is shown that there exists no false-name-proof protocol that is Pareto efficient. Thus, it is inevitable to give up the efficiency to some extent. However, the theoretical lower-bound of the efficieny loss, i.e., the amount of the efficiency loss that is inevitabe for any false-name-proof protocol, is not identified yet. Also, the efficiency loss of existing false-name-proof protocols can be quite large. More efficient false-name-proof protocols in various settings are needed.

Cross-References

▸ Generalized Vickrey Auction

Recommended Reading

1. Iwasaki A, Yokoo M, Terada K (2005) A robust open ascending-price multi-unit auction protocol against false-name bids. Decis Support Syst 39:23–39
2. Yokoo M (2003) The characterization of strategy/false-name proof combinatorial auction protocols: price-oriented, rationing-free protocol. In: Proceedings of the 18th international joint conference on artificial intelligence, pp 733–739
3. Yokoo M, Sakurai Y, Matsubara S (2001) Robust combinatorial auction protocol against false-name bids. Artif Intell 130:167–181
4. Yokoo M, Sakurai Y, Matsubara S (2004) The effect of false-name bids in combinatorial auctions: new fraud in internet auctions. Game Econ Behav 46:174–188
5. Yokoo M, Sakurai Y, Matsubara S (2005) Robust double auction protocol against false-name bids. Decis Support Syst 39:23–39

Fast Minimal Triangulation

Yngve Villanger
Department of Informatics, University of Bergen, Bergen, Norway

Keywords

Minimal fill problem

Years and Authors of Summarized Original Work

2005; Heggernes, Telle, Villanger

Problem Definition

Minimal triangulation is the addition of an inclusion minimal set of edges to an arbitrary undirected graph, such that a chordal graph is obtained. A graph is *chordal* if every cycle of length at least 4 contains an edge between two nonconsecutive vertices of the cycle.

More formally, Let $G = (V, E)$ be a simple and undirected graph, where $n = |V|$ and $m = |E|$. A graph $H = (V, E \cup F)$, where

$E \cap F = \emptyset$ is a *triangulation* of G if H is chordal, and H is a *minimal* triangulation if there exists no $F' \subset F$, such that $H' = (V, E \cup F')$ is chordal. Edges in F are called *fill edges*, and a triangulation is minimal if and only if the removal of any single fill edge results in a chordless four cycle [10].

Since minimal triangulations were first described in the mid-1970s, a variety of algorithms have been published. A complete overview of these along with different characterizations of chordal graphs and minimal triangulations can be found in the survey of Heggernes et al. [5] on minimal triangulations. Minimal triangulation algorithms can roughly be partitioned into algorithms that obtain the triangulation through elimination orderings, and those that obtain it through vertex separators. Most of these algorithms have an $O(nm)$ running time, which becomes $O(n^3)$ for dense graphs. Among those that use elimination orderings, Kratsch and Spinrad's $O(n^{2.69})$-time algorithm [8] is currently the fastest one. The fastest algorithm is an $o(n^{2.376})$-time algorithm by Heggernes et al. [5]. This algorithm is based on vertex separators, and will be discussed further in the next section. Both the algorithm of Kratsch and Spinrad [8] and the algorithm of Heggernes et al. [5] use the matrix multiplication algorithm of Coppersmith and Winograd [3] to obtain an $o(n^3)$-time algorithm.

Key Results

For a vertex set $A \subset V$, the subgraph of G induced by A is $G[A] = (A, W)$, where $uv \in W$ if $u, v \in A$ and $uv \in E\}$). The closed neighborhood of A is $N[A] = U$, where $u, v \in U$ for every $uv \in E$, where $u \in A\}$ and $N(A) = N[A] \setminus A$. A is called a *clique* if $G[A]$ is a complete graph. A vertex set $S \subset V$ is called a *separator* if $G[V \setminus S]$ is disconnected, and S is called a *minimal* separator if there exists a pair of vertices $a, b \in V \setminus S$ such that a, b are contained in different connected components of $G[V \setminus S]$, and in the same connected component of $G[V \setminus S']$ for any $S' \subset S$. A vertex set $\Omega \subseteq V$ is a *potential maximal clique* if there exists no connected component of $G[V \setminus \Omega]$ that contains Ω in its neighborhood, and for every vertex pair $u, v \in \Omega$, uv is an edge or there exists a connected component of $G[V \setminus \Omega]$ that contains both u and v in its neighborhood.

From the results in [1, 7], the following recursive minimal triangulation algorithm is obtained. Find a vertex set A which is either a minimal separator or a potential maximal clique. Complete $G[A]$ into a clique. Recursively for each connected component C of $G[V \setminus A]$ where $G[N[C]]$ is not a clique, find a minimal triangulation of $G[N[C]]$. An important property here is that the set of connected components of $G[V \setminus A]$ defines independent minimal triangulation problems.

The recursive algorithm just described defines a tree, where the given input graph G is the root node, and where each connected component of $G[V \setminus A]$ becomes a child of the root node defined by G. Now continue recursively for each of the subproblems defined by these connected components. A node H which is actually a subproblem of the algorithm is defined to be at *level* i, if the distance from H to the root in the tree is i. Notice that all subproblems at the same level can be triangulated independently. Let k be the number of levels. If this recursive algorithm can be completed for every subgraph at each level in $O(f(n))$ time, then this trivially provides an $O(f(n) \cdot k)$-time algorithm.

The algorithm in Fig. 1 uses queues to obtain this level-by-level approach, and matrix multiplication to complete all the vertex separators at a given level in $O(n^\alpha)$ time, where $\alpha < 2.376$ [3]. In contrast to the previously described recursive algorithm, the algorithm in Fig. 1 uses a partitioning subroutine that either returns a *set* of minimal separators or a potential maximal clique.

Even though all subproblems at the same level can be solved independently they may share vertices and edges, but no nonedges (i.e., pair of vertices that are not adjacent). Since triangulation involves edge addition, the number of nonedges will decrease for each level, and the sum of nonedges for all subproblems at the same level will never exceed n^2. The partitioning algorithm

Algorithm FMT - Fast Minimal Triangulation
Input: An arbitrary graph $G = (V,E)$.
Output: A minimal triangulation G' of G.

Let Q_1, Q_2 and Q_3 be empty queues; Insert G into Q_1; $G' = G$;
repeat
  Construct a zero matrix M with a row for each vertex in V (columns are added later);
  while Q_1 is nonempty **do**
    Pop a graph $H = (U, D)$ from Q_1;
    Call **Algorithm Partition**(H) which returns a vertex subset $A \subset U$;
    Push vertex set A onto Q_3;
    for each connected component C of $H[U \setminus A]$ **do**
      Add a column in M such that $M(v, C) = 1$ for all vertices $v \in N_H(C)$;
      if there exists a non-edge uv in $H[N_H[C]]$ with $u \in C$ **then**
        Push $H_C = (N_H[C], D_C)$ onto Q_2, where $uv \notin D_C$ if $u \in C$ and $uv \notin D$;
  Compute MM^T;
  Add to G' the edges indicated by the nonzero elements of MM^T;
  while Q_3 is nonempty **do**
    Pop a vertex set A from Q_3;
    if $G'[A]$ is not complete **then** Push $G'[A]$ onto Q_2;
  Swap names of Q_1 and Q_2;
until Q_1 is empty

Fast Minimal Triangulation, Fig. 1 Fast minimal triangulation algorithm

in Fig. 2 exploits this fact and has an $O(n^2 - m)$ running time, which sums up to $O(n^2)$ for each level. Thus, each level in the fast minimal triangulation algorithm given in Fig. 1 can be completed in $O(n^2 + n^\alpha)$ time, where $O(n^\alpha)$ is the time needed to compute MM^T. The partitioning algorithm in Fig. 2 actually finds a set A that defines a set of minimal separators, such that no subproblem contains more than four fifths of the nonedges in the input graph. As a result, the number of levels in the fast minimal triangulation algorithm is at most $\log_{4/5}(n^2) = 2\log_{4/5}(n)$, and the running time $O(n^\alpha \log n)$ is obtained.

Applications

The first minimal triangulation algorithms were motivated by the need to find good pivotal orderings for Gaussian elimination. Finding an optimal ordering is equivalent to solving the minimum triangulation problem, which is a nondeterministic polynomial-time hard problem. Since any minimum triangulation is also a minimal triangulation, and minimal triangulations can be found in polynomial time, then the set of minimal triangulations can be a good place to search for a pivotal ordering.

Probably because of the desired goal, the first minimal triangulation algorithms were based on orderings, and produced an ordering called a minimal elimination ordering. The problem of computing a minimal triangulation has received increasing attention since then, and several new applications and characterizations related to the vertex separator properties have been published. Two of the new applications are computing the tree-width of a graph, and reconstructing evolutionary history through phylogenetic trees [6]. The new separator-based characterizations of minimal triangulations have increased the knowledge of minimal triangulations [1, 7, 9]. One result based on these characterizations is an algorithm that computes

Algorithm Partition
Input: A graph $H = (U, D)$ (a subproblem popped from Q_I).
Output: A subset A of U such that either $A = N[K]$ for some connected $H[K]$ or A is a potential maximal clique of H (and G').

Part I: defining *P*
Unmark all vertices of H; $k = 1$;
while there exists an unmarked vertex u **do**
  if $\mathcal{E}_{\bar{H}}(U \setminus N_H[u]) < \frac{2}{5}|\bar{E}(H)|$ **then** Mark u as an **s**-vertex (stop vertex);
  else
    $C_k = \{u\}$; Mark u as a **c**-vertex (component vertex);
    while there exists a vertex $v \in N_H[C_k]$ which is unmarked or marked as an **s**-vertex **do**
      if $\mathcal{E}_{\bar{H}}(U \setminus N_H[C_k \cup \{v\}]) \geq \frac{2}{5}|\bar{E}(H)|$ **then**
        $C_k = C_k \cup \{v\}$; Mark v as a **c**-vertex (component vertex);
      else
        Mark v as a **p**-vertex (potential maximal clique vertex); Associate v with C_k;
    $k = k + 1$;
P = the set of all **p**-vertices and **s**-vertices;

Part II: defining *A*
if $H[U \setminus P]$ has a full component C **then** $A = N_H[C]$;
else if there exist two non-adjacent vertices u,v such that u is an **s**-vertex and v is an **s**-vertex or a **p**-vertex **then** $A = N_H[u]$;
else if there exist two non-adjacent **p**-vertices u and v,where u is associated with C_i and v is associated with C_j and $u \notin N_H(C_j)$ and $v \notin N_H(C_i)$ **then** $A = N_H[C_i \cup \{u\}]$;
else $A = P$;

Fast Minimal Triangulation, Fig. 2 Partitioning algorithm. Let $\bar{E}(H) = W$, where $uv \in W$ if $uv \notin D$ be the set of nonedges of H. Define $\mathcal{E}_{\bar{H}}(S)$ to be the sum of degrees in $\bar{H} = (U, \bar{E})$ of vertices in $S \subseteq U = V(H)$

the tree-width of a graph in polynomial time if the number of minimal separators is polynomially bounded [2]. A second application is faster exact (exponential-time) algorithms for computing the tree-width of a graph [4].

Open Problems

The algorithm described shows that a minimal triangulation can be found in $O((n^2 + n^\alpha) \log n)$ time, where $O(n^\alpha)$ is the time required to preform an $n \times n$ binary matrix multiplication. As a result, any improved binary matrix multiplication algorithm will result in a faster algorithm for computing a minimal triangulation. An interesting question is whether or not this relation goes the other way as well. Does there exist an $O((n^2 + n^\beta) f(n))$ algorithm for binary matrix multiplication, where $O(n^\beta)$ is the time required to find a minimal triangulation and $f(n) = o(n^{\alpha-2})$ or at least $f(n) = O(n)$. A possibly simpler and related question previously asked in [8] is: Is it at least as hard to compute a minimal triangulation as to determine whether a graph contains at least one triangle? A more algorithmic question is if there exists an $O(n^2 + n^\alpha)$-time algorithm for computing a minimal triangulation.

Cross-References

▸ Treewidth of Graphs

Recommended Reading

1. Bouchitté V, Todinca I (2001) Treewidth and minimum fill-in: grouping the minimal separators. SIAM J Comput 31:212–232
2. Bouchitté V, Todinca I (2002) Listing all potential maximal cliques of a graph. Theor Comput Sci 276(1–2):17–32
3. Coppersmith D, Winograd S (1990) Matrix multiplication via arithmetic progressions. J Symb Comput 9(3):251–280
4. Fomin FV, Kratsch D, Todinca I (2004) Exact (exponential) algorithms for treewidth and minimum fill-in. In: ICALP of LNCS, vol 3142. Springer, Berlin, pp 568–580
5. Heggernes P, Telle JA, Villanger Y (2005) Computing minimal triangulations in time $O(n^{\alpha} log\, n) = o(n^{2.376})$. SIAM J Discret Math 19(4):900–913
6. Huson DH, Nettles S, Warnow T (1999) Obtaining highly accurate topology estimates of evolutionary trees from very short sequences. In: RECOMB, pp 198–207
7. Kloks T, Kratsch D, Spinrad J (1997) On treewidth and minimum fill-in of asteroidal triple-free graphs. Theor Comput Sci 175:309–335
8. Kratsch D, Spinrad J (2006) Minimal fill in $O(n^{2.69})$ time. Discret Math 306(3):366–371
9. Parra A, Scheffler P (1997) Characterizations and algorithmic applications of chordal graph embeddings. Discret Appl Math 79:171–188
10. Rose D, Tarjan RE, Lueker G (1976) Algorithmic aspects of vertex elimination on graphs. SIAM J Comput 5:146–160

Fast Subset Convolution

Petteri Kaski
Department of Computer Science, School of Science, Aalto University, Helsinki, Finland
Helsinki Institute for Information Technology (HIIT), Helsinki, Finland

Keywords

Convolution; Min-sum semiring; Möbius inversion; Set covering; Set packing; Set partitioning; Sum-product ring; Zeta transform

Years and Authors of Summarized Original Work

2007; Björklund, Husfeldt, Kaski, Koivisto

Problem Definition

A basic strategy to solve hard problems by dynamic programming is to express the partial solutions using a recurrence over the 2^n subsets of an n-element set U. Our interest here is in recurrences that have the following structure:

> For each subset $S \subseteq U$, in order to obtain the partial solution at S, we consider all possible ways to *partition* S into two disjoint parts, T and $S \setminus T$, with $T \subseteq S$.

Fast subset convolution [1] is a technique to speed up the evaluation of such recurrences, assuming the recurrence can be reduced to a suitable algebraic form. In more precise terms, let R be an algebraic ring, such as the integers equipped with the usual arithmetic operations (addition, negation, multiplication). We seek a fast solution to:

Problem (Subset Convolution)

INPUT: *Two functions* $f : 2^U \to R$ *and* $g : 2^U \to R$.
OUTPUT: *The function* $f * g : 2^U \to R$, *defined for all* $S \subseteq U$ *by*

$$(f * g)(S) = \sum_{T \subseteq S} f(T) g(S \setminus T). \qquad (1)$$

Here, we may view the output $f * g$ and the inputs f and g each as a table with 2^n entries, where each entry is an element of R. If we evaluate the sum (1) directly for each $S \subseteq U$ in turn, in total we will execute $\Theta\left(\sum_{s=0}^{n} \binom{n}{s} 2^s\right) = \Theta(3^n)$ arithmetic operations in R to obtain $f * g$ from f and g.

Key Results

We can considerably improve on the $\Theta(3^n)$ direct evaluation by taking advantage of the ring

structure of R (i.e., the possibility to form and, after multiplication, *cancel* linear combinations of the entries in f and g):

Theorem 1 (Fast Subset Convolution [1]) *There exists an algorithm that solves* SUBSET CONVOLUTION *in* $O(2^n n^2)$ *arithmetic operations in* R.

In what follows, we present an algorithm that proceeds via reduction to the union product and fast Möbius inversion; an alternative proof is possible via reduction to the symmetric difference product and fast Walsh–Hadamard (Fourier) transforms.

Fast Evaluation via the Union Product

Let us start with a relaxed version of subset convolution. Namely, instead of *partitioning* S, we split S in all possible ways into a *cover* (A, B) with $A \cup B = S$; this cover need not be disjoint (i.e., we need not have $A \cap B = \emptyset$) as would be required by subset convolution. For f and g as earlier, define the *union product* (*covering product*) $f \cup g : 2^U \to R$ for all $S \subseteq U$ by

$$(f \cup g)(S) = \sum_{\substack{A,B \subseteq U \\ A \cup B = S}} f(A)g(B).$$

The union product diagonalizes into a pointwise product via a pair of mutually inverse linear transforms. For a given $f : 2^U \to R$, the *zeta transform* $f\zeta : 2^U \to R$ is defined for all $S \subseteq U$ by $f\zeta(S) = \sum_{T \subseteq S} f(T)$, and its inverse the *Möbius transform* $f\mu : 2^U \to R$ is defined for all $S \subseteq U$ by $f\mu(S) = (-1)^{|S|} \sum_{T \subseteq S} (-1)^{|T|} f(T)$. Using the zeta and Möbius transforms to diagonalize into a pointwise product, the union product can be evaluated as

$$f \cup g = ((f\zeta) \cdot (g\zeta))\mu. \tag{2}$$

We can now reduce subset convolution to a union product over a polynomial ring with coefficients in the original ring R. Denote by $R[w]$ the univariate polynomial ring with indeterminate w and coefficients in the ring R. Let $f, g : 2^U \to R$ be the given input to subset convolution. Extend the input $f : 2^U \to R$ to the input $f_w : 2^U \to R[w]$ defined for all $S \subseteq U$ by $f_w(S) = f(S)w^{|S|}$. Extend g similarly to g_w. Compute the union product $f_w \cup g_w$ using (2) over $R[w]$. For all $S \subseteq U$, it now holds that the coefficient of the monomial $w^{|S|}$ in the polynomial $(f_w \cup g_w)(S)$ is equal to $(f * g)(S)$.

To compute (2) fast, we require algorithms that evaluate zeta and Möbius transforms over an arbitrary ring in $O(2^n n)$ arithmetic operations. We proceed via the following recurrence for $j = 1, 2, \ldots, n$. Let us assume that $U = \{1, 2, \ldots, n\}$. Let $z_0 = f$. Suppose $z_{j-1} : 2^U \to R$ is available. Then, we compute $z_j : 2^U \to R$ for all $S \subseteq U$ by

$$z_j(S) = \begin{cases} z_{j-1}(S) & \text{if } j \notin S; \\ z_{j-1}(S) + z_{j-1}(S \setminus \{j\}) & \text{if } j \in S. \end{cases}$$

We have $f\zeta = z_n$. The recurrence carries out exactly $2^{n-1}n$ additions in R. To compute the Möbius transform $f\mu$ of a given input f, first transform the input by negating the values of all sets that have odd size, then run the previous recurrence with the transformed input as z_0, and transform the output z_n by negating the values of all sets that have odd size. The result is $f\mu$.

Remarks The fast algorithm (2) for the union product (in a dual form that considers intersections instead of unions) is due to Kennes [9], who used the algorithm to speed up an implementation of the Dempster–Shafer theory of evidence. The fast recurrences for the zeta and Möbius transforms are special cases of an algorithm of Yates [12] for multiplying a vector with an iterated Kronecker product; see Knuth [10, §4.6.4].

Extensions and Variations

A number of extensions and variations of the basic framework are possible [1]. Iterated subset convolution (union product) enables one to solve set partitioning and packing (covering) problems. Assuming the input is sparse, more careful control over the space usage of the framework can be obtained by splitting the fast zeta transform

into two parts [5]. Similarly, the running time can be controlled by *trimming* [4] the transforms, for example, to the down-closure (subset-closure) of the desired outputs and/or the up-closure (superset-closure) of the supports of the inputs in 2^U. A trimmed complementary dual to the union product is investigated in [3].

Beyond the subset lattice $(2^U, \subseteq, \cup, \cap)$, fast algorithms are known for the zeta and Möbius transforms of lattices $(L, \leq, \vee, \wedge)$ with few join-irreducible elements [6]. This implies fast analogs of the union product (the *join product*) for such lattices.

Applications

Fast subset convolution and its variants are applied to speed up dynamic programming algorithms that build up a solution from partitions of smaller solutions such that *there is little or no interaction between the parts*. Connectivity, partitioning, and subgraph counting problems on graphs are natural examples [1–3, 8, 11].

To apply fast subset convolution, it is necessary to reduce the recurrence at hand into the algebraic form (1). Let us briefly discuss two types of recurrences as examples.

Boolean Subset Convolution

Suppose that f and g are $\{0, 1\}$-valued, and we are seeking to decide whether there exists a valid partition of S into two parts so that one part is valid by f and the other part valid by g. This can be modeled as a Boolean (OR–AND) subset convolution:

$$(f *_{\vee,\wedge} g)(S) = \bigvee_{T \subseteq S} f(T) \wedge g(S \setminus T).$$

Boolean subset convolutions can be efficiently reduced into a subset convolution (1) over the integers simply by replacing the OR with a sum and the AND with multiplication.

Min-Sum Subset Convolution

Another common situation occurs when we are seeking the *minimum cost* to partition S so that the cost of one part is measured by f and the other by g, where both f and g take nonnegative integer values. This can be modeled as a min-sum subset convolution:

$$(f *_{\min,+} g)(S) = \min_{T \subseteq S} f(T) + g(S \setminus T).$$

A min-sum subset convolution over nonnegative integers can be reduced to a subset convolution (1) over a univariate polynomial ring $\mathbb{Z}[x]$ with integer coefficients. Extend $f : 2^U \to \mathbb{Z}_{\geq 0}$ to $f_x : 2^U \to \mathbb{Z}[x]$ by setting $f_x(S) = x^{f(S)}$ for all $S \subseteq U$. Extend g similarly to g_x. Now observe that the degree of the least-degree monomial with a nonzero coefficient in the polynomial $(f_x * g_x)(S)$ equals $(f *_{\min,+} g)(S)$. This reduction requires computation with polynomials of degree $O(D)$ with $D = \max\{\max_{S \subseteq U} f(S), \max_{S \subseteq U} g(S)\}$, which may not be practical compared with the $O(3^n)$ baseline if D is large.

We refer to [1] for a more detailed discussion and examples.

Cross-References

- Enumeration of Paths, Cycles, and Spanning Trees
- Exact Algorithms and Time/Space Tradeoffs
- Exact Graph Coloring Using Inclusion-Exclusion
- Steiner Trees

Recommended Reading

1. Björklund A, Husfeldt T, Kaski P, Koivisto M (2007) Fourier meets Möbius: fast subset convolution. In: Johnson DS, Feige U (eds) STOC. ACM, pp 67–74. doi:10.1145/1250790.1250801, http://doi.acm.org/10.1145/1250790.1250801
2. Björklund A, Husfeldt T, Kaski P, Koivisto M (2008) Computing the Tutte polynomial in vertex-exponential time. In: FOCS. IEEE Computer Society, pp 677–686. doi:10.1109/FOCS.2008.40, http://doi.ieeecomputersociety.org/10.1109/FOCS.2008.40
3. Björklund A, Husfeldt T, Kaski P, Koivisto M (2009) Counting paths and packings in halves. In: [7], pp 578–586. doi:10.1007/978-3-642-04128-0_52, http://dx.doi.org/10.1007/978-3-642-04128-0_52

4. Björklund A, Husfeldt T, Kaski P, Koivisto M (2010) Trimmed Moebius inversion and graphs of bounded degree. Theory Comput Syst 47(3):637–654. doi:10.1007/s00224-009-9185-7, http://dx.doi.org/10.1007/s00224-009-9185-7
5. Björklund A, Husfeldt T, Kaski P, Koivisto M (2011) Covering and packing in linear space. Inf Process Lett 111(21–22):1033–1036. doi:10.1016/j.ipl.2011.08.002, http://dx.doi.org/10.1016/j.ipl.2011.08.002
6. Björklund A, Koivisto M, Husfeldt T, Nederlof J, Kaski P, Parviainen P (2012) Fast zeta transforms for lattices with few irreducibles. In: Rabani Y (ed) SODA. SIAM, pp 1436–1444. doi:10.1137/1.9781611973099.113, http://dx.doi.org/10.1137/1.9781611973099.113
7. Fiat A, Sanders P (eds) (2009) Algorithms – ESA 2009, 17th annual european symposium, Copenhagen, 7–9 Sept 2009. Proceedings, lecture notes in computer science, vol 5757. Springer
8. Fomin FV, Lokshtanov D, Raman V, Saurabh S, Rao BVR (2012) Faster algorithms for finding and counting subgraphs. J Comput Syst Sci 78(3):698–706. doi:10.1016/j.jcss.2011.10.001, http://dx.doi.org/10.1016/j.jcss.2011.10.001
9. Kennes R (1992) Computational aspects of the mobius transformation of graphs. IEEE Trans Syst Man Cybern 22(2):201–223. doi:10.1109/21.148425, http://dx.doi.org/10.1109/21.148425
10. Knuth DE (1998) The art of computer programming, vol 2. Seminumerical algorithms, 3rd edn. Addison–Wesley, Upper Saddle River
11. van Rooij JMM, Bodlaender HL, Rossmanith P (2009) Dynamic programming on tree decompositions using generalised fast subset convolution. In: [7], pp 566–577. doi:10.1007/978-3-642-04128-0_51, http://dx.doi.org/10.1007/978-3-642-04128-0_51
12. Yates F (1937) The design and analysis of factorial experiments. Imperial Bureau of Soil Science, Harpenden

Faster Deterministic Fully-Dynamic Graph Connectivity

Christian Wulff-Nilsen
Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

Keywords

Connectivity; Data structures; Dynamic graphs

Years and Authors of Summarized Original Work

2013; Wulff-Nilsen

Problem Definition

We consider the fully dynamic graph connectivity problem. Here we wish to maintain a data structure for a simple graph that changes over time. We assume a fixed set of n vertices and updates consist of adding and deleting single edges. The data structure should support queries for vertex pairs (u, v) of whether u and v are connected in the current graph.

Key Results

The first nontrivial data structure is due to Frederickson [2] who showed how to achieve deterministic worst-case update time $O(\sqrt{m})$ and query time $O(1)$, where m is the current number of edges of the graph. Using a sparsification technique, Eppstein et al. [1] obtained $O(\sqrt{n})$ update time. Much faster amortized bounds can be achieved. Henzinger and King [3] gave a data structure with $O(\log^3 n)$ randomized expected amortized update time and $O(\log n / \log\log n)$ query time. Update time was improved to $O(\log^2 n)$ by Henzinger and Thorup [4]. A deterministic data structure with $O(\log^2 n)$ amortized update time and $O(\log n / \log\log n)$ query time was given by Holm et al. [5]. Thorup [8] achieved a randomized expected amortized update time of $O(\log n(\log\log n)^3)$ and a query time of $O(\log n / \log\log\log n)$. The fastest known deterministic amortized data structure was given in [9]. Its update time is $O(\log^2 n / \log\log n)$ and query time is $O(\log n / \log\log n)$. Kapron et al. [6] gave a Monte Carlo algorithm with polylogarithmic worst-case operation time. A general cell-prove lower bound of $\Omega(\log n)$ was shown by Pătraşcu and Demaine [7].

In the following, we sketch the main ideas in the data structure presented in [9].

A Simple Data Structure

We start with a simple data structure similar to that of Thorup [8] (which is based on the data structure of Holm et al. [5]) that achieves $O(\log^2 n)$ update time and $O(\log n)$ query time. In the subsection below, we give the main ideas for improving these bounds by a factor of $\log \log n$.

In the following, denote by $G = (V, E)$ the current graph. The data structure maintains for each edge $e \in E$ a *level* $\ell(e)$ between 0 and $\ell_{\max} = \lfloor \log n \rfloor$. Initially, an edge has level 0 and its level can only increase over time. For the amortization, we can think of $\ell_{\max} - \ell(e)$ as the amount of credits associated with edge e, and every time $\ell(e)$ increases, e pays one credit (which may correspond to more than one unit of time). Let G_i denote the subgraph of G with vertex set V and containing the edges of level at least i and refer to each connected component of G_i as a *level i cluster*. The following invariant is maintained:

Invariant: For each i, any level i cluster contains at most $n/2^i$ vertices.

The clusters nest and thus have a forest representation. More specifically, the *cluster forest* of G is a forest $\mathcal{C}$ of rooted trees where a node u at depth i corresponds to a level i cluster $C(u)$ and the children of u correspond to level $i+1$ clusters contained in $C(u)$. Note that roots of $\mathcal{C}$ correspond to components of $G_0 = G$ and leaves correspond to vertices of G. Hence, if we can maintain $\mathcal{C}$, we can answer a connectivity query (u, v) in $O(\log n)$ time by traversing the leaf-to-root paths from u and v, respectively, and checking whether the roots are distinct.

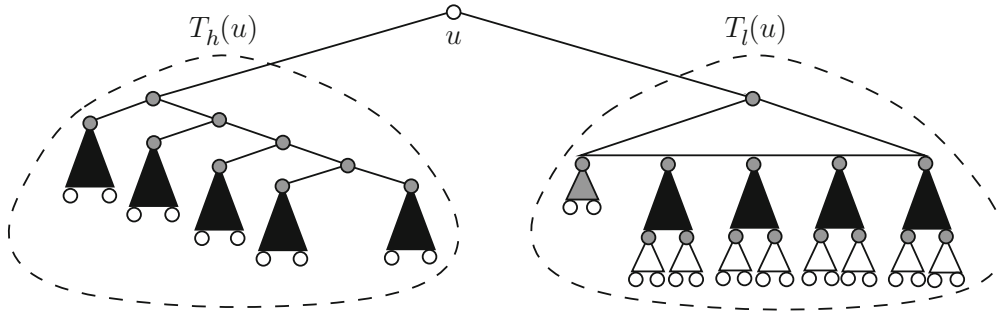
In the following, for each node $w \in \mathcal{C}$, denote by $n(w)$ the number of vertices of G contained in $C(w)$; equivalently, $n(w)$ is the number of leaves in the subtree of $\mathcal{C}$ rooted at w.

Edge insertions: When a new edge e is inserted into G, its level $\ell(e)$ is initialized to 0. Updating $\mathcal{C}$ amounts to merging the roots corresponding to the endpoints of e if these roots are distinct.

Edge deletions: Handling the deletion of an edge $e = (u, v)$ is more involved. Let $i = \ell(e)$, let $C(w_i)$ be the level i cluster containing e, and let $C(u_{i+1})$ and $C(v_{i+1})$ be the level $i+1$ clusters containing u and v, respectively. If $C(u_{i+1}) = C(v_{i+1})$, no changes occur in $\mathcal{C}$. Otherwise, consider the multigraph M obtained from $C(w_i)$ by contracting its level $i+1$ child clusters to single vertices. We search in parallel in M from $C(u_{i+1})$ and $C(v_{i+1})$, respectively, using a standard search procedure like BFS or DFS. Note that all edges visited have level i. If a search procedure visits a vertex a already visited by the other procedure, the removal of e does not disconnect the level i cluster containing e. In this case, we terminate both search procedures. Consider the two sets V_u and V_v of vertices of M visited by the procedures from $C(u_{i+1})$ resp. $C(v_{i+1})$ where we only include a in one of the sets. Then $V_u \cap V_v = \emptyset$, and since $n(w_i) \leq n/2^i$ by our invariant, either $\sum_{w \in V_u} n(w) \leq n/2^{i+1}$ or $\sum_{w \in V_v} n(w) \leq n/2^{i+1}$. Assume w.l.o.g. the former. Then we increase the level of all visited edges between vertices in V_u to $i+1$ without violating the invariant. These level increases pay for the search procedure from $C(u_{i+1})$, and since we ran the two procedures in parallel, they also pay for the search procedure from $C(v_{i+1})$.

If the two search procedures do not meet, we increase edge levels on one side as above but now $C(w_i)$ is split into two subclusters since we did not manage to reconnect it with level i edges. In this case, we recursively try to connect these two subclusters in the level $i-1$ cluster containing them. If we are in this case at level 0, it means that a connected component of $G_0 = G$ is split in two.

Performance: To show how to implement the above with $O(\log^2 n)$ update time, let us assume for now that $\mathcal{C}$ is a forest of *binary* trees. In order for a search procedure to visit a level i edge from a level $i+1$ cluster $C(a_{i+1})$ to a level $i+1$ cluster $C(b_{i+1})$, it identifies the start point a of this edge (a, b) in G by traversing the path in $\mathcal{C}$ from a_{i+1} down to leaf a. It then visits (a, b) and traverses the path in $\mathcal{C}$ from leaf b up to



Faster Deterministic Fully-Dynamic Graph Connectivity, Fig. 1 Hybrid local tree for u. Left subtree $T_h(u)$ is a simple local tree for the heavy children and the black subtrees are rank trees attached to a rank path ending in the root of $T_h(u)$. Right subtree $T_l(u)$ is a lazy local tree for the light children; see [8, 9] for details on the structure of this tree

b_{i+1}. To guide the downward searches for level i edges, we maintain for every node w of $\mathcal{C}$ a bitmap whose ith bit is 1 iff there is a level i-edge incident to a leaf in the subtree of $\mathcal{C}$ rooted at w. Since trees in $\mathcal{C}$ are binary, the start point a of (a, b) can be identified from a_{i+1} in $O(\log n)$ time using these bitmaps. Hence, each edge level increase costs $O(\log n)$. Since edge levels can only increase, an edge pays a total of $O(\log^2 n)$. Hence, we achieve an amortized update time of $O(\log^2 n)$.

Above, we assumed that trees in $\mathcal{C}$ are binary. To handle the general case, we modify $\mathcal{C}$ to a different forest $\mathcal{C}_L$ by adding a *simple local tree* $L(u)$ between each non-leaf node u and its children. Associate with each node $v \in \mathcal{C}$ a *rank* $\text{rank}(v) = \lfloor \log n(v) \rfloor$. To form $L(u)$, let C be the set of its children. As long as there are nodes in C with the same rank r, we give them a common parent with rank $r + 1$ and replace them by this parent in C. When this procedure terminates, we have at most $\log n$ *rank trees* whose roots have pairwise distinct ranks and we attach these roots to a *rank path* whose root is u; rank tree roots with bigger rank are attached closer to u than rank tree roots of smaller rank. The resulting tree $L(u)$ is binary; see the left subtree in Fig. 1 for an illustration. Hence, the trees in $\mathcal{C}_L$ are binary as well, and it is easy to see that they have height $O(\log n)$. The performance analysis above for $\mathcal{C}$ then carries through to $\mathcal{C}_L$, and we still have an amortized update time of $O(\log^2 n)$.

A Faster Data Structure

The above data structure has update time $O(\log^2 n)$ and query time $O(\log n)$. We now sketch how to speed up both bounds by a factor of $\log \log n$. We can get the speedup for query time by adding an upward shortcutting system in $\mathcal{C}_L$ where for each leaf-to-root path, we have shortcuts each skipping $\Theta(\log \log n)$ vertices. Maintaining this shortcutting system can be done efficiently. This system also gives a factor $\log \log n$ speedup for each of the upward searches performed by the search procedures described earlier. Speeding up downward searches can be done using a variant of a downward shortcutting system of Thorup [8].

These two shortcutting systems alone do not suffice to improve update time to $O(\log^2 n / \log \log n)$. The data structure needs to support merges and splits of clusters, and with the simple local trees defined above, this costs $O(\log n)$ time per merge/split, and each edge needs to pay a total of $O(\log^2 n)$ for this over all its level increases. Thorup [8] considered *lazy local trees* which can be maintained much more efficiently under cluster merges/splits. However, using these trees to form $\mathcal{C}_L$ may increase the height of trees in this forest to order $\log n \log \log n$ which will slow down our upward shortcutting system by a factor of $\log \log n$. To handle this, consider a hybrid of the simple local tree and Thorup's lazy local tree. For a non-leaf node u in $\mathcal{C}$, a child v is called *heavy* if $n(v) \geq n(u)/\log^{\epsilon} n$ where $\epsilon > 0$ is a constant that we

can pick arbitrarily small. A child that is not heavy is called *light*. Now, the *hybrid local tree* of u consists of a simple local tree $T_h(u)$ for the heavy children and a lazy local tree $T_l(u)$ for the light children; see Fig. 1 for an illustration. It can be shown that trees in $\mathcal{C}_L$ have height $O(\frac{1}{\epsilon}\log n)$ if we use hybrid local trees. Furthermore, $\mathcal{C}_L$ can be maintained efficiently under cluster merges and splits. The reason is that, although the hybrid local trees contain simple local trees which are expensive to maintain, these simple local trees are very small as each of them has at most $\log^{\epsilon} n$ leaves. Hence, maintaining them is not a bottleneck in the data structure.

Combining hybrid local trees with the two shortcutting systems suffice to obtain a factor $\log\log n$ speedup for updates and queries. This gives a deterministic data structure with $O(\log^2 n/\log\log n)$ update time and $O(\log n/\log\log n)$ query time.

Open Problems

Two main open problems for dynamic connectivity are:

- Is there a data structure with $O(\log n)$ operation time (which would be optimal by the lower bound in [7])?
- Is there a data structure with worst-case polylogarithmic operation time which is not Monte Carlo?

Recommended Reading

1. Eppstein D, Galil Z, Italiano GF, Nissenzweig A (1997) Sparsification – a technique for speeding up dynamic graph algorithms. J ACM 44(5):669–696. See also FOCS'92
2. Frederickson GN (1985) Data structures for on-line updating of minimum spanning trees, with applications. SIAM J Comput 14(4):781–798. See also STOC'83
3. Henzinger MR, King V (1995) Randomized dynamic graph algorithms with polylogarithmic time per operation. In: Proceedings of twenty-seventh annual ACM symposium on theory of computing (STOC), Las Vegas, pp 519–527
4. Henzinger MR, Thorup M (1997) Sampling to provide or to bound: with applications to fully dynamic graph algorithms. Random Struct Algorithms 11(4):369–379. See also ICALP'96
5. Holm J, de Lichtenberg K, Thorup M (2001) Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J ACM 48(4):723–760. See also STOC'98
6. Kapron BM, King V, Mountjoy B (2013) Dynamic graph connectivity in polylogarithmic worst case time. In: Proceedings of twenty-fourth annual ACM-SIAM symposium on discrete algorithms (SODA), New Orleans, pp 1131–1142
7. Pătraşcu M, Demaine E (2006) Logarithmic lower bounds in the cell-probe model. SIAM J Comput 35(4). Special issue 36th ACM symposium on theory of computing (STOC 2004)
8. Thorup M (2000) Near-optimal fully-dynamic graph connectivity. In: Proceedings of thirty-second annual ACM symposium on theory of computing (STOC), Portland, pp 343–350
9. Wulff-Nilsen C (2013) Faster deterministic fully-dynamic graph connectivity. In: Proceedings of twenty-fourth annual ACM-SIAM symposium on discrete algorithms (SODA), New Orleans, pp 1757–1769

Fault-Tolerant Connected Dominating Set

Donghyun Kim[1], Wei Wang[2], Weili Wu[3,4,5], and Alade O. Tokuta[1]

[1]Department of Mathematics and Physics, North Carolina Central University, Durham, NC, USA

[2]School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an, Shaanxi, China

[3]College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi Province, China

[4]Department of Computer Science, California State University, Los Angeles, CA, USA

[5]Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Keywords

Approximation algorithm; Connected dominating set; Fault tolerance; Graph algorithm; Vertex connectivity; Virtual backbone

Years and Authors of Summarized Original Work

2010; Kim, Wang, Li, Zhang, Wu
2013; Wang, Kim, An, Gao, Li, Zhang, Wu

Problem Definition

The problem of interest is to find a virtual backbone with a certain level of fault tolerance. Virtual backbone is a subset of nodes to be in charge of routing messages among the other nodes and is a very effective tool to improve the communication efficiency of various wireless networks such as mobile ad hoc networks and wireless sensor networks [3]. It is known that a virtual backbone with smaller cardinality works more efficiently. Without the fault-tolerance consideration, the problem of computing minimum cardinality virtual backbone can be formulated as a minimum connected dominating set problem [1], which is a well-known NP-hard problem [2]. To improve the fault tolerance of a connected dominating set C in homogenous wireless networks, C needs to exhibit two additional properties [4]:

- k-connectivity: C has to be k-vertex-connected so that the virtual backbone can survive even after $k-1$ backbone nodes fail.
- m-domination: each node v has to be adjacent to at least m nodes in C so that v can be still connected even after $m-1$ neighboring backbone nodes fail.

The actual value of the two integers, k and m, can be determined by a network operator based on the degree of fault tolerance desired. The majority of the results on this topic consider homogenous wireless networks which is a wireless network of uniform hardware functionality. In this case, the network can be abstracted using the unit disk graph model [6].

Mathematical Formulation

Given a unit disk graph $G = (V, E)$, a subset $C \subseteq V$ is a dominating set in G if for each node $v \in V \backslash C$, v has a neighboring node in C. C is an m-dominating set in G if for each node $v \in V \backslash C$, v has at least m neighboring nodes in C. C is a connected dominating set in G if C is a dominating set in G and if $G[C]$, the subgraph of G induced by C, is connected. C is a k-connected dominating set in G if $G[C]$ is a dominating set in G and $G[C]$ is k-vertex-connected. Finally, C is a k-connected m-dominating set if (a) $G[C]$ is k-vertex-connected and (b) C is an m-dominating set in G. Given $G = (V, E)$, the minimum k-connected m-dominating set problem is to find a minimum cardinality subset C of V satisfying those two requirements.

Key Results

The initial discussion about the need of fault tolerance in virtual backbones has been made by Dai and Wu [4]. Since the minimum k-connected m-dominating set problem in NP-hard, many efforts are made to design a constant factor approximation algorithm for the problem. In [7], Wang et al. proposed a constant factor approximation algorithm for the problem with $k = 2$ and $m = 1$. In [8], Shang et al. introduced a constant factor approximation algorithm for arbitrary integer m and $k = 1, 2$. Later, lots of efforts are made to introduce a constant factor approximation algorithm for arbitrary k and m pairs [9–13]. However, all of them do not work or lose the claimed constant approximation bound in some instances when $k \geq 3$ [14, 15].

In [16], the authors introduce an $O(1)$ approximation algorithm, Fault-Tolerant Connected Dominating Sets Computation Algorithm (FT-CDS-CA), which computes $(3, m)$-CDSs in UDGs. The core part of the algorithm is for computing a $(3, 3)$-CDS, and then it can be easily adapted to compute $(3, m)$-CDS for any $m \geq 1$. The following sections will introduce some key ideas and results of this work.

Constant Approximation for 3-Connected m-Dominating Set

Core Idea

The algorithm starts from a 2-connected 3-dominating set $Y_0 := C_{2,3}$, which can be done

by the algorithm in [8]. Then, it augments the connectivity of the subset by adding a set of nodes $C_0 \subset V \setminus Y_0$ into Y_0 while guaranteeing the number of the newly added nodes C_0 is within a constant factor of $|Y_0|$. In order to do so, the entry introduces the concept of a good node and a bad node. A node u in a 2-connected graph G_2 is called *a good node* if $G_2 \setminus \{u\}$ is still 2-connected, that is, it cannot constitute a separator with any other node in G_2; otherwise it is *a bad node*. An important observation is that a 2-connected graph without bad nodes is 3-connected. Then the entry shows that one can always convert a bad node into a good node by adding a constant number of nodes into Y_0 while not introducing new bad nodes, and they gave an efficient way to achieve this goal. By repeatedly changing bad nodes in Y_0 into good nodes until no bad node is left, Y_0 eventually becomes 3-connected whose size is guaranteed to be within a constant factor of the optimal solution.

Brief Description

A. Removing Separators If a 2-connected graph G_2 is not 3-connected, then there exists a pair of nodes u and v, called separator of G_2, such that $G_2 \setminus \{u, v\}$ splits into several parts. It can be shown, due to the properties of UDG, that by adding the internal nodes of at most a constant number of H_3-paths (by an H_3-path we mean a path with length at most three connecting two nodes of a subgraph H of G_2, the internal nodes of which do not belong to H) into Y_0, $\{u, v\}$ is no longer a separator of Y_0, and the nodes newly added are good nodes because $Y_0 = C_{2,3}$ is a 3-dominating set.

B. Decomposition of a Connected Graph into a Leaf-Block Tree In graph theory, a block of a graph is a maximal 2-connected subgraph [5]. Given a 2-connected subgraph Y_0 (initially, this is a $C_{2,3}$) and the set X of bad points in Y_0, we select $v \in X$ as a root and compute a leaf-block tree T_0 of $Y_0 \setminus \{v\}$ [5]. Then, T_0 constitutes of a set of blocks $\{B_1, B_2, \ldots, B_s\}$ and a set of cut vertices $\{c_1, \ldots, c_t\}$. An important fact is that v can constitute a separator only with another node in $\{c_1, \ldots, c_t\}$.

C. Good Blocks vs Bad Blocks In the process of decomposing a block B with root v into a leaf-block tree, it is important to identify those blocks B_i containing *internal bad nodes*, that is, those bad nodes in B_i that cannot be connected with nodes outside B_i directly without going through v (otherwise, it is called *external bad nodes*). We call such a block B_i with (resp. without) an internal bad node *a bad block* (resp. *a good block*). A key fact is that an internal bad node in B_i can only constitute a separator of Y_0 with another node inside B_i, while this may not be true for external bad nodes.

D. Multilevel Decomposition The purpose of the multilevel decomposition is to find a block B with root v such that $B \setminus \{v\}$ contains only good blocks. We assume that $X \neq \emptyset$, since otherwise Y_0 is already 3-connected. After setting $B \leftarrow Y_0$, FT-CDS-CA first picks one $v \in X$ and starts the initial decomposition process (say level-0 decomposition). Then, $B \setminus \{v\}$ is decomposed into a (level-0) leaf-block graph T_0, which is a tree whose vertices consist of a set of blocks $\{B_1, \ldots, B_s\}$ and a set of cut vertices $\{c_1, c_2, \ldots, c_t\}$ ($s \geq 2$ and $t \geq 1$). Now, FT-CDS-CA examines each block in T_0 to see if there is a block B_i having an internal bad node in it. If all blocks are good blocks, then we are done in this step. Otherwise, there must exist some B_i having an internal node $w \in B_i$ which constitutes a separator $\{w, u\}$ of Y_0 with another node $u \in B_i \subset Y_0$. Now, set $v \leftarrow w$ and $B \leftarrow B_i$, start next level (level-1) decomposition. By repeating such process, we can keep making our problem smaller and eventually can find a block B with root v such that $B \setminus \{v\}$ contains only good blocks.

E. Merging Blocks (Reconstructing the Leaf-Block Tree with a New Root) After the multilevel decomposition process, we obtain a series of blocks: $Y_l \subset Y_{l-1} \subset \cdots Y_1 \subset Y_0$, where $Y_l = B$ is the final block with a root v such that there is no bad block in the leaf-block tree of $Y_l \setminus \{v\}$. In the induced subgraph $G[Y_l]$, v constitutes a separator with any of $c_1, c_2, \ldots, c_t$, but in Y_0 this is not necessarily true, since there exist some blocks B_i having external nodes that are adjacent

to some nodes in $Y_0 \setminus Y_l$ (otherwise, Y_l and $Y_0 \setminus Y_l$ cannot be connected with each other). So these blocks that can be connected directly with $Y_0 \setminus Y_l$ without going through v should be merged together with $Y_0 \setminus Y_l$ into a larger block. After merging all possible blocks into one bigger block, we obtain a modified leaf-block tree T_l' in which one bigger block VB (we call it a *virtual block*) is added representing all the merged blocks and $Y_0 \setminus Y_l$, and all the cut vertices c_i which do not constitute a separator with v have to be removed. Moreover, we mark every remaining cut vertex of VB as a *virtual cut vertex*. In essence, the above merging process can be considered as a process to generate a leaf-block tree directly from $Y_0 \setminus \{v\}$ with all blocks being good except possibly for the virtual block.

F. One Bad-Node Elimination At this point, we have a leaf-block tree T_l' with $V(T_l') = \{B_1, B_2, \ldots, B_s, VB\} \cup \{c_1, \ldots, c_t\}$, which is obtained through the decomposition of $Y_0 \setminus \{v\}$ (or, equivalently, through the merging process), where v is the internal bad node chosen as root in $B = Y_l$. Note in T_l', every B_i is a good block except possibly for the virtual block VB. The key point here is that we must have $s \geq 1$; otherwise v would be a good node. In this step, a simple process is employed to make either v or one of the cut vertices in $\{v_1, v_2, \ldots, v_t\} \setminus C$ (C is the set of cut vertices in the virtual block VB) to be a good node. Consider two cases: (i) if the leaf-block tree T_l' has only virtual cut vertices (i.e., T_l is a star centered at VB), then the bad node v becomes a good node by removing the separators consisting of v and the virtual cut vertices, and (ii) if the leaf-block tree T_l' has a cut vertex which is not a virtual cut vertex, then we can find a path $P = (\tilde{B}_0, \tilde{c}_1, \ldots, R)$ in T_l' with one endpoint $\tilde{B}_0$ being a leaf in the tree T_l' and the other endpoint R being a block (cut vertex) with degree larger than two or the virtual block VB (a virtual cut vertex c), if the former does not exist. In this case, two consecutive blocks $\tilde{B}_i$ and $\tilde{B}_{i+1}$ can be found which share a common cut vertex $\tilde{c}_i$. Then it can be shown that at most five H_3-paths are needed such that $\tilde{c}_i$ cannot constitute a pair of separator of Y_0 with any of the remaining cut vertex or v. Meanwhile, it is still possible that $\tilde{c}_i$ may constitute separators of Y_0 with the external nodes $\tilde{B}_i$ and $\tilde{B}_{i+1}$ (clearly $\tilde{c}_i$ cannot constitute separators of Y_0 with the internal nodes $\tilde{B}_i$ and $\tilde{B}_{i+1}$). It can be proved that the total number of external nodes in $\tilde{B}_i$ and $\tilde{B}_{i+1}$ that may constitute separators of Y_0 with $\tilde{c}_i$ is at most five. In both cases, the number of H_3-paths added to change one bad node (v or $\tilde{c}_i$) into a good node is at most a constant.

Open Problems

While Wang et al. [16] manage to introduce a constant factor approximation algorithm for the minimum k-connected m-dominating set problem in unit disk graph with $k = 3$ and arbitrary integer $m \geq 1$, it is still open to design an approximation algorithm for the case with $k \geq 4$.

Experimental Results

Wang et al.'s work [16] presents some simulation results. The results show that when a 2-connected 3-dominating set computed by Shang et al.'s approach [8] is augmented to a 3-connected 3-dominating set using their algorithm, the size of the connected dominating set will modestly increase roughly less than 25 %. Their algorithm is also compared with an optimal solution using an exhaustive computation within small-scale random unit disk graphs. The result shows the performance gap between exact algorithm and their algorithm is no greater than 39.27 %.

Cross-References

- Connected Dominating Set
- Efficient Dominating and Edge Dominating Sets for Graphs and Hypergraphs
- Exact Algorithms for Dominating Set
- Strongly Connected Dominating Set

Recommended Reading

1. Guha S, Khuller S (1998) Approximation algorithms for connected dominating sets. Algorithmica 20:374–387
2. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco
3. Sinha P, Sivakumar R, Bharghavan V (2001) Enhancing ad hoc routing with dynamic virtual infrastructures. In: Proceedings of the 20th annual joint conference of the IEEE computer and communications societies, Anchorage vol 3, pp 1763–1772
4. Dai F, Wu J (2005), On constructing k-connected k-dominating set in wireless network. In: Proceedings of the 19th IEEE international parallel and distributed processing symposium, Denver
5. Diestel R (2005) Graph theory, 3rd edn. Springer, Heidelberg
6. Clark BN, Colbourn CJ, Johnson DS (1990) Unit disk graphs. Discret Math 86:165–177
7. Wang F, Thai MT, Du DZ (2009) 2-connected virtual backbone in wireless network. IEEE Trans Wirel Commun 8(3):1230–1237
8. Shang W, Yao F, Wan P, Hu X (2007) On minimum m-connected k-dominating set problem in unit disc graphs. J Comb Optim 16(2):99–106
9. Li Y, Wu Y, Ai C, Beyah R (2012) On the construction of k-connected m-dominating sets in wireless networks. J Comb Optim 23(1):118–139
10. Thai MT, Zhang N, Tiwari R, Xu X (2007) On approximation algorithms of k-connected m-dominating sets in disk graphs. Theor Comput Sci 358:49–59
11. Wu Y, Wang F, Thai MT, Li Y (2007) Constructing k-connected m-dominating sets in wireless sensor networks. In: Proceedings of the 2007 military communications conference, Orlando
12. Wu Y, Li Y (2008) Construction algorithms for k-connected m-dominating sets in wireless sensor networks. In: Proceedings of 9th ACM international symposium on mobile ad hoc networking and computing, Hong Kong
13. Zhang N, Shin I, Zou F, Wu W, Thai MT (2008) Trade-off scheme for fault tolerant connected dominating sets on size and diameter. In: Proceedings of the 1st ACM international workshop on foundations of wireless ad hoc and sensor networking and computing (FOWANC '08), Hong Kong
14. Kim D, Gao X, Zou F, Du DZ (2011) Construction of fault-tolerant virtual backbones in wireless networks. Handbook on security and networks. World Scientific, Hackensack, pp 488–509
15. Kim D, Wang W, Li X, Zhang Z, Wu W (2010) A new constant factor approximation for computing 3-connected m-dominating sets in homogeneous wireless networks. In: Proceedings of the 29th IEEE conference on computer communications, San Diego
16. Wang W, Kim D, An MK, Gao W, Li X, Zhang Z, Wu W (2013) On construction of quality fault-tolerant virtual backbone in wireless networks. IEEE/ACM Trans Netw 21(5):1499–1510

Fault-Tolerant Quantum Computation

Ben W. Reichardt
Electrical Engineering Department, University of Southern California (USC), Los Angeles, CA, USA

Keywords

Quantum noise threshold

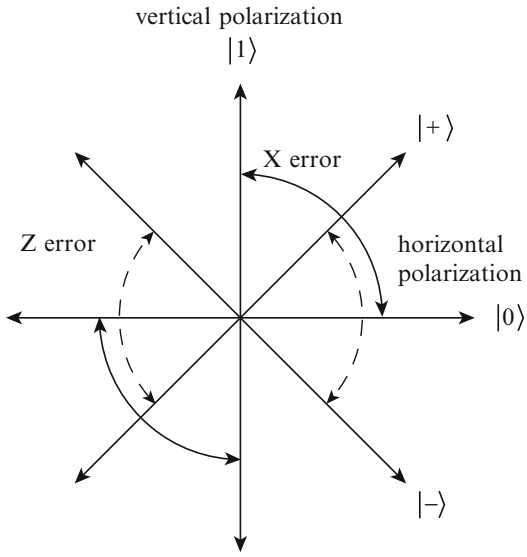
Years and Authors of Summarized Original Work

1996; Shor, Aharonov, Ben-Or, Kitaev

Problem Definition

Fault tolerance is the study of reliable computation using unreliable components. With a given noise model, can one still reliably compute? For example, one can run many copies of a classical calculation in parallel, periodically using majority gates to catch and correct faults. Von Neumann showed in 1956 that if each gate fails independently with probability p, flipping its output bit $0 \leftrightarrow 1$, then such a fault tolerance scheme still allows for arbitrarily reliable computation provided that p is below some constant threshold (whose value depends on the model details) [10].

In a quantum computer, the basic gates are much more vulnerable to noise than classical transistors – after all, depending on the implementation, they are manipulating single electron spins, photon polarizations, and similarly fragile subatomic particles. It might not be possible to engineer systems with noise rates less than



Fault-Tolerant Quantum Computation, Fig. 1 Bit-flip X errors flip 0 and 1. In a qubit, $|0\rangle$ and $|1\rangle$ might be represented by *horizontal* and *vertical polarization* of a photon, respectively. Phase-flip Z errors flip the $\pm 45°$ polarized states $|+\rangle$ and $|-\rangle$

10^{-2}, or perhaps 10^{-3}, per gate. Additionally, the phenomenon of entanglement makes quantum systems *inherently* fragile. For example, in Schrödinger's cat state – an equal superposition between a living cat and a dead cat, often idealized as $1/\sqrt{2}(|0^n\rangle + |1^n\rangle)$ – an interaction with just one quantum bit ("qubit") can collapse, or decohere, the entire system. Fault tolerance techniques will therefore be essential for achieving the considerable potential of quantum computers. Practical fault tolerance techniques will need to control high noise rates and do so with low overhead, since qubits are expensive.

Quantum systems are continuous, not discrete, so there are many possible noise models. However, the essential features of quantum noise for fault tolerance results can be captured by a simple discrete model similar to the one Von Neumann used. The main difference is that, in addition to bit-flip X errors which swap 0 and 1, there can also be phase-flip Z errors which swap $|+\rangle \equiv 1/\sqrt{2}(|0\rangle + |1\rangle)$ and $|-\rangle \equiv 1/\sqrt{2}(|0\rangle - |1\rangle)$ (Fig. 1). A noisy gate is modeled as a perfect gate followed by independent introduction of X, Z, or Y (which is both X and Z) errors with respective probabilities p_X, p_Z, p_Y. One popular model is independent depolarizing noise ($p_X = p_Z = p_Y \equiv p/3$); a depolarized qubit is completely randomized.

Faulty measurements and preparations of single-qubit states must additionally be modeled, and there can be memory noise on resting qubits. It is often assumed that measurement results can be fed into a classical computer that works perfectly and dynamically adjusts the quantum gates, although such control is not necessary. Another common, though unnecessary, assumption is that any pair of qubits in the computer can interact; this is called a *nonlocal* gate. In many proposed quantum computer implementations, however, qubit mobility is limited so gates can be applied only locally, between physically nearby qubits.

Key Results

The key result in fault tolerance is the existence of a noise *threshold*, for certain noise and computational models. The noise threshold is a positive, constant noise rate (or set of model parameters) such that with noise below this rate, reliable computation is possible. That is, given an inputless quantum circuit $\mathcal{C}$ of perfect gates, there exists a "simulating" circuit $FT\mathcal{C}$ of faulty gates such that with probability at least 2/3, say, the measured output of $\mathcal{C}$ agrees with that of $FT\mathcal{C}$. Moreover, $FT\mathcal{C}$ should be only polynomially larger than $\mathcal{C}$.

A quantum circuit with N gates can a priori tolerate only $O(1/N)$ error per gate, since a single failure might randomize the entire output. In 1996, Shor showed how to tolerate $O(1/\text{poly}(\log N))$ error per gate by encoding each qubit into a poly(log N)-sized quantum error-correcting code and then implementing each gate of the desired circuit directly on the encoded qubits, alternating computation and error correction steps (similar to Von Neumann's scheme) [8]. Shor's result has two main technical pieces:

1. The discovery of quantum error-correcting codes (QECCs) was a major result. Remarkably, even though quantum errors can be continuous, codes that correct discrete errors suffice. (Measuring the syndrome of a code block projects into a discrete error event.) The first quantum code, discovered by Shor, was a nine-qubit code consisting of the concatenation of the three-qubit repetition code $|0\rangle \rightarrow |000\rangle$, $|1\rangle \rightarrow |111\rangle$ to protect against bit-flip errors, with its dual $|+\rangle \mapsto |++\rangle$, $|-\rangle \mapsto |--\rangle$ to protect against phase-flip errors. Since then, many other QECCs have been discovered. Codes like the nine-qubit code that can correct bit- and phase-flip errors separately are known as Calderbank-Shor-Steane (CSS) codes and have quantum code words which are simultaneously superpositions over code words of classical codes in both the $|0/1\rangle$ and $|+/-\rangle$ bases.
2. QECCs allow for quantum memory or for communicating over a noisy channel. For computation, however, it must be possible to compute on encoded states without first decoding. An operation is said to be *fault tolerant* if it cannot cause correlated errors within a code block. With the n-bit majority code, all classical gates can be applied *transversely* – an encoded gate can be implemented by applying the unencoded gate to bit i of each code block, $1 \leq i \leq n$. This is fault tolerant because a single failure affects at most 1 bit in each block, and thus, failures can't spread too quickly. For CSS quantum codes, the controlled-NOT gate CNOT, $|a, b\rangle \rightarrow |a, a \oplus b\rangle$, can similarly be applied transversely. However, the CNOT gate by itself is not universal, so Shor also gave a fault-tolerant implementation of the Toffoli gate $|a, b, c\rangle \rightarrow |a, b, c \oplus (a \wedge b)\rangle$. Procedures are additionally needed for error correction using faulty gates and for the initial preparation step. The encoding of $|0\rangle$ will be a highly entangled state and difficult to prepare (unlike 0^n for the classical majority code).

However, Shor did not prove the existence of a *constant* tolerable noise rate, a noise threshold. Several groups – Aharonov/Ben-Or, Kitaev, and Knill/Laflamme/Zurek – each had the idea of using smaller codes and *concatenating* the procedure repeatedly on top of itself. Intuitively, with a distance-three code (i.e., code that corrects any one error), one expects the "effective" logical error rate of an encoded gate to be at most cp^2 for some constant c, because one error can be corrected but two errors cannot. The effective error rate for a twice-encoded gate should then be at most $c(cp^2)^2$; and since the effective error rate is dropping doubly exponentially fast in the number of levels of concatenation, the overhead in achieving a $1/N$ error rate is only poly(log N). The threshold for improvement, $cp^2 < p$, is $p < 1/c$. However, this rough argument is not rigorous, because the effective error rate is ill defined, and logical errors need not fit the same model as physical errors (e.g., they will not be independent).

Aharonov and Ben-Or and Kitaev gave independent rigorous proofs of the existence of a positive constant noise threshold, in 1997 [1, 5].

Broadly, there has since been progress on two fronts of the fault tolerance problem:

1. First, work has proceeded on extending the set of noise and computation models in which a fault tolerance threshold is known to exist. For example, correlated or even adversarial noise, leakage errors (where a qubit leaves the $|0\rangle$, $|1\rangle$ subspace), and non-Markovian noise (in which the environment has a memory) have all been shown to be tolerable in theory, even with only local gates.
2. Threshold existence proofs establish that building a working quantum computer is possible *in principle*. Physicists need only engineer quantum systems with a low enough constant noise rate. But realizing the potential of a quantum computer will require *practical* fault tolerance schemes. Schemes will have to tolerate a high noise rate (not just some constant) and do so with low overhead (not just polylogarithmic).

However, rough estimates of the noise rate tolerated by the original existence proofs are not promising – below 10^{-6} noise per gate. If the true threshold is only 10^{-6}, then building a quantum computer will be next to impossible. Therefore, second, there has been substantial work on optimizing fault tolerance schemes primarily in order to improve the tolerable noise rate. These optimizations are typically evaluated with simulations and heuristic analytical models. Recently, though, Aliferis, Gottesman, and Preskill have developed a method to prove reasonably good threshold lower bounds, up to 2×10^{-4}, based on counting "malignant" sets of error locations [3].

In a breakthrough, Knill has constructed a novel fault tolerance scheme based on very efficient distance-*two* codes [6]. His codes cannot correct any errors, and the scheme uses extensive postselection on no detected errors – i.e., on detecting an error, the enclosing subroutine is restarted. He has estimated a threshold above 3 % per gate, an order of magnitude higher than previous estimates. Reichardt has proved a threshold lower bound of 10^{-3} for a similar scheme [7], somewhat supporting Knill's high estimate. However, reliance on postselection leads to an enormous overhead at high error rates, greatly limiting practicality. (A classical fault tolerance scheme based on error detection could not be efficient, but quantum teleportation allows Knill's scheme to be at least theoretically efficient.) There seems to be tradeoff between the tolerable noise rate and the overhead required to achieve it.

There are several complementary approaches to quantum fault tolerance. For maximum efficiency, it is wise to exploit any known noise structure before switching to general fault tolerance procedures. Specialized techniques include careful quantum engineering, techniques from nuclear magnetic resonance (NMR) such as dynamical decoupling and composite pulse sequences, and decoherence-free subspaces. For very small quantum computers, such techniques may give sufficient noise protection.

It is possible that an inherently reliable quantum-computing device will be engineered or discovered, like the transistor for classical computing, and this is the goal of *topological* quantum computing [4].

Applications

As quantum systems are noisy and entanglement fragile, fault tolerance techniques will probably be essential in implementing any quantum algorithms – including efficient factoring and quantum simulation.

The quantum error-correcting codes originally developed for fault tolerance have many other applications, including quantum key distribution.

Open Problems

Dealing with noise may turn out to be the most daunting task in building a quantum computer. Currently, physicists' low-end estimates of achievable noise rates are only slightly below theorists' high-end (mostly simulation based) estimates of tolerable noise rates, at reasonable levels of overhead. However, these estimates are made with different noise models – most simulations are based on the simple independent depolarizing noise model, and threshold lower bounds for more general noise are much lower. Also, both communities may be being too optimistic. Unanticipated noise sources may well appear as experiments progress. The probabilistic noise models used by theorists in simulations may not match reality closely enough, or the overhead/threshold tradeoff may be impractical. It is not clear if fault-tolerant quantum computing will work in practice, unless inefficiencies are wrung out of the system. Developing more efficient fault tolerance techniques is a major open problem. Quantum system engineering, with more realistic simulations, will be required to understand better various tradeoffs and strategies for working with gate locality restrictions.

The gaps between threshold upper bounds, threshold estimates, and rigorously proven threshold lower bounds are closing, at least for simple noise models. Our understanding of what to expect with more realistic noise models

is less developed, though. One current line of research is in extending threshold proofs to more realistic noise models – e.g., [2]. A major open question here is whether a noise threshold can be shown to even *exist* where the bath Hamiltonian is unbounded – e.g., where system qubits are coupled to a non-Markovian, harmonic oscillator bath. Even when a threshold is known to exist, rigorous threshold lower *bounds* in more general noise models may still be far too conservative (according to arguments, mostly intuitive, known as "twirling") and, since simulations of general noise models are impractical, new ideas are needed for more efficient analyses.

Theoretically, it is of interest what is the best asymptotic overhead in the simulating circuit FTC versus C? Overhead can be measured in terms of size N and depth/time T. With concatenated coding, the size and depth of FTC are $O(N\text{polylog } N)$ and $O(T\text{polylog } N)$, respectively. For classical circuit C, however, the depth can be only $O(T)$. It is not known if the quantum depth overhead can be improved.

Experimental Results

Fault tolerance schemes have been simulated for large quantum systems, in order to obtain threshold estimates. For example, extensive simulations including geometric locality constraints have been run by Thaker et al. [9].

Error correction using very small codes has been experimentally verified in the lab.

URL to Code

Andrew Cross has written and distributes code for giving Monte Carlo estimates of and rigorous lower bounds on fault tolerance thresholds: http://web.mit.edu/awcross/www/qasm-tools/.
Emanuel Knill has released *Mathematica* code for estimating fault tolerance thresholds for certain postselection-based schemes: http://arxiv.org/e-print/quant-ph/0404104.

Cross-References

▸ Quantum Error Correction

Recommended Readings

1. Aharonov D, Ben-Or M (1997) Fault-tolerant quantum computation with constant error rate. In: Proceedings 29th ACM symposium on theory of computing (STOC), pp 176–188. quant-ph/9906129
2. Aharonov D, Kitaev AY, Preskill J (2006) Fault-tolerant quantum computation with long-range correlated noise. Phys Rev Lett 96:050504. quant-ph/0510231
3. Aliferis P, Gottesman D, Preskill J (2006) Quantum accuracy threshold for concatenated distance-3 codes. Quantum Inf Comput 6:97–165. quant-ph/0504218
4. Freedman MH, Kitaev AY, Larsen MJ, Wang Z (2002) Topological quantum computation. Bull AMS 40(1):31–38
5. Kitaev AY (1997) Quantum computations: algorithms and error correction. Russ Math Surv 52:1191–1249
6. Knill E (2005) Quantum computing with realistically noisy devices. Nature 434:39–44
7. Reichardt BW (2006) Error-detection-based quantum fault tolerance against discrete Pauli noise. Ph.D. thesis, University of California, Berkeley. quant-ph/0612004
8. Shor PW (1996) Fault-tolerant quantum computation. In: Proceedings of the 37th symposium on foundations of computer science (FOCS). quant-ph/9605011
9. Thaker DD, Metodi TS, Cross AW, Chuang IL, Chong FT (2006) Quantum memory hierarchies: efficient designs to match available parallelism in quantum computing. In: Proceedings of the 33rd international symposium on computer architecture (ISCA), pp 378–390. quant-ph/0604070
10. von Neumann J (1956) Probabilistic logic and the synthesis of reliable organisms from unreliable components. In: Shannon CE, McCarthy J (eds) Automata studies. Princeton University Press, Princeton, pp 43–98

Finding Topological Subgraphs

Paul Wollan
Department of Computer Science, University of Rome La Sapienza, Rome, Italy

Keywords

Disjoint paths; Fixed-parameter tractability; Topological minor; Topological subgraph

Years and Authors of Summarized Original Work

2011; Grohe, Kawarabayashi, Marx, Wollan

Problem Definition

To *subdivide* an edge e in a graph G with endpoints u and v, delete the edge from the graph and add a path of length two connecting the vertices u and v. A graph G is a *subdivision* of graph H if G can be obtained from H by repeatedly subdividing edges. A graph H is a *topological subgraph* (or *topological minor*) of graph G if a subdivision of H is a subgraph of G. Equivalently, H is a topological subgraph of G if H can be obtained from G by deleting edges, deleting vertices, and suppressing vertices of degree 2 (to *suppress* a vertex of degree 2, delete the vertex and add an edge connecting its two neighbors). The notion of topological subgraphs appears in the classical result of Kuratowski in 1935 stating that a graph is planar if and only if it does not have a topological subgraph isomorphic to K_5 or $K_{3,3}$. This entry considers the problem of determining, given a graph G and H, whether G contains H as a topological minor.

Topological Subgraph Testing

Input: Graphs G and H
Output: Determine if H is a topological subgraph of G

Observe that a graph G on n vertices contains the cycle of length n as a topological subgraph if and only if G contains a Hamiltonian cycle. Thus, it is NP-complete to decide if H is a topological subgraph of a graph G with no further restrictions on G or H.

Previous Work

The algorithmic problem of testing for topological subgraphs was already studied in the 1970s by Lapaugh and Rivest [12] (also see [7]). Fortune, Hopcroft, and Wyllie [6] showed that the analogous problem in directed graphs is NP-complete even when H is a fixed small graph. Robertson and Seymour, as a consequence of their seminal work on graphs minors, showed that for a fixed graph H, there exists a polynomial time algorithm to check whether H is a topological subgraph of a graph G given in input. However, the running time of the Robertson-Seymour algorithm is $|V(G)|^{O(|V(H)|)}$. Following this, Downey and Fellows [4] (see also [5]) conjectured that the problem of topological subgraph testing is fixed parameter tractable: they conjectured that there exists a function f and a constant c such that there exists an algorithm for testing whether a graph H is a topological subgraph of G which runs in time $f(|V(H)|) \cdot |V(G)|^c$.

The problem of topological subgraph testing is closely related to that of minor testing and the k-disjoint paths problem. A graph H is a *minor* of G if H can be obtained from a subgraph of G by contracting edges. The k-disjoint paths problem instead takes as input k pairs of vertices $(s_1, t_1), \ldots, (s_k, t_k)$ of vertices in a graph G and asks if there exist pairwise internally vertex disjoint paths $P_1, \ldots, P_k$ such that the endpoints of P_i are s_i and t_i for all $1 \leq i \leq k$. Robertson and Seymour [13] considered a model of *labeled minor containment* that unites these two problems and showed that there is an $O(|V(G)|^3)$ time algorithm for both H-minor testing for a fixed graph H and the k-disjoint paths problem for a fixed value k.

For every H, there exists a finite list $H_1, \ldots, H_t$ of graphs such that a graph G contains H as a minor if and only if G contains H_i as a topological minor for some index i; this follows from the definition of minor and topological minor. Thus, the problem of minor testing reduces to the harder problem of topological minor testing. It is not difficult to reduce the problem of topological subgraph containment for a fixed graph H to the k-disjoint paths problem. For each vertex v of H, guess a vertex v' of G, and then for each edge uv of H, and seek to find a path connecting u' and v' in G such that these $|E(H)|$ paths are pairwise internally vertex disjoint. This approach yields the $|V(G)|^{O(|V(H)|)}$ time algorithm for topological subgraph testing mentioned above.

Key Results

The following theorem of Grohe, Kawarabayashi, Marx, and Wollan [8] shows that topological

subgraph testing is fixed parameter tractable, confirming the conjecture of Downey and Fellows.

Theorem 1 *For every fixed, undirected graph H, there is an $O(|V(G)|^3)$ time algorithm that decides if H is a topological subgraph of G.*

Outline of the Proof

The algorithm given by Theorem 1 builds on the techniques first developed by Robertson and Seymour in their algorithm for minor testing and the k-disjoint paths problem. Fix a graph H and let G be a graph given in input. The algorithm separately considers each of the following three cases:

1. The tree-width of G is bounded (by an appropriate function on $|V(H)|$);
2. G has large tree-width, but the size of the largest clique minor is bounded (again by an appropriate function on $|V(H)|$);
3. G has a large clique minor.

Note that in the third case, the existence of a large clique minor necessarily forces the graph G to have large tree-width. We do not use any technical aspects of the parameter tree-width here and direct interested readers to [1, 2] for further discussion of this topic.

The Robertson-Seymour algorithm for minor testing offers a roadmap for the proof of Theorem 1; the discussion of the proof of Theorem 1 highlights where the proof builds on the tools of Robertson and Seymour and where new techniques are required. As in Robertson-Seymour's algorithm for minor testing, the algorithm considers a rooted version of the problem.

G has Bounded Tree-Width

Numerous problems can be efficiently solved when the input graph is restricted to have bounded tree-width (see [1, 3] for examples). For example, the k-disjoint paths problem can be solved in linear time in graphs of bounded tree-width [15]. Standard dynamic programming techniques can be used to solve the more general rooted version of the topological subgraph problem which the algorithm considers.

G has Large Tree-Width, but no Large Clique Minor

Robertson and Seymour showed that graphs of large tree-width which do not contain a fixed clique minor must contain a large, almost planar subgraph [11, 13]; this result is sometimes known as the flat-wall theorem. The proof of correctness for their disjoint paths algorithm hinges upon this theorem by showing that a vertex in the planar subgraph can be deleted without affecting the feasibility of a given disjoint paths problem. The proof of Theorem 1 builds on this approach. Given graphs G and H, say a vertex $v \in V(G)$ is *irrelevant* for the problem of topological subgraph testing if G contains H as a topological minor if and only if $G - v$ contains H as a topological minor. If the algorithm can efficiently find an irrelevant vertex v, then it can proceed by recursing on the graph $G - v$. In order to apply a similar irrelevant vertex argument to that developed by Robertson and Seymour for the disjoint paths problem, the proof of Theorem 1 shows that a large flat wall contains an irrelevant vertex for a given topological subgraph testing problem by first generalizing several technical results on rerouting systems of paths in graphs [10, 13] as well as deriving a stronger version of the flat-wall theorem.

G has a Large Clique Minor

In the Robertson and Seymour algorithm for minor testing, once the graph can be assumed to have a large clique minor, the algorithm trivially terminates. When considering the k-disjoint paths problem, again it is a relatively straightforward matter to find an irrelevant vertex for a given disjoint paths problem assuming the existence of a large clique minor. Instead, if we are considering the problem of testing topological subgraph containment, the presence of a large clique minor does not yield an easy recursion. Consider the case where we are testing for the existence of a topological subgraph of a 4-regular graph H in a graph G which contains a subcubic subgraph G' such that G' has a large clique minor. Whether or not G' will prove useful in finding a topological subgraph of H in G will depend entirely on whether

or not it is possible to link many vertices of degree four (in G) to the clique minor in G' and not on the size itself of the clique minor in G'. Similar issues arise in [9] when developing structure theorems for excluded topological subgraphs.

The proof of Theorem 1 proceeds by considering separately the case when the large-degree vertices can be separated from the clique minor by a bounded sized separator or not. If they cannot, one can find the necessary rooted topological minors. Alternatively, if they can, the algorithm recursively calculates the rooted topological minors in subgraphs of G and replaces a portion of the graph with a bounded size gadget. This portion of the argument is substantially different from the approach of Robertson and Seymour to minor testing and comprises the major new development in the proof.

Applications

An *immersion* of a graph H into a graph G is defined like a topological embedding, except that the paths in G corresponding to the edges of H are only required to be pairwise edge disjoint instead of pairwise internally vertex disjoint. Formally, an immersion of H into G is a mapping ν that associates with each vertex $v \in V(H)$ a distinct vertex $\nu(v) \in V(G)$ and with each edge $e = vw \in E(H)$ a path $\nu(e)$ in G with endpoints $\nu(v)$ and $\nu(w)$ in such a way that the paths $\nu(e)$ for $e \in E(H)$ are mutually edge disjoint. Robertson and Seymour [14] showed that graphs are well quasi-ordered under the immersion relation, proving a conjecture of Nash-Williams. In [8], the authors give a construction which implies the following corollary of Theorem 1.

Corollary 1 *For every fixed undirected graph H, there is an $O(|V(G)|^3)$ time algorithm that decides if there is an immersion of H into G.*

Again, the algorithm is uniform in H, which implies that the immersion problem is fixed parameter tractable. This answers another open question by Downey and Fellows [4, 5]. Corollary 1 also holds for the more restrictive "strong immersion" version, where $\nu(v)$ cannot be the internal vertex of the path $\nu(e)$ for any $v \in V(G)$ and $e \in E(G)$.

Recommended Readings

1. Bodlaender H (2006) Treewidth: characterizations, applications, and computations. In: Graph-theoretic concepts in computer science. Lecture notes in computer science, vol 4271. Springer, Berlin/Heidelberg, pp 1–14
2. Bodlaender H, Koster A (2008) Combinatorial optimization on graphs of bounded treewidth. Comput J 51(3): 255–269
3. Courcelle B (1990) The monadic second-order logic of graphs I: Recognizable sets of finite graphs. Info Comput 85:12–75
4. Downey RG, Fellows MR (1992) Fixed-parameter intractability. In: Proceedings of the seventh annual structure in complexity theory conference, Boston, npp 36–49
5. Downey RG, Fellows MR (1999) Parameterized complexity. Monographs in computer science. Springer, New York
6. Fortune S, Hopcroft JE, Wyllie J (1980) The directed subgraph homeomorphism problem. Theor Comput Sci 10:111–121
7. Garey MR, Johnson DS (1979) Computers and intractability. W.H. Freeman, San Francisco
8. Grohe M, Kawarabayashi K, Marx D, Wollan P (2011) Finding topological subgraphs is fixed parameter tractable. In: STOC'11 proceedings of the 43rd ACM symposium on theory of computing, San Jose, pp 479–488
9. Grohe M, Marx D (2012) Structure theorem and isomorphism test for graphs with excluded topological subgraphs. In: STOC '12 proceedings of the forty-fourth annual ACM symposium on theory of computing, New York, p 173–192
10. Kawarabayashi K, Wollan P (2011) A shorter proof of the graph minor algorithm: the unique linkage theorem. In: STOC'10 proceedings of the 42rd ACM symposium on theory of computing, Cambridge, pp 687–694
11. Kawarabayashi K, Thomas R, Wollan P(2013) A new proof of the flat wall theorem. http://arxiv.org/abs/1207.6927
12. LaPaugh AS, Rivest RL (1980) The subgraph homeomorphism problem. J Comput Syst Sci 20(2):133–149
13. Robertson N, Seymour PD (1995) Graph minors XIII: the disjoint paths problem. J Combin Theory Ser B 63:65–110
14. Robertson N, Seymour PD (2010) Graph minors XXIII: Nash Williams' immersion conjecture. J Combin Theory Ser B 100(2):181–205
15. Scheffler P (1989) Linear time algorithms for graphs of bounded tree-width: the disjoint paths problem. Dresdner Reihe Forschung 5(9):49–52

First Fit Algorithm for Bin Packing

Gyorgy Dosa
University of Pannonia, Veszprém, Hungary

Keywords

Approximation ratio; Bin packing; First fit; Tight result

Years and Authors of Summarized Original Work

1974; Johnson, Demers, Ullman, Garey, Graham
2013; Dósa, Sgall

Problem Definition

In the classical bin packing (BP) problem, we are given a set of items with rational sizes between 0 and 1, and we try to pack them into a minimum number of bins of unit size so that no bin contains items with total size more than 1. The problem definition originates in the early 1970s: Johnson's thesis [10] on bin packing together with Graham's work on scheduling [8, 9] (among other pioneering works) started and formed the whole area of approximation algorithms. The First Fit (FF) algorithm is one among the first algorithms which were proposed to solve the BP problem and analyzed in the early works. FF performs as follows: The items are first given in some list L and then are handled by the algorithm in this given order. Then, algorithm FF packs each item into the first bin where it fits; in case the item does not fit into any already opened bin, the algorithm opens a new bin and puts the actual item there. A closely related algorithm is Best Fit (BF); it packs the items also according to a given list, but each item is packed into the most full bin where it fits or the item is packed into a new bin only if it does not fit into any open bin. If the items are ordered in the list by decreasing sizes, the algorithms are called as FFD (first fit decreasing) and BFD (best fit decreasing).

Applications

There are many applications for bin packing (in industry, computer science, etc), and BP has many different versions. It is worth noting that BP has a strong relationship to the area of scheduling. So the scientific communities of packing and scheduling are almost the same.is a major

Key Results

It was immediately shown in the early works [6,12,15] that the asymptotic approximation ratio of FF and BF bin packing is 1.7. It means that if the optimum packing needs OPT bins, algorithm FF never uses more than $1.7 \cdot OPT + C$ bins, where C is a fixed constant (The same holds for the BF algorithm). It is easy to see that the multiplicative factor, i.e., 1.7, cannot be smaller. But the minimum value of the C constant, for which the statement remains valid, is not a simple issue.

First, Ullman in 1971 [15] showed that C can be chosen to be 3. But this is not the best choice.

Soon, the additive term was decreased in [6] to 2 and then in [7] to $FF \leq \lceil 1.7 \cdot OPT \rceil$; since both FF and OPT denote integer numbers, this is the same as $FF \leq 1.7 \cdot OPT + 0.9$.

Then, for many years, no new results were published regarding the possible decreasing of the additive term.

Another direction is considered in the many-times-cited work of Simchi-Levy [14]. He showed that the absolute approximation ratio of FF (and BF) is at most 1.75. It means that if we do not use an additive term in the inequality, then $FF \leq 1.75 \cdot OPT$ is valid.

Now, if we are interested in the tight result, we have two options. One is that we can try to decrease the multiplicative factor in the inequality of the absolute approximation ratio, i.e., the question is the following: What is the smallest number, say α, that can be substituted in the place of 1.75 such that the inequality $FF \leq \alpha \cdot OPT$ is valid for any input? The other direction is the following: What is the smallest possible value of the additive constant C such that the $FF \leq 1.7 \cdot OPT + C$ inequality is true for every input?

The next step was made independently from each other in two works. Xia and Tan [17] and Boyar et al. [1] proved that the absolute approximation ratio of FF is not larger than $12/7 \approx 1.7143$.

Moreover, [17] also dealt with the other direction and decreased the value of C to $FF \leq 1.7 \cdot OPT + 0.7$.

If we are interested in how much the additive term (or the α factor) can be decreased, we must also deal with the lower bound of the algorithm. Regarding this, the early works give examples for both the asymptotic and absolute ratios. For the asymptotic bound, there exists such input for which $FF = 17k$ holds whenever $OPT = 10k + 1$; thus, the asymptotic upper bound 1.7 is tight, see [6, 12, 15]. For the absolute ratio, an example is given with $FF = 17$ and $OPT = 10$, i.e., an instance with approximation ratio exactly 1.7 [6, 12]. But no example was shown for large values of OPT.

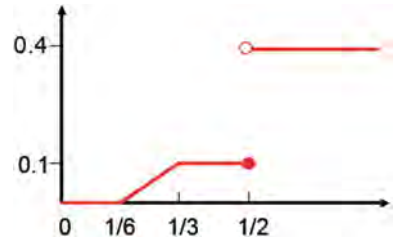
It means that soon, it turned out that the value of the multiplicative factor of the absolute approximation ratio (i.e., α) cannot be smaller than 1.7 or, regarding the another measure, the additive constant cannot be chosen to be smaller than zero. But this remained an open question for 40 years whether the smallest possible choice of α is really 1.7 or, in other words, the smallest possible choice of the additive term is really zero.

Finally, the papers [3, 4] answered the question. Lower-bound instances are given with $FF = BF = \lfloor 1.7 \cdot OPT \rfloor$ for any value of OPT, and it is also shown that $FF = BF \leq \lfloor 1.7 \cdot OPT \rfloor$ holds for any value of OPT. So this is the tight bound which was looked for 40 years.

Methods

To prove the upper bound, the main technique is the usage of a weighting function. Any item gets some weight according to its size. Then, to get the asymptotic ratio, it is only needed that any optimal bin has a weight at most 1.7 and any bin in the FF packing (with bounded exception) has a weight at least 1.

In the recent paper [13], a nice and surprising idea is presented: The same weight function that was used traditionally in the analysis is divided into two parts: scaled size and bonus. Thus, the weight of any item a is $w(a) = s(a) + b(a)$, where $s(a) = 6/5 \cdot a$ is the scaled size of the item and the remaining part $b(a)$ is the bonus of the item, which is defined as follows:



First Fit Algorithm for Bin Packing, Fig. 1 The bonus function

$b(a)$ is zero if the size of the item is below 1/6. The bonus is just 0.1 if a is between 1/3 and 1/2, and it is 0.4 if the size is above 1/2. Between 1/6 and 1/3, the bonus function is continuous and linear. We emphasize that this is the *same* old weighting function, only in a new costume. The bonus function can be seen in Fig. 1.

By this separation, it is easy to show that the weight of any optimal bin is at most 1.7, and this implies that the weight of the whole instance is at most $1.7 \cdot OPT$.

The key part is to show that on average, the weight of each FF bin is at least 1. This property trivially holds if the total size of the items in the bin is at least 5/6. It is not hard to handle the bins with single items; here, almost all of them must be bigger than 1/2, and such items have huge bonus (i.e., 0.4), together with the scaled size, that is, at least 0.6, we are again done. In the remaining bins, the next tricky calculation is used: The scaled size of the bin plus the bonus of the *following* bin is at least 1. By this trick, the proof will be almost done, but several further examinations are also needed for completing the tightness result.

New Lower Bound Construction

The lower bound construction works in the following way. Suppose, for the sake of simplicity,

that OPT = 10k for some integer k, and let $\varepsilon > 0$, a small value.

The input consists of OPT small items of size approximately 1/6, followed by OPT medium-sized items of size approximately 1/3, followed by OPT large items of size exactly $1/2 + \varepsilon$. The optimum packs in each bin one item from each group. FF packs the small items into 2k bins with 5 items with the exception of the first and last of these bins, which will have 6 and 4 items, respectively. The sizes of items differ from 1/3, or differ from 1/6, in both directions by a small amount δ_i. Finally, every large item will occupy its own bin.

In the original construction, the choice of the small and medium-sized items is a bit difficult, so one could think that the construction *must* be so difficult, and thus, the construction cannot be tightened. It turns out, however, that this is not the case. The construction can be modified in the way that δ_i is exponentially decreasing but remains greater than ε for all i. This guarantees that only the item with the largest δ_i in a bin is relevant for its final size, and this in turn enables us to order the items so that no additional later item fits into these bins. Thus, by the modification, not only the construction is simpler but it also makes possible to prove the tightness.

Open Problems

There are many open problems regarding bin packing. For example, the tight absolute approximation ratio of BFD is an open question (For FFD, it was recently proved that $FFD \leq (11/9) \cdot OPT + 6/9$ and this is the tight result, see [5].).

Cross-References

- Current Champion for Online Bin Packing
- Harmonic Algorithm for Online Bin Packing
- Lower Bounds for Online Bin Packing
- Selfish Bin Packing Problems
- Subset Sum Algorithm for Bin Packing

Recommended Readings

1. Boyar J, Dósa G, Epstein L (2012) On the absolute approximation ratio for First Fit and related results. Discret Appl Math 160:1914–1923
2. Coffman EG, Garey MR, Johnson DS (1997) Approximation algorithms for bin packing: a survey. In: Hochbaum D (ed) Approximation algorithms. PWS Publishing Company, Boston
3. Dósa G, Sgall J (2013) First Fit bin packing: a tight analysis. In: Proceedings of the 30th symposium on theoretical aspects of computer science (STACS), LIPIcs vol 3. Schloss Dagstuhl, Kiel, Germany, pp 538–549
4. Dósa G, Sgall J (2014) Optimal analysis of Best Fit bin packing. In: Esparza J et al (eds) ICALP 2014. LNCS, part I, vol 8572. Springer, Heidelberg, Copenhagen, Denmark, pp 429–441
5. Dósa G, Li R, Han X, Tuza Zs (2013) Tight absolute bound for First Fit Decreasing bin-packing: FFD(L) <= 11/9 OPT(L) + 6/9. Theor Comput Sci 510:13–61
6. Garey MR, Graham RL, Ullman JD (1973) Worst-case analysis of memory allocation algorithms. In: Proceedings of the 4th symposium on theory of computing (STOC). ACM, Denver, Colorado, USA, pp 143–150
7. Garey MR, Graham RL, Johnson DS, Yao ACC (1976) Resource constrained scheduling as generalized bin packing. J Combin Theory Ser A 21:257–298
8. Graham RL (1966) Bounds for certain multiprocessing anomalies. Bell Syst Tech J 45:1563–1581
9. Graham RL (1969) Bounds on multiprocessing timing anomalies. SIAM J Appl Math 17:263–269
10. Johnson DS (1973) Near-optimal bin packing algorithms. PhD thesis, MIT, Cambridge, MA
11. Johnson DS (1974) Fast algorithms for bin packing. J Comput Syst Sci 8:272–314
12. Johnson DS, Demers A, Ullman JD, Garey MR, Graham RL (1974) Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM J Comput 3:256–278
13. Sgall J (2012) A new analysis of Best Fit bin packing. In: Kranakis et al (eds) Proceedings of the 6th international conference FUN with algorithms. LNCS, vol 7288. Springer, Venice, Italy, pp 315–321
14. Simchi-Levi D (1994) New worst case results for the bin-packing problem. Naval Res Logist 41:579–585
15. Ullman JD (1971) The performance of a memory allocation algorithm. Technical report 100, Princeton University, Princeton
16. Williamson DP, Shmoys DB (2011) The design of approximation algorithms. Cambridge University Press, Cambridge
17. Xia B, Tan Z (2010) Tighter bounds of the First Fit algorithm for the bin-packing problem. Discret Appl Math 158:1668–1675

Fixed-Parameter Approximability and Hardness

Guy Kortsarz
Department of Computer Science, Rutgers University, Camden, NJ, USA

Keywords

Approximation; Exponential time conjecture; Fixed parameter; Inapproximability

Years and Authors of Summarized Original Work

1995; Alon, Yuster, Zwick
2005; Marx
2006; McCartin
2013; Hajiaghayi, Kortsarz

Problem Definition

NP-hard problems are believed to be intractable. This is the widely believed assumption that $P \neq NP$. For all our problems, the size of their input is denoted by n. In parameterized complexity, the input is refined to (I, k) with k a parameter related to the input, and the goal is to find an exact algorithm for the problem that runs in time $f(k) \cdot n^{O(1)}$, for some function f. In this survey, we parameterize by the optimum value of the instance unless stated otherwise. In addition, the optimum is always integral. In approximation algorithms, a ρ approximation for a minimization (maximization) problem P is a polynomial time algorithm A, such that for any instance I, A returns a solution of value $A(I)$ and $A(I)/\text{OPT}(I) \leq \rho$ ($\text{OPT}(I)/A(I) \leq \rho$) with $\text{OPT}(I)$ the optimum value for the instance. In both subjects, there are intractability results. The class FPT are the problems that admit an $f(k)n^{O(1)}$ time, exact solution for some function f. The classes $W[i]$ for every integer $i \geq 1$ satisfy FPT $\subseteq W[1] \subseteq W[2] \subseteq \ldots$. It is widely believed that all inclusions are strict. Consider the CLIQUE problem. Given a graph $G(V, E)$, a subset $U \subseteq V$, forms a *clique*, if for every $u, v \in U$, $(u, v) \in E$. The problem is

Input: A graph G and a parameter k.
Question: Is there in G a clique U of size $|U| \geq k$?

In [21], it is proved that CLIQUE admits no $n^{1-\epsilon}$ approximation unless $P = NP$. It is known that CLIQUE is $W[1]$-complete. Thus it is considered highly unlikely that CLIQUE $\in$ FPT. The SETCOVER problem is defined as follows:

Input: A universe U and a collection $\mathcal{S} = \{S_i\}$ of subsets of $\mathcal{U}$ and a parameter k.
Question: Is there a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ containing at most k sets so that $\bigcup_{S_i \in \mathcal{S}'} S_i = \mathcal{U}$?

SETCOVER is $W[2]$-complete. In addition, Raz and Safra [27] show that unless $P = NP$, SETCOVER admits no $c \ln n$ algorithm for some constant c, almost matching the simple greedy $\ln n + 1$ ratio approximation algorithm.

Our Subject

Formally, we deal with the following subject: An algorithm for a minimization (resp., maximization) problem P is called an (r, t)-FPT-approximation algorithm for P with input parameter k, if the algorithm takes as input an instance I with value OPT and an integer parameter k and either computes a feasible solution to I with value at most $k \cdot r(k)$ (resp., at least $k/r(k)$ and $k/r(h) = o(k)$) or computes a certificate that $k <$ OPT (resp., $k >$ OPT) in time $t(k) \cdot |I|^{O(1)}$. The requirement that $k/r(k) = o(k)$ avoids returning a single vertex in the clique problem, claiming OPT approximation.

A problem is called (r, t)-FPT-inapproximable (or, (r, t)-FPT-hard) if it does not admit an (r, t)-FPT-approximation algorithm. An FPT approximation is mainly interesting if the problem is $W[1]$-hard and allowing running time $f(k) \cdot n^{O(1)}$ gives improved approximation. We restrict our attention to this scenario. Thus, we do not discuss many subjects such as approximations in OPT that run in *polynomial*

Supported in part by NSF grant number 1218620.

time in n and upper and lower bounds, on algorithms with sub exponential time in n, for several conbinatorial problems.

Our Complexity Assumption

We assume the following conjecture throughout Impagliazzo et al. [4] conjectured the following:

> *Exponential Time Hypothesis (*ETH*)*
> 3-SAT cannot be solved in $2^{o(q)}(q+m)^{O(1)}$ time where q is the number of variables and m is the number of clauses.

The following is due to [4].

Lemma 1 *Assuming* ETH, *3-SAT cannot be solved in* $2^{o(m)}(q+m)^{O(1)}$ *time where* q *is the number of variables and* m *is the number of clauses.*

It is known that the ETH implies that $W[1] \neq$ FPT. This implies that $W[2] \neq$ FPT as well.

Key Results

We survey some FPT-approximability and FPT-inapproximability results. Our starting point is a survey by Marx [23], and we also discuss recent results. The simplest example we are aware of in which combining FPT running time and FPT-approximation algorithm gives an improved result is for the strongly connected directed subgraph (SCDS) problem.

Input: A directed graph $G(V, E)$, a set $T = \{t_1, t_2, \ldots, t_p\}$ of terminals, and an integer k.
Question: Is there a subgraph $G'(V, E')$ so that $|E'| \leq k$ and for every $t_i, t_j \in T$, there is a directed path in G' from t_i to t_j and vice versa?

The problem is in $W[1]$-hard. The best approximation algorithm known for this problem is n^{ϵ} for any constant ϵ. See Charikar et al. [5].

The following is due to [7].

Theorem 1 *The SCDS problem admits an* FPT *time* 2 *approximation ratio.*

Proof The directed Steiner tree problem is given a directed edge-weighted graph and a root r and a set $T = \{t_1, t_2, \ldots, t_p\}$ of terminals; find a minimum cost-directed tree rooted by r containing T. This problem belongs to FPT. See Dreyfus and Wagner [12]. Note that for every terminal t_i, any feasible solution contains a directed tree from t_i to T and a a reverse-directed Steiner tree from T to t_i. These two problems can be solved optimally in FPT time. In the second application, we reverse the direction of edges before we find the directed Steiner tree. Moreover, two such trees give a feasible solution for the SCDS problem as every two terminals t_j, t_k have a path via t_i. Clearly, the solution has value at most $2 \cdot$ OPT with OPT the optimum value for the SCDS instance. The claim follows.

Definition 1 A *polynomial time approximation scheme* (PTAS) for a problem P is a $1+\epsilon$ approximation for any constant ϵ that runs in time $n^{f(1/\epsilon)}$. An *EPTAS* is such an algorithm that runs in time $f(1/\epsilon)n^{O(1)}$.

The *vertex cover* problem is to select the smallest possible subset U of V so that for every edge (u, v), either $u \in U$ or $v \in U$ (or both). In the *partial vertex cover problem*, a graph $G(V, E)$ and an integer k are given. The goal is to find a set U of k vertices that is touched by the largest number of edges. An edge (u, v) is touched by a set U if $u \in U$ or $v \in U$ or both. It is known that this problem admits no *PTAS* unless $P = NP$ (see Dinur and Safra [10]). The corresponding minimum partial vertex cover problem requires a set of k vertices touched by the *least number of edges*. This problem admits no better than 2-ratio, under the *small set expansion conjecture*. See [15]. Both problems belong to $W[1]$-hard. The following theorem of [23] relies on a technique called *color coding* [1].

Theorem 2 ([23]) *For every constant* ϵ*, the partial vertex cover problem (and in a similar proof the minimum partial vertex cover problem) admits an EPTAS that runs in time* $f(k, 1/\epsilon) \cdot n^{O(1)}$ *with* n *the number of vertices in the graph.*

Proof Let $D = \binom{k}{2}/\epsilon$. Sort the vertices $v_1, v_2, \ldots, v_n$ by nonincreasing degrees. If for

the largest degree, $d(v_1)$ satisfies $d(v_1) \geq D$, the algorithm outputs the set $\{v_1, v_2, \ldots, v_k\}$. These k vertices cover at least $\sum_{i=1}^{k} \deg(v_i) - \binom{k}{2}$ edges. Clearly, OPT $\leq \sum_{i=1}^{k} \deg(v_i)$. Hence, the value of the constructed solution is at least

$$\frac{\sum_{i=1}^{k} \deg(v_i) - \binom{k}{2}}{\sum_{i=1}^{k} \deg(v_i)} \geq 1 - \frac{\binom{k}{2}}{D} \geq 1 - \frac{\epsilon}{2} \geq \frac{1}{1+\epsilon}$$

times the optimum for a $1+\epsilon$ approximation. In the other case, the optimum OPT $\leq k \cdot D$. We guess the correct value of the optimum by trying all values between $1, \ldots, k \cdot D$. Fix the run with the correct OPT. Let E^* be the set of OPT edges that are touched by the optimum. An OPT labeling is an assignment of a label in $\{1, \ldots, \text{OPT}\}$ to the edges of E. We show that if the labels of E^* are pairwise distinct, we can solve the problem in time $h(k, 1/\epsilon)$. Let $\{u_1, u_2, \ldots, u_k\}$ be the optimum set. Let L_i be the labels of the edges of u_i. As all labels of E^* are pairwise distinct, $\{L(u_i)\}$ is a disjoint partition of all labels (as otherwise there is a labeling with less than OPT labels). The number of possible partitions of the labels into k sets is at most k^{OPT}. Given the *correct* partition $\{L_i\}$, we need to match every L_i with a vertex u_i so that the labels of u_i are L_i. This can be done in polynomial time by matching computation. To get a labeling with different pairwise labels on E^*, we draw for every edges a label between 1 and OPT, randomly and independently. The probability that the labels of E^* are disjoint is more than $1/\text{OPT}^{\text{OPT}}$. Repeating the random experiment for OPT^{OPT} times implies that with probability at least $1 - 1/e$, one of the labeling has different pairwise labels for E^*. This result can be derandomized [1].

We consider one example in which OPT is not the parameter [23]. Consider a graph that contains a set $X = \{x_1, \ldots, x_k\}$. so that $G \setminus X$ is a planar graph. Thus the parameter here is the number of vertices that need to be removed to make the graph. Consider the minimum coloring problem on G. We can determine the best coloring of X in time k^k. Then we can color $G \setminus X$ by four (different) colors. A simple calculation shows that this algorithm has approximation ratio at most $7/3$.

The following is a simple relation exist between *EPTAS* and FPT theory.

Proposition 1 *If an optimization problem P admits an EPTAS, then $P \in$ FPT.*

Proof We prove the theorem for minimization problems. For maximization problems, the proof is similar. Assume that P has a $1+\epsilon$ approximation that runs in time $f(1/\epsilon) \cdot n^{O(1)}$. Set $\epsilon = 1/(2k)$. Using the EPTAS algorithm gives an $f(2k)n^{O(1)}$ time $(1+\epsilon)$ approximation. If the optimum is at most k, we get a solution of size at most $(1+\epsilon)k = k + 1/2 < k + 1$. As the solution is integral, the cost is at most k. If the minimum is $k+1$, the approximation will not return a better than $k+1$ size solution. Thus the approximation returns cost at most k if and only if there is a solution of size at most k.

Thus we can rule out the possibility of an *EPTAS* if a problem is $W[1]$-hard. For example, this shows that the maximum independent set for unit disks graphs admits no *EPTAS* as it belongs to $W[1]$. See many more examples in [23]. Chen Grohe and Grüber [6] provide an early discussion of our topic. Lange wrote a PDF presentation for recent FPT approximation. The following theorem is due to Grohe and Grüber (see [19]).

Theorem 3 *If a maximization problem admits an FPT-approximation algorithm with performance ratio $\rho(k)$, then for some function ρ', there exists a $\rho'(k)$ polynomial time approximation algorithm for the problem.*

In the traveling salesperson with a deadline, the input is a metric on n points and a set $D \subseteq V$ with each $v \in D$ having a deadline t_v. A feasible solution is a simple path containing all vertices, so that *for every* $v \in D$, the length of the tour until v is at most t_v. The problem admits no constant approximation and is not in FPT when parameterized by $|D|$. See Bockenhauer, Hromkovic, Kneis, and Kupke [2]. In this entry, the authors give a 2.5 approximation that runs in time $n^{O(1)} + |D|! \cdot |D|$. The parameterized

undirected multicut problem is given an undirected graph and a collection $\{s_i, t_i\}_{i=1}^{m}$ of pairs, and a parameter k is possible to remove at most k edges and disconnect all pairs. Garg, Vazirani, and Yannakakis give an $O(\log n)$ approximation for the problem [16]. In 2009, it was given a ratio 2 fixed-parameter approximation (Marx and Razgon) algorithm. However, Marx and Razgon [25] and Bousquet et al. [3] show that this problem is in fact in FPT. Fellows, Kulik, Rosamond, and Shachnai give the following tradeoff (see [14]). The best known exact time algorithm for the vertex cover problem has running time 1.273^k. The authors show that if we settle for an approximation result, then the running time can be improved. Specifically, they gave $\alpha \geq 1$ approximation for vertex cover that runs in time $1.237^{(2-\alpha)k}$. The minimum edge dominating set problem is given a graph and a parameter k, and there is a subset $E' \subset E$ of size at most k so that every edge in $E \setminus E'$ is adjacent to at least one edge in E'. Escoffier, Monnot, Paschos, and Mingyu Xiao (see [13]) prove that the problem admits a $1 + \epsilon$ ratio for any $0 \leq \epsilon \leq 1$ that runs in time $2^{(2-\epsilon)\cdot k}$. A kernel for a problem P is a reduction from an instance I to an instance I' whose size is $g(k)$, namely, a function of k, so that a yes answer for I implies a yes answer for I' and a no answer for I implies a no answer for I'. If a kernel exists, it is clear that $P \in$ FPT. However, the size of the kernel may determine what is the function of k in the $f(k)\cdot n^{O(1)}$ exact solution. The following result seems interesting because it may not be intuitive. In the *tree deletion* problem, we are given a graph $G(V, E)$ and a number k and the question is if we can delete up to k vertices and get a tree. Archontia Giannopoulou, Lokshtanov, Saket, and Suchy prove (see [17]) that the tree deletion problem admits a kernel of size $O(k^4)$. However, the problem does not admit an approximation ratio of OPT^c for any constant c.

Other Parameters

An *independent set* is a set vertices so that no two vertices in the set share an edge. In parameterized version given k, the question is if there is an independent set of size at least k. Clearly, the problem is $W[1]$-complete. Grohe [18] show that the maximum independent set admits a FPT-approximation scheme if the parameter is the genus of the graph. E. D. Demaine, M. Hajiaghayi, and K. Kawarabayashi [9] showed that vertex coloring has a ratio 2 approximation when parameterized by the *genus* of a graph. The tree augmentation problem is given an edge-weighted graph and a spanning tree whose edges have cost 0; find a minimum cost collection of edges to add to the tree, so that the resulting graph is 2-edge connected. The problem admits several polynomial time, ratio 2, and approximation algorithms. Breaking the 2 ratio for the problem is an important challenge in approximation algorithms. Cohen and Nutov parameterized the problem by the diameter D of the tree and gave an $f(D) \cdot n^{O(1)}$ time, $1 + \ln 2 < 1.7$ approximation algorithm for the problem [8].

Fixed-Parameter Inapproximability

The following inapproximability is from [11]. The additive maximum independent set problem is given a graph and a parameter k and a constant c, and the question is if the problem admits an independent set of size at least $k - c$ or no independent set of size k exists.

It turns out that the problem is equivalent to the independent set problem.

Theorem 4 *Unless* $W[1] =$ FPT *(hence, under the* ETH*), the independent set problem admits no additive c approximation.*

Proof Let I be the instance. Find the smallest d so that

$$\left\lceil \frac{dk - c}{d} \right\rceil \geq k.$$

Output d copies of G and let $k \cdot d$ be the parameter of the new instance I'. We show that the new graph has independent set of size $dk - c$ if and only if the original instance has an independent set of size k. If the original instance has an independent set of size k, taking union of d independent sets, we get an independent set of size $k \cdot d$.

Now say that I' has an independent set of size $dk - c$. The average size of an independent set in a graph in I' is then $(dk - c)/d$. Since the size

of the independent set is integral, there is a copy that admits an independent set of size

$$\left\lceil \frac{dk - c}{d} \right\rceil \geq k.$$

An independent set I is *maximal* if for every $v \notin I$, $v + I$ is not an independent set. The problem of minimum size maximal independent set (MSDIS) is shown to be completely inapproximable in [11]. Namely, this problem is $(r(k), t(k))$-FPT-hard for any r, t, unless FPT $=$ $W[2]$ (hence, under the ETH). The problem admits no $n^{1-\epsilon}$ approximation (see [20]). In the min-WSAT problem, a Boolean circuit is given and the task is to find a satisfying assignment of minimum weight. The weight of an assignment is the number of true variables. Min-WSAT was given a complete inapproximability by Chen, Grohe, Grüber (see [6]) 2006.

The above two problems are not monotone. This implies that the above results are non-surprising. The most meaningful complete inapproximability is given by Marx [24] who shows that the weighted circuit satisfiability for monotone or antimonotone circuits is completely FPT inapproximable.

Of course, if the problem has almost no gap, namely, the instance can have value k or $k - 1$, it is hard to get a strong hardness.

A natural question is if we can use gap reductions from approximation algorithms theory to get some strong lower bounds, in particular for clique and setcover. It turns out that this is very difficult even under the ETH conjecture. This subject is related to almost linear PCP (see [26]). In this entry, Moshkovitz poses a conjecture called *the projection game conjecture* (PGC). M. Hajiaghayi, R. Khandekar, and G. Kortsarz show the following theorem.

Theorem 5 *Under the* ETH *and* PGC *conjectures,* SETCOVER *is* (r, t)-FPT-*hard for* $r(k) = (\log k)^{\gamma}$ *and* $t(k) = \exp(\exp((\log k)^{\gamma})) \cdot \text{poly}(n) = \exp\left(k^{(\log^f k)}\right) \cdot \text{poly}(n)$ *for some constant* $\gamma > 1$ *and* $f = \gamma - 1$.

Cross-References

▸ Color Coding

Recommended Readings

1. Alon N, Yuster R, Zwick U (1995) Color coding. J ACM 42(4):844–856
2. Böckenhauer HJ, Hromkovic J, Kneis J, Kupke J (2007) The parameterized approximability of TSP with deadlines. Theory Comput Syst 41(3):431–444
3. Bousquet N, Daligault J, Thomassé S (2011) Multicut is FPT. In: STOC'11, New York, pp 459–468
4. Calabro C, Impagliazzo R, Paturi R (1995) A duality between clause width and clause density for SAT. In: Computational Complexity, Prague, pp 252–260
5. Charikar M, Chekuri C, Cheung TY, Dai Z, Goel A, Guha S, Li M (1999) Approximation algorithms for directed steiner problems. J Algorithms 33(1):73–91
6. Chen Y, Grohe M, Grüber M (2006) On parameterized approximability. In: IWPEC 2006, Zurich, pp 109–120
7. Chitnis R, Hajiaghayi MT, Kortsarz G (2013) Fixed parameter and approximation algorithms: a new look. In: IPEC, Sophia Antipolis, pp 110–122
8. Cohen N, Nutov Z (2013) A (1+ln2)-approximation algorithm for minimum-cost 2-edge-connectivity augmentation of trees with constant radius. Theor Comput Sci 489–490: 67–74
9. Demaine E, Hajiaghayi MT, Kawarabayashi KI (2005) Algorithmic graph minor theory: decomposition, approximation, and coloring. In: FOCS 2005, Pittsburgh, pp 637–646
10. Dinur I, Safra S (2002) On the importance of being biased. In: STOC 2002, Québec, pp 33–42
11. Downey R, Fellows M (1995) Parameterized approximation of dominating set problems. Inf Process Lett 109(1):68–70
12. Dreyfus SE, Wagner RA (1971) The steiner problem in graphs. Networks 1(3):195–207
13. Escoffier B, Monnot J, Paschos VT, Xiao M (2012) New results on polynomial inapproximability and fixed parameter approximability of edge dominating set. In: IPEC, Ljubljana, pp 25–36
14. Fellows MR, Kulik A, Rosamond FA, Shachnai H (2012) Parameterized approximation via fidelity preserving transformations. In: ICALP, Warwick, pp 351–362
15. Gandhi R, Kortsarz G (2014) On set expansion problems and the small set expansion conjecture. In: WG, Nouan-le-Fuzelier, pp 189–200
16. Garg N, Vazirani VV, Yannakakis M (1996) Approximate max-flow min-(multi)cut theorems and their applications. SIAM J Comput 25(2):235–251

17. Giannopoulou A, Lokshtanov D, Saket S, Suchy O (2013) Tree deletion set has a polynomial kernel (but no $OPT^O(1)$ approximation). arXiv:1309.7891
18. Grohe M (2003) Local tree-width, excluded minors, and approximation algorithms. Combinatorica 23(4):613–632
19. Grohe M, Grüber M (2007) Parameterized approximability of the disjoint cycle problem. In: ICALP 2007, Wroclaw, pp 363–374
20. Halldórsson MM (1993) Approximating the minimum maximal independence number. Inf Process Lett 46(4):169–172
21. Hastad J (1996) Clique is hard to approximate within $n^{1-\epsilon}$. In: FOCS, Burlington, pp 627–636
22. Impagliazzo R, Paturi R, Zane F (1998) Which problems have strongly exponential complexity? J Comput Syst Sci 63(4):512–530
23. Marx D (2005) Parameterized complexity and approximation algorithms. Comput J 51(1): 60–78
24. Marx D (2013) Completely inapproximable monotone and antimonotone parameterized problems. J Comput Syst Sci 79(1):144–151
25. Marx D, Razgon I (2014) Fixed-parameter tractability of multicut parameterized by the size of the cutset. SIAM J Comput 43(2):355–388
26. Moshkovitz D (2012) The projection games conjecture and the NP-hardness of ln n-approximating set-cover. In: APPROX-RANDOM 2012, Boston, pp 276–287
27. Raz R, Safra S (1997) A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In: STOC 1997, San Diego, pp 475–484

Floorplan and Placement

Yoji Kajitani
Department of Information and Media Sciences, The University of Kitakyushu, Kitakyushu, Japan

Keywords

Alignment; Dissection; Layout; Packing

Years and Authors of Summarized Original Work

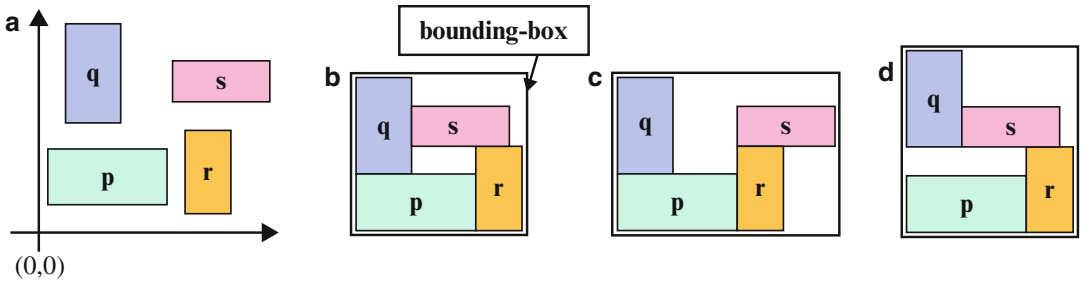
1994; Kajitani, Nakatake, Murata, Fujiyoshi

Problem Definition

The problem is concerned with efficient coding of the constraint that defines the placement of objects on a plane without mutual overlapping. This has numerous motivations, especially in the design automation of integrated semiconductor chips, where almost hundreds of millions of rectangular modules shall be placed within a small rectangular area (chip). Until 1994, the only known coding efficient in computer-aided design was *Polish-Expression* [1]. However, this can only handle a limited class of placements of the *slicing structure*. In 1994 Nakatake, Fujiyoshi, Murata, and Kajitani [2] and Murata, Fujiyoshi, Nakatake, and Kajitani [3] were finally successful to answer this long-standing problem in two contrasting ways. Their code names are *Bounded-Sliceline-Grid* (BSG) for floorplanning and *Sequence-Pair* (SP) for placement.

Notations

1. *Floorplanning, placement, compaction, packing, layout:* Often they are used as exchangeable terms. However, they have their own implications to be used in the following context. *Floorplanning* concerns the design of the plane by restricting and partitioning a given area on which objects are able to be properly *placed. Packing* tries a placement with an intention to reduce the area occupied by the objects. *Compaction* supports packing by pushing objects to the center of the placement. The result, including other environments, is the *layout*. BSG and SP are paired concepts, the former for "floorplanning," the latter for "placement."
2. *ABLR-relation:* The objects to be placed are assumed rectangles in this entry though they could be more general depending on the problem. For two objects p and q, p is said to be *above* q (denoted as pAq) if the bottom edge (boundary) of p is above the top edge of q. Other relations with respect to "*below*"



Floorplan and Placement, Fig. 1 (**a**) A feasible placement whose ABLR-relations could be observed differently. (**b**) Compacted placement if ABLR-relations are (qLr), (sAp), ... Its sequence-pair is SP = (qspr,pqrs) and single-sequence is SS = (2413). (**c**) Compacted placement for (qLr), (sRp), ... SP = (qpsr,pqrs). SS = (2143). (**d**) Compacted placement if (qAr), (sAp), ... SP = (qspr,prqs). SS = (3412)

(pBq), "*left-of*" (pLq), and "*right-of*" (pRq) are analogously defined. These four relations are generally called *ABLR-relations*.

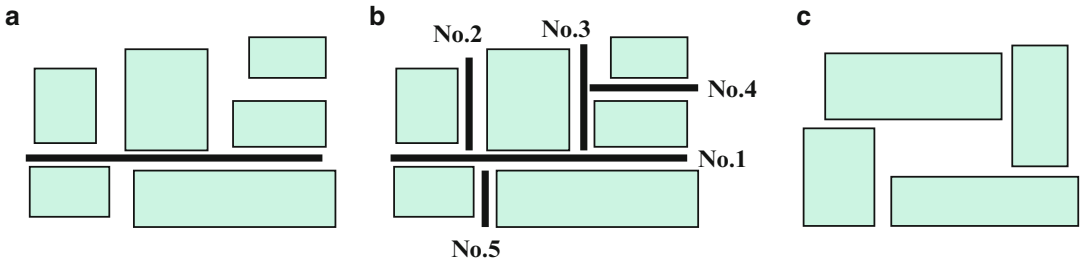
A placement without mutual overlapping of objects is said to be *feasible*. Trivially, a placement is feasible if and only if every pair of objects is in one of ABLR-relations. The example in Fig. 1 will help these definitions.

It must be noted that a pair of objects may satisfy two ABLR-relations simultaneously, but not three. Furthermore, an arbitrary set of ABLR-relations is not necessarily *consistent* for any feasible placement. For example, any set of ABLR-relations including relations (pAq), (qAr), and (rAp) is not consistent.

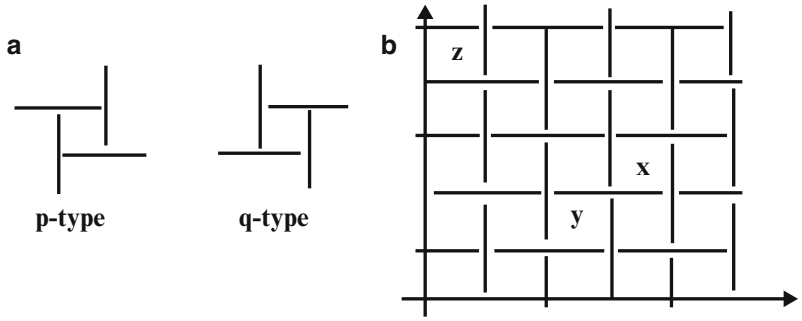
3. *Compaction:* Given a placement, its *bounding-box* is the minimum rectangle that encloses all the objects. A placement of objects is evaluated by the smallness of the bounding-box's area, abbreviated as the *bb-area*. An ABLR-relation set is also evaluated by the minimum bb-area of all the placements that satisfy the set. However, given a consistent ABLR-relation set, the corresponding placement is not unique in general. Still, the minimum bb-area is easily obtained by a common technique called the "Longest-Path Algorithm." (See, e.g., [4].)

   Consider the placement whose objects are all inside the 1st quadrant of the xy-coordinate system, without loss of generality with respect to minimizing the bb-area. It is evident that if a given ABLR-relation set is feasible, there is an object that has no object left or below it. Place it such that its left-bottom corner is at the origin. From the remaining objects, take one that has no object left of or below it. Place it as leftward and downward as long as any ABLR-relation with already fixed objects is not violated. See Fig. 1 to catch the concept, where the ABLR-relation set is the one obtained the placement in (a) (so that it is trivially feasible). It is possible to obtain different ABLR-relation sets, according to which compaction would produce different placements.
4. *Slice-line:* If it is possible to draw a straight horizontal line or vertical line to separate the objects into two groups, the line is said a *slice-line*. If each group again has a slice-line, and so does recursively, the placement is said to be a *slicing structure*. Figure 2 shows placements of slicing and non-slicing structures.
5. *Spiral:* Two structures each consisting of four line segments connected by a *T-junction* as shown in Fig. 3a are *spirals*. Their regular alignment in the first quadrant as shown in (b) is the *Bounded-Sliceline-Grid* or BSG. A BSG is a *floorplan*, or a *T-junction dissection*, of the rectangular area into rectangular regions called *rooms*. It is denoted as an $n \times m$ BSG if the numbers of rows and columns of its rooms are n and m, respectively. According to the left-bottom room being p-type or q-type, the BSG is said to be p-type or q-type, respectively.

In a BSG, take two rooms x and y. The ABLR-relations between them are all that is defined by



Floorplan and Placement, Fig. 2 (**a**) A placement with a slice-line. (**b**) A slicing structure since a slice-line can be found in each ith hierarchy No. $k(k = 1, 2, 3, 4)$. (**c**) A placement that has no slice-line



Floorplan and Placement, Fig. 3 (**a**) Two types of the spiral structure (**b**) 5 × 5 p-type bounded-slice line-grid (BSG)

the rule: If the bottom segment of x is the top segment of y (Fig. 3), room x is above room y. Furthermore, *Transitive Law* is assumed: If "x is above y" and "z is above x," then "z is above y."

Other relations are analogously defined.

Lemma 1 *A room is in a unique ABLR-relation with every other room.*

An $n \times n$ BSG has n^2 rooms. A BSG-assignment is a *one-to-one mapping* of n objects into the rooms of $n \times n$ BSG. ($n^2 - n$ rooms remain vacant.)

After a BSG-assignment, a pair of two objects inherits the same ABLR-relation as the ABLR-relation defined between corresponding rooms. In Fig. 3, if x, y, and z are the names of objects, the ABLR-relations among them are {(xAy), (xRz), (yBx), (yBz), (zLx), (zAy)}.

Key Results

The input is n objects that are rectangles of arbitrary sizes. The main concern is the *solution space*, the collection of distinct consistent ABLR-relation sets, to be generated by BSG or SP.

Theorem 1 ([4,5])

1. *For any feasible ABLR-relation set, there is a BSG-assignment into $n \times n$ BSG of any type that generates the same ABLR-relation set.*
2. *The size $n \times n$ is a minimum: if the number of rows or columns is less than n, there is a feasible ABLR-relation set that is not obtained by any BSG-assignment.*

The proof to (1) is not trivial [5] (Appendix). The number of solutions is ${}_{n^2}C_n$. A remarkable feature of an $n \times n$ BSG is that any ABLR-relation set of n objects is generated by a proper BSG-assignment. By this property, BSG is said to be *universal* [11].

In contrast to the BSG-based generation of consistent ABLR-relation sets, SP directly imposes the ABLR-relations on objects.

A pair of permutations of object names, represented as (Γ^+, Γ^-), is called the sequence-pair, or SP. See Fig. 1. An SP is decoded to a unique ABLR-relation set by the rule:

Consider a pair (x, y) of names such that x is before y in Γ^-. Then (xLy) or (xAy) if x is before or after y in Γ^+, respectively. ABLR-relations

"B" and "R" can be derived as the inverse of "A" and "L." Examples are given in Fig. 1.

A remarkable feature of sequence-pair is that its generation and decoding are both possible by simple operations. The question is what the solution space of all SPs is.

Theorem 2 *Any feasible placement has a corresponding SP that generates an ABLR-relation set satisfied by the placement. On the other hand, any SP has a corresponding placement that satisfies the ABLR-relation set derived from the SP.*

Using SP, a common compaction technique mentioned before is described in a very simple way:

Minimum Area Placement from $SP = (\Gamma^+, \Gamma^-)$

1. Relabel the objects such that $\Gamma^- = (1, 2, \ldots, n)$. Then $\Gamma^+ = (p_1, p_2, \ldots, p_n)$ will be a permutation of numbers $1, 2, \ldots, n$. It is simply a kind of normalization of SP [6]. But Kajitani [11] considers it a concept derived from Q-sequence [10] and studies its implication by the name of *single-sequence* or SS. In the example in Fig. 1b, p, q, r, and s are labeled as 1, 2, 3, and 4 so that SS = (2413).
2. Take object 1 and place it at the left-bottom corner in the 1st quadrant.
3. For $k = 2, 3, \ldots, n$, place k such that its left edge is at the rightmost edge of the objects with smaller numbers than k and lie before k in SS, and its bottom edge is at the topmost edge of the objects with smaller numbers than k and lie after k in SS.

Applications

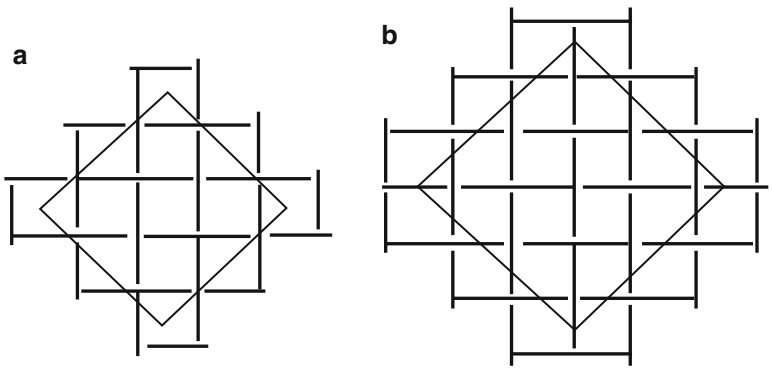
Many ideas followed after BSG and SP [2–5] as seen in the reference. They all applied a common methodology of a stochastic heuristic search, called simulated annealing, to generate feasible placements one after another based on some evaluation (with respect to the smallness of the bb-area) and to keep the best-so-far as the output. This methodology has become practical by the speed achieved due to their simple data structure. The first and naive implementation of BSG [2] could output the layout of sufficiently small area placement of 500 rectangles in several minutes. (Finding a placement with the minimum bb-area is NP-hard [3].) Since then many ideas followed, including currently widely used codes such as O-tree [7], B*-tree [8], corner block list [9], Q-sequence [10], single-sequence [11], and others. Their common feature is in coding the nonoverlapping constraint along horizontal and vertical directions, which is the inheritant property of rectangles.

As long as applications are concerned with the rectangle placement in the minimum area and do not mind mutual interconnection, the problem can be solved practically enough by BSG, SP, and those related ideas. However, in an integrated circuit layout problem, mutual connection is a major concern. Objects are not restricted to rectangles, even soft objects are used for performance. Many efforts have been devoted with a certain degree of success. For example, techniques concerned with rectilinear objects, rectilinear chip, insertion of small but numerous elements like buffers and decoupling capacitors, replacement for design change, symmetric placement for analog circuit design, three-dimensional placement, etc. have been developed. Here few of them is cited but it is recommended to look at proceedings of ICCAD (International Conference on Computer-Aided Design), DAC (Design Automation Conference), ASPDAC (Asia and South Pacific Design Automation Conference), DATE (Design Automation and Test in Europe), and journals TCAD (IEEE Trans. on Computer-Aided Design) and TCAS (IEEE Trans. on Circuit and Systems), particularly those that cover VLSI (Very Large Scale Integration) physical design.

Open Problems

BSG

The claim of Theorem 1 that a BSG needs n rows to provide any feasible ABLR-relation set is reasonable if considering a placement of all objects



Floorplan and Placement, Fig. 4 Octagonal BSG of size n, p-type: (**a**) If n is odd, it has $(n^2+1)/2$ rooms. (**b**) If n is even, it has $(n^2+2n)/2$ rooms

aligned vertically. This is due to the rectangular framework of a BSG. However, experiments have been suggesting a question if from the beginning [5] if we need such big BSGs. The octagonal BSG is defined in Fig. 4. It is believed to hold the following claim expecting a drastic reduction of the solution space.

Conjecture (BSG): For any feasible ABLR-relation set, there is an assignment of n objects into octagonal BSG of size n, any type, that generates the same ABLR-relation set.

If this is true, then the size of the solution space needed by a BSG reduces to ${}_{(n^2+1)/2}C_n$ or ${}_{(n^2+2n)/2}C_n$.

SP or SS

It is possible to define the universality of SP or SS in the same manner as defined for BSG. In general, two sequences of arbitrary k numbers $\mathrm{P} = (\mathrm{p}_1, \mathrm{p}_2, \ldots, \mathrm{p}_k)$ and $\mathrm{Q} = (\mathrm{q}_1, \mathrm{q}_2, \ldots, \mathrm{q}_k)$ are said *similar* with each other if $\mathrm{ord}(\mathrm{p}_i) = \mathrm{ord}(\mathrm{q}_i)$ for every i where $\mathrm{ord}(\mathrm{p}_i) = j$ implies that p_i is the jth smallest in the sequence. If they are single-sequences, two similar sequences generate the same set of ABLR-relations under the natural one-to-one correspondence between numbers.

An SS of length m (necessarily $\geq n$) is said *universal of order n* if SS has a subsequence (a sequence obtained from SS by deleting some of the numbers) that is similar to any sequence of length n. Since rooms of a BSG are considered n^2 objects, Theorem 1 implies that there is a universal SS of order n whose length is n^2. The known facts about smaller universal SS are:

1. For $n = 2, 132, 231, 213$, and 312 are the shortest universal SS. Note that 123 and 321 are not universal.
2. For $n = 3$, SS $=$ 41352 is the shortest universal SP.
3. For $n = 4$, the shortest length of universal SS 10 or less.
4. The size of universal SS is $\Omega(n^2)$ (Imahori S, Dec 2005, Private communication).

Open Problem (SP)

It is still an open problem to characterize the universal SP. For example, give a way to (1) certify a sequence as universal and (2) generate a minimum universal sequence for general n.

Cross-References

- Circuit Placement
- Slicing Floorplan Orientation

Recommended Readings

1. Wong DF, Liu CL (1985) A new algorithm for floorplan design. In: ACM/IEEE design automation conference (DAC), 23 Nov 1985, pp 101–107
2. Nakatake S, Murata H, Fujiyoshi K, Kajitani Y (1994) Bounded sliceline grid (BSG) for module packing. IEICE Technical report, Oct 1994, VLD94-66, vol 94, no 313, pp 19–24 (in Japanese)

3. Murata H, Fujiyoshi K, Nakatake S, Kajitani Y (1995) A solution space of size $(n!)^2$ for optimal rectangle packing. In: 8th Karuizawa workshop on circuits and systems, April 1995, pp 109–114
4. Murata H, Nakatake S, Fujiyoshi K, Kajitani Y (1996) VLSI module placement based on rectangle-packing by sequence-pair. IEEE Trans Comput Aided Des (TCAD) 15(12):1518–1524
5. Nakatake S, Fujiyoshi K, Murata H, Kajitani Y (1998) Module packing based on the BSG-structure and IC layout applications. IEEE Trans Comput Aided Des (TCAD) 17(6):519–530
6. Kodama C, Fujiyoshi K (2003) Selected sequence-pair: an efficient decodable packing representation in linear time using sequence-pair. In: Proceedings of Asia and South Pacific design automation conference (ASP-DAC), Bangalore, 2003, pp 331–337
7. Guo PN, Cheng CK, Yoshimura T (1998) An O-tree representation of non-slicing floorplan and its applications. In: 36th design automation conference (DAC), June 1998, pp 268–273
8. Chang Y-C, Chang Y-W, Wu G-M, Wu S-W (2000) B*-trees: a new representation for non-slicing floorplans. In: 37th design automation conference (DAC), June 2000, pp 458–463
9. Hong X, Dong S, Ma Y, Cai Y, Cheng CK, Gu J (2000) Corner block list: an efficient topological representation of non-slicing floorplan. In: International computer aided design (ICCAD) '00, San Jose, Nov 2000, pp 8–12
10. Sakanushi K, Kajitani Y, Mehta D (2003) The quarter-state-sequence floorplan representation. IEEE TCAS-I 50(3):376–386
11. Kajitani Y (2006) Theory of placement by single-sequence related with DAG, SP, BSG, and O-tree. In: International symposium on circuits and systems, May 2006

Flow Time Minimization

Luca Becchetti[1], Stefano Leonardi[1], Alberto Marchetti-Spaccamela[1], and Kirk Pruhs[2]
[1]Department of Information and Computer Systems, University of Rome, Rome, Italy
[2]Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

Keywords

Flow time: response time

Years and Authors of Summarized Original Work

2001; Becchetti, Leonardi, Marchetti-Spaccamela, Pruhs

Problem Definition

Shortest-job-first heuristics arise in sequencing problems, when the goal is minimizing the perceived latency of users of a multiuser or multitasking system. In this problem, the algorithm has to schedule a set of jobs on a pool of m identical machines. Each job has a release date and a processing time, and the goal is to minimize the average time spent by jobs in the system. This is normally considered a suitable measure of the quality of service provided by a system to interactive users. This optimization problem can be more formally described as follows:

Input

A set of m identical machines and a set of n jobs $1, 2, \ldots, n$. Every job j has a release date r_j and a processing time p_j. In the sequel, $\mathcal{I}$ denotes the set of feasible input instances.

Goal

The goal is minimizing the *average flow* (also known as *average response*) time of the jobs. Let C_j denote the time at which job j is completed by the system. The flow time or response time F_j of job j is defined by $F_j = C_j - r_j$. The goal is thus minimizing

$$\min \frac{1}{n} \sum_{j=1}^{n} F_j \ .$$

Since n is part of the input, this is equivalent to minimizing the *total* flow time, i.e., $\sum_{j=1}^{n} F_j$.

Off-line versus On-line

In the *off-line setting*, the algorithm has full knowledge of the input instance. In particular, for every $j = 1, \ldots, n$, the algorithm knows r_j and p_j.

Conversely, in the *on-line setting*, at any time t, the algorithm is only aware of the set of jobs released up to time t.

In the sequel, A and OPT denote, respectively, the algorithm under consideration and the optimal, off-line policy for the problem. $A(I)$ and $OPT(I)$ denote the respective costs on a specific input instance I.

Further Assumptions in the On-line Case

Further assumptions can be made as to the algorithm's knowledge of processing times of jobs. In particular, in this survey an important case is considered, realistic in many applications, i.e., that p_j is completely unknown to the on-line algorithms until the job eventually completes (*non-clairvoyance*) [1, 3].

Performance Metric

In all cases, as is common in combinatorial optimization, the performance of the algorithm is measured with respect to its optimal, off-line counterpart. In a minimization problem such as those considered in this survey, the competitive ratio ρ_A is defined as:

$$\rho_A = \max_{I \in \mathcal{I}} \frac{A(I)}{OPT(I)} .$$

In the off-line case, ρ_A is the *approximation ratio* of the algorithm. In the on-line setting, ρ_A is known as the *competitive ratio* of A.

Preemption

When *preemption* is allowed, a job that is being processed may be interrupted and resumed later after processing other jobs in the interim. As shown further, preemption is necessary to design efficient algorithms in the framework considered in this survey [5, 6].

Key Results

Algorithms

Consider any job j in the instance and a time t in A's schedule, and denote by $w_j(t)$ the amount of time spent by A on job j until t. Denote by $x_j(t) = p_j - w_j(t)$ its *remaining processing time* at t.

The best known heuristic for minimizing the average flow time when preemption is allowed is *shortest remaining processing time* (SRPT). At any time t, SRPT executes a pending job j such that $x_j(t)$ is minimum. When preemption is not allowed, this heuristic translates to *shortest job first* (SJF): at the beginning of the schedule, or when a job completes, the algorithm chooses a pending job with the shortest processing time and runs it to completion.

Complexity

The problem under consideration is polynomially solvable on a single machine when preemption is allowed [9, 10]. When preemption is allowed, SRPT is optimal for the single-machine case. On parallel machines, the best known upper bound for the preemptive case is achieved by SRPT, which was proven to be $O(\log \min n/m, P)$-approximate [6], P being the ratio between the largest and smallest processing times of the instance. Notice that SRPT is an on-line algorithm, so the previous result holds for the on-line case as well. The authors of [6] also prove that this lower bound is tight in the on-line case. In the off-line case, no non-constant lower bound is known when preemption is allowed.

In the non-preemptive case, no off-line algorithm can be better than $\Omega(n^{1/3-\epsilon})$-approximate, for every $\epsilon > 0$, the best upper bound being $O(\sqrt{n/m}\log(n/m))$ [6]. The upper and lower bound become $O(\sqrt{n})$ and $\Omega(n^{1/2-\epsilon})$ for the single machine case [5].

Extensions

Many extensions have been proposed to the scenarios described above, in particular for the preemptive, on-line case. Most proposals concern the power of the algorithm or the knowledge of the input instance. For the former aspect, one interesting case is the one in which the algorithm is equipped with faster machines than its optimal counterpart. This aspect has been considered in [4]. There the authors prove that even a moderate increase in speed makes some very

simple heuristics have performances that can be very close to the optimum.

As to the algorithm's knowledge of the input instance, an interesting case in the on-line setting, consistent with many real applications, is the non-clairvoyant case described above. This aspect has been considered in [1, 3]. In particular, the authors of [1] proved that a randomized variant of the MLF heuristic described above achieves a competitive ratio that in the average is at most a polylogarithmic factor away from the optimum.

Applications

The first and traditional field of application for scheduling policies is resource assignment to processes in multitasking operating systems [11]. In particular, the use of shortest-job-like heuristics, notably the MLF heuristic, is documented in operating systems of wide use, such as UNIX and WINDOWS NT [8, 11]. Their application to other domains, such as access to Web resources, has been considered more recently [2].

Open Problems

Shortest-job-first-based heuristics such as those considered in this survey have been studied in depth in the recent past. Still, some questions remain open. One concerns the off-line, parallel-machine case, where no non-constant lower bound on the approximation is known yet. As to the on-line case, there still is no tight lower bound for the non-clairvoyant case on parallel machines. The current $\Omega(\log n)$ lower bound was achieved for the single-machine case [7], and there are reasons to believe that it is below the one for the parallel case by a logarithmic factor.

Cross-References

- ▸ Minimum Flow Time
- ▸ Minimum Weighted Completion Time
- ▸ Multilevel Feedback Queues
- ▸ Shortest Elapsed Time First Scheduling

Recommended Reading

1. Becchetti L, Leonardi S (2004) Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. J ACM 51(4):517–539
2. Crovella ME, Frangioso R, Harchal-Balter M (1999) Connection scheduling in web servers. In: Proceedings of the 2nd USENIX symposium on Internet technologies and systems (USITS-99), pp 243–254
3. Kalyanasundaram B, Pruhs K (2003) Minimizing flow time nonclairvoyantly. J ACM 50(4):551–567
4. Kalyanasundaram B, Pruhs K (2000) Speed is as powerful as clairvoyance. J ACM 47(4):617–643
5. Kellerer H, Tautenhahn T, Woeginger GJ (1996) Approximability and nonapproximability results for minimizing total flow time on a singlemachine. In: Proceedings of 28th annual ACM symposium on the theory of computing (STOC'96), pp 418–426
6. Leonardi S, Raz D (1997) Approximating total flow time on parallel machines. In: Proceedings of the annual ACM symposium on the theory of computing (STOC), pp 110–119
7. Motwani R, Phillips S, Torng E (1994) Nonclairvoyant scheduling. Theor Comput Sci 130(1):17–47
8. Nutt G (1999) Operating system projects using Windows NT. Addison-Wesley, Reading
9. Schrage L (1968) A proof of the optimality of the shortest remaining processing time discipline. Oper Res 16(1):687–690
10. Smith DR (1976) A new proof of the optimality of the shortest remaining processing time discipline. Oper Res 26(1):197–199
11. Tanenbaum AS (1992) Modern operating systems. Prentice-Hall, Englewood Cliffs

Force-Directed Graph Drawing

Ulrik Brandes
Department of Computer and Information Science, University of Konstanz, Konstanz, Germany

Keywords

Force-directed placement; Graph drawing; MDS; Spring embedder

Years and Authors of Summarized Original Work

1963; Tutte
1984; Eades

Problem Definition

Given a connected undirected graph, the problem is to determine a straight-line layout such that the structure of the graph is represented in a readable and unbiased way. Part of the problem is the definition of readable and unbiased.

Formally, we are given a simple, undirected graph $G = (V, E)$ with vertex set V and edge set $E \subseteq \binom{V}{2}$. Let $n = |V|$ be the number of vertices and $m = |E|$ the number of edges. The *neighbors* of a vertex v are defined as $N(v) = \{u : \{u, v\} \in E\}$, and $\deg(v) = |N(v)|$ is its *degree*. We assume that G is connected, for otherwise the connected components can be treated separately.

A (two-dimensional) layout for G is a vector $p = (p_v)_{v \in V}$ of vertex positions $p_v = \langle x_y, y_v \rangle \in \mathbb{R}^2$. Since edges are drawn as line segments, the drawing is completely determined by these vertex positions. All approaches in this chapter generalize to higher-dimensional layouts, and there are variants for various graph classes and desired layout features; we only discuss some of them briefly at the end.

The main idea is to make use of physical analogies. A graph is likened to a system of objects (the vertices) that are subject to varying forces (derived from structural features). Forces cause the objects to move around until those pushing and pulling into different directions cancel each other out and the graph layout reaches an equilibrium state. Equivalently, states might be described by an energy function so that forces are not specified directly, but derived from gradients of the energy function.

As a reference model, consider the layout energy function

$$A(p) = \sum_{\{u,v\} \in E} \|p_u - p_v\|^2 \qquad (1)$$

where $\|p_u - p_v\|^2 = (x_u - x_v)^2 + (y_u - y_v)^2$ is the squared Euclidean distance of the endpoints of edge $\{u, v\}$. It associates with a layout the sum of squared edge lengths, so that its minimization marks an attempt to position adjacent vertices close to each other. Because of its straightforward physical analogy, we refer to the minimization of Eq. (1) as the *attraction model*.

Note that minimum-energy layouts of this pure attraction model are degenerate in that all vertices are placed in the same position, since such layouts p are exactly those for which $A(p) = 0$ for a connected graph.

Even if it has not been the starting point of any of the approaches sketched in the next section, it is instructive to think of them as different solutions to the degeneracy problem inherent in the attraction model.

Key Results

We present force-directed layout methods as variations on the attraction model. The first two variants retain the objective but introduce constraints, whereas the other two modify the objective (Fig. 1).

For the constraint-based variants, it is more convenient to analyze the attraction model in matrix form. A necessary condition for a (local) minimum of any objective function is that all partial derivatives vanish. For the attraction model (1), this amounts to

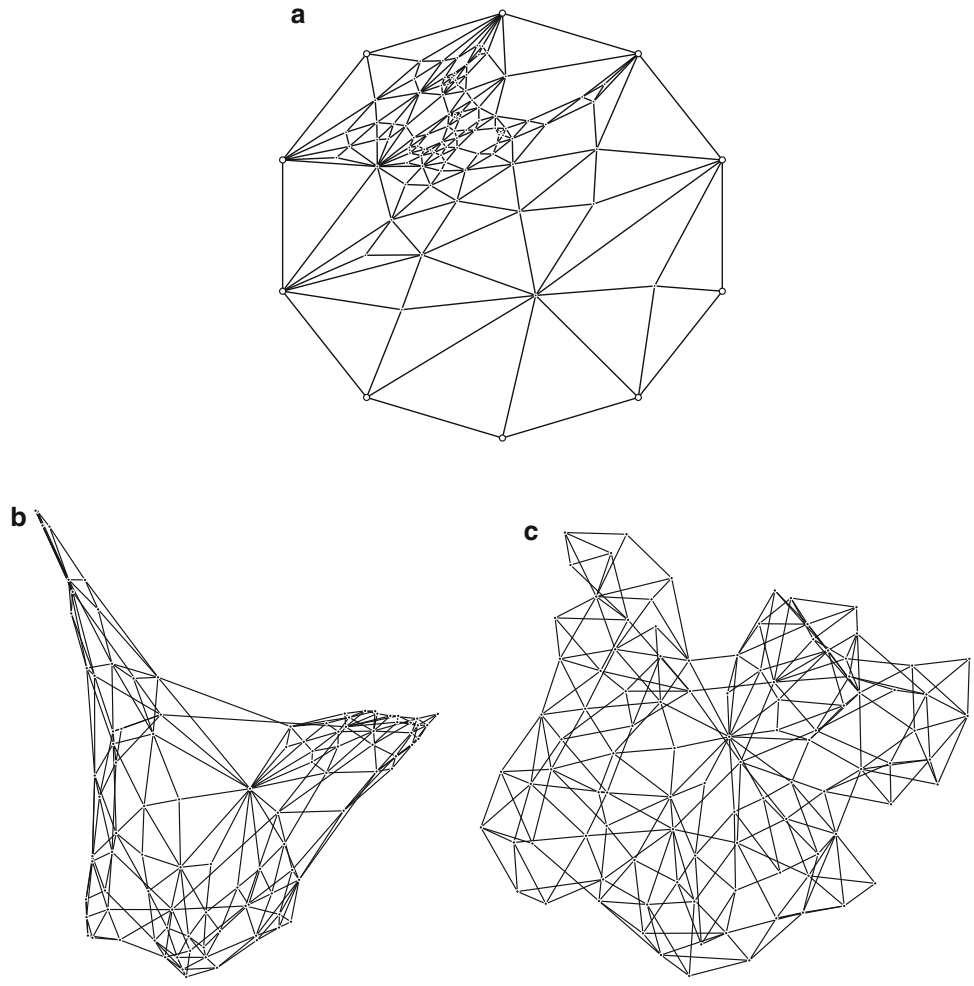
$$\frac{\partial}{\partial x_v} A(p) = \frac{\partial}{\partial y_v} A(p) = 0 \quad \text{for all } v \in V.$$

For any $v \in V$,

$$\frac{\partial}{\partial x_v} A(p) = \sum_{u \in N(v)} 2(x_v - x_u) \stackrel{!}{=} 0,$$

and likewise for $\frac{\partial}{\partial y_v} A(p)$. The necessary conditions can therefore be translated into

$$x_v = \frac{\sum_{u \in N(v)} x_u}{\deg(v)} \quad \text{and} \quad y_v = \frac{\sum_{u \in N(v)} y_u}{\deg(v)}$$



Force-Directed Graph Drawing, Fig. 1 Three different layouts of the same planar triconnected graph. (**a**) Barycentric. (**b**) Spectral. (**c**) Stress

for all $v \in V$, i.e., every vertex must lie in the barycenter of its neighbors. Bringing all variables to the left-hand side, we obtain a system of linear equations whose coefficients form an eminent graph-related matrix, the *Laplacian matrix* $L(G) = D(G) - A(G)$, where $D(G)$ is a diagonal matrix with diagonal entries $\deg(v)$, $v \in V$, and $A(G)$ is the adjacency matrix of G. The entries of $L = L(G) = (\ell_{uv})_{u,v \in V}$ are thus

$$\ell_{uv} = \begin{cases} \deg(v) & \text{if } u = v \\ -1 & \text{if } u \neq v \text{ and } \{u, v\} \in E \\ 0 & \text{otherwise.} \end{cases}$$

so that the optimality conditions can be written as

$$L \cdot p = \mathbf{0}, \tag{2}$$

where $\mathbf{0}$ is an $n \times 2$-matrix of zeros. As discussed above, the solutions to this system of linear equations are given by those layouts p in which all x- and all y-coordinates are identical.

Fixed Boundary

An intuitive approach to prevent attraction from collapsing an entire graph onto a single point is to grab a few of its vertices and drag them apart. Technically, this corresponds to constraining

the layout by fixing select vertices to distinct positions.

Let $B \subseteq V$ be a nonempty subset of *boundary vertices* for which positions $\hat{p}_v = \langle \hat{x}_v, \hat{y}_v \rangle$, $v \in B$ are pre-specified. A layout is called *barycentric* (with respect to these constraints) if it satisfies

$$p_v = \begin{cases} \hat{p}_v & \text{if } v \in B \\ \frac{1}{\deg(v)} \sum_{u \in N(v)} p_u & \text{otherwise.} \end{cases}$$

We next show that the solution of the attraction model with a proper boundary constraint is unique by showing that the reduced system of linear equations has a coefficient matrix with a nonzero determinant. Let L^B denote the matrix obtained by striking out the rows and columns of L indexed by B. Then, a barycentric layout is a solution of

$$L^B \cdot p_{V \setminus B} = \left(\sum_{u \in N(v) \cap B} \hat{p}_u \right)_{V \setminus B} . \quad (3)$$

Different from the pure attraction model, the barycentric model (with a nondegenerate boundary) has a nondegenerate solution that is uniquely defined. Recall that a system of linear equations has a unique solution if and only if the determinant of its matrix of coefficients is positive. The Matrix Tree Theorem [9] asserts that the determinant of every principal minor of a Laplacian matrix equals the number of spanning trees of its associated multigraph, and L^B is a principal minor of the Laplacian of the graph obtained from G by contracting the vertices in B. Since this graph has at least one spanning tree, the determinant of L^B is positive and the solution of (3) is thus unique.

The barycentric approach was introduced in Tutte [15]. The main result shown in this paper is, in fact, that barycentric layouts of a triconnected planar graph with one face constrained to a convex polygon are planar. For the purpose of graph drawing, less desirable properties are exponentially small resolution of angles and edge lengths as evidenced by a family of triangular graphs obtained by starting from a triangle and adding vertices adjacent to the same two initial vertices and the most recently added one. For nontriconnected graphs, degenerate subgraph layouts are possible because components lacking boundary vertices are mapped to a line if between a separation pair, or to a point if hinging on a cut vertex.

Orthogonality

Barycentric layouts are systematically biased by the choice of boundary. An alternative constraint avoiding the single-point collapse is to constrain the coordinate vector of each dimension to be orthogonal to the degenerate layout.

Observe that the one-dimensional version of Eq. (2) can also be read as a special case of the eigenequation $Lx = \lambda x$, since $\lambda = 0$ is, in fact, an eigenvalue of L associated with eigenvector $x = \mathbf{1}$. The Laplacian of a simple undirected graph is a real, symmetric, and positive semidefinite matrix so that the eigenvalues are real and nonnegative, and eigenvectors associated with different eigenvalues are orthogonal.

Rearranging the eigenequation yields $\lambda = \frac{x^T L x}{x^T x}$, where $x^T x$ only normalizes for scale. Since $x^T L x = A(x)$, eigenvectors $x \perp y$ associated with the smallest positive eigenvalues yield the best layout in the attraction model subject to orthogonality also with the degenerate layout $\mathbf{1}$. Note that $\mathbf{1} \perp x$ implies that the average of all coordinates x_v is zero, so that the layouts are centered on the origin.

Spectral drawings based on the Laplacian have been proposed by Hall [7], but can be also be defined via other matrices [11]. It is interesting to note that Laplacian spectral layout corresponds to classical multidimensional scaling using the square root of effective resistance as a measure of distance between vertices. While spectral layouts display symmetries, they are highly cluttered and imbalanced for graphs of low algebraic connectivity as measured by their smallest positive eigenvalue.

Distances

The terms in the objective function of the attraction model correspond to the potential energy of a spring with ideal length zero. To avoid

collapse, one can thus replace them by springs of some nonzero ideal length. While this takes care of the adjacent pairs of vertices, vertices that are more than one edge away from each other can be connected by springs of different length, say proportional to their shortest-path distance. Down-weighting the influence of distant pairs, we obtain the *stress-minimization model* with objective

$$S(p) = \sum_{u,v \in V} \frac{1}{d(u,v)^2} \left(\|p_u - p_v\| - d(u,v) \right)^2 ,$$

constituting another special case of MDS [12] with graph-theoretic distances $d(u,v)$ as input and inverse quadratic weights. This instantiation has been proposed as a graph drawing method by Kamada and Kawai [8] using gradient descent to determine locally optimal layouts. The use of majorization [13] was shown to be superior by Gansner, Koren, and North [6]. A comprehensive survey of variant layout objective functions is given in Chen and Buja [3].

Repulsion

Instead of springs with nonzero ideal length, a dual physical analogy motivates another approach to counter the collapse caused by attraction, namely, repulsion.

The classic *spring embedder* of Eades [4] specifies forces rather than an energy function. While there is a logarithmic force $\log \frac{\|p_u - p_v\|}{l} \cdot (p_u - p_v)$ between adjacent vertices that is neutral if their distance equals a desired value l, nonadjacent vertices push each other apart with quadratically with $\frac{(p_u - p_v)}{\|p_u - p_v\|}$. Both forces are up to scaling constants. A layout is obtained by iteratively evaluating the forces exerted on a vertex by all others and then moving it in the direction of the resulting force, until an approximate equilibrium is obtained.

Many, many variants of the spring embedder have been proposed. The most widely used from Fruchterman and Reingold [5] replaces the forces by quadratically declining repulsion between all pairs of vertices and additional quadratic attraction between adjacent pairs and also introduces several pragmatic improvements. Brandes and Pich [2] find that suitably initialized stress MDS yields superior results, though.

More force-directed methods are surveyed in Brandes [1] and Kobourov [10], and forces have been used very creatively to realize different layout objectives such as common direction of edges, edge curvatures, angles between incident edges, preferred locations, and many more. A relation with graph clustering is pointed out in Noack [14].

Recommended Reading

1. Brandes U (2001) Drawing on physical analogies. In: Kaufmann M, Wagner D (eds) Drawing graphs: methods and models. Lecture notes in computer science, vol 2025. Springer, Berlin/Heidelberg, pp 71–86
2. Brandes U, Pich C (2009) An experimental study on distance-based graph drawing. In: Proceedings of the 16th international symposium on graph drawing (GD'08), Heraklion. Lecture notes in computer science, vol 5417. Springer, pp 218–229
3. Chen L, Buja A (2013) Stress functions for nonlinear dimension reduction, proximity analysis, and graph drawing. J Mach Learn Res 14:1145–1173
4. Eades P (1984) A heuristic for graph drawing. Congr Numerantium 42:149–160
5. Fruchterman TMJ, Reingold EM (1991) Graph drawing by force-directed placement. Softw Pract Exp 21(11):1129–1164
6. Gansner ER, Koren Y, North SC (2005) Graph drawing by stress majorization. In: Proceedings of the 12th international symposium on graph drawing (GD'04), New York. Lecture notes in computer science, vol 3383. Springer, New York, pp 239–250. doi: 10.1007/978-3-540-31843-9_25
7. Hall KM (1970) An r-dimensional quadratic placement algorithm. Manag Sci 17(3):219–229
8. Kamada T, Kawai S (1989) An algorithm for drawing general undirected graphs. Inf Process Lett 31:7–15
9. Kirchhoff GR (1847) Über die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Verteilung galvanischer Ströme geführt wird. Ann Phys Chem 72:497–508
10. Kobourov SG (2013) Force-directed drawing algorithms. In: Tamassia R (ed) Handbook of graph drawing and visualization. CRC, Boca Raton, pp 383–408
11. Koren Y (2005) Drawing graphs by eigenvectors: theory and practice. Comput Math Appl 49(11–12):1867–1888. doi:10.1016/j.camwa.2004.08.015
12. Kruskal JB (1964) Multidimensional scaling for optimizing goodness of fit to a nonmetric hypothesis. Psychometrika 29(1):1–27

13. de Leeuw J (1977) Applications of convex analysis to multidimensional scaling. In: Barra JR, Brodeau F, Romier G, van Cutsem B (eds) Recent developments in statistics. North-Holland, Amsterdam, pp 133–145
14. Noack A (2009) Modularity clustering is force-directed layout. Phys Rev E 79:026,102
15. Tutte WT (1963) How to draw a graph. Proc Lond Math Soc 13(3):743–768

FPGA Technology Mapping

Jason Cong[1] and Yuzheng Ding[2]
[1]Department of Computer Science, UCLA, Los Angeles, CA, USA
[2]Xilinx Inc., Longmont, CO, USA

Keywords

FlowMap; Lookup-table mapping; LUT mapping

Years and Authors of Summarized Original Work

1992; Cong, Ding

Problem Definition

Introduction

Field-programmable gate array (FPGA) is a type of integrated circuit (IC) device that can be (re)programmed to implement custom logic functions. A majority of FPGA devices use lookup table (LUT) as the basic logic element, where a LUT of K logic inputs (K-LUT) can implement any Boolean function of up to K variables. An FPGA also contains other logic elements, such as registers, programmable interconnect resources, dedicated logic resources such as memory blocks and digital signal processing (DSP) blocks, and input/output resources [6].

The programming of an FPGA involves the transformation of a logic design into a form suitable for implementation on the target FPGA device. This generally takes multiple steps. For LUT-based FPGAs, *technology mapping* is to transform a general Boolean logic network (obtained from the design specification through earlier transformations) into a functionally equivalent K-LUT network that can be implemented by the target FPGA device. The objective of a technology mapping algorithm is to generate, among many possible solutions, an optimized one according to certain criteria, some of which are timing optimization, which is to make the resultant implementation operable at faster speed; area minimization, which is to make the resultant implementation compact in size; and power minimization, which is to make the resultant implementation low in power consumption. The algorithm presented here, named *FlowMap* [2], is for timing optimization; it was the first provably optimal polynomial time algorithm for technology mapping problems on general Boolean networks, and the concepts and approach it introduced have since generated numerous useful derivations and applications.

Data Representation and Preliminaries

The input data to a technology mapping algorithm for LUT-based FPGA is a *general Boolean network*, which can be modeled as a direct acyclic graph $N = (V, E)$. A node $v \in V$ can either represent a logic signal source from outside of the network, in which case it has no incoming edge and is called a *primary input* (PI) node, or it can represent a *logic gate*, in which case it has incoming edge(s) from PIs and/or other gates, which are its logic input(s). If the logic output of the gate is also used outside of the network, its node is a *primary output* (PO), which can have no outgoing edge if it is only used outside.

If edge $\langle u, v\rangle \in E, u$ is said to be a *fanin* of v and v a *fanout* of u. For a node $v, input(v)$ denotes the set of its fanins; similarly, for a subgraph $H, input(H)$ denotes the set of distinct nodes outside of H that are fanins of nodes in H. If there is a direct path in N from a node u to a node v, u is said to be a *predecessor* of v and v a *successor* of u. The *input network* of a node v, denoted N_v, is the subgraph containing v and all of its predecessors. A *cone* of a non-PI node

v, denoted C_v, is a subgraph of N_v containing v and possibly some of its *non-PI* predecessors, such that for any node $u \in C_v$, there is a path from u to v in C_v. If $|input(C_v)| \leq K, C_v$ is called a *K-feasible* cone. The network N is *K-bounded* if every non-PI node has a K-feasible cone. A *cut* of a non-PI node v is a bipartition (X, X') of nodes in N_v such that X' is a cone of v; $input(X')$ is called the *cut-set* of (X, X') and $n(X, X') = |input(X')|$ the *size* of the cut. If $n(X, X') \leq K$, (X, X') is a *K-feasible* cut. The *volume* of (X, X') is $vol(X, X') = |X'|$.

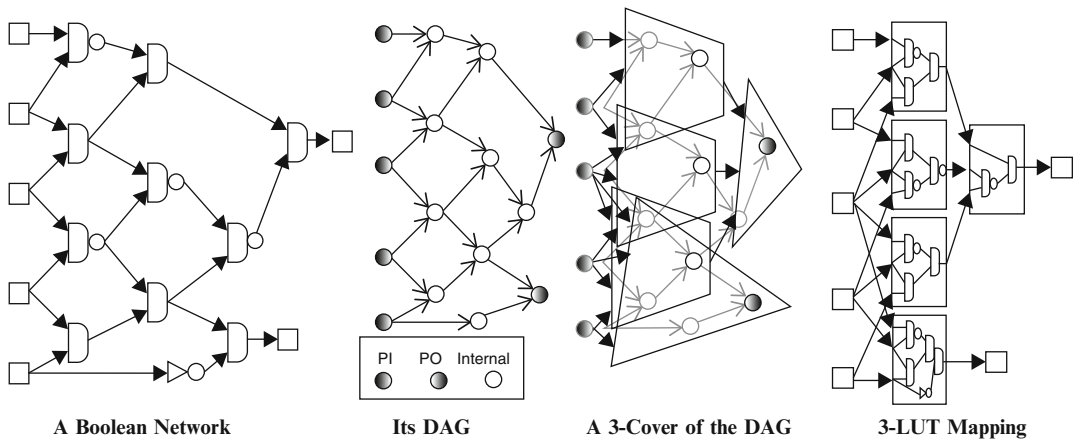
A *topological order* of the nodes in the network N is a linear ordering of the nodes in which each node appears after all of its predecessors and before any of its successors. Such an order is always possible for an acyclic graph.

Problem Formulation

A *K-cover* of a given Boolean network N is a network $N_M = (V_M, E_M)$, where V_M consists of the PI nodes of N and some K-feasible cones of nodes in N, such that for each PO node v of N, V_M contains a cone C_v of v; and if $C_u \in V_M$, then for each non-PI node $v \in input(C_u)$, V_M also contains a cone C_v of v. Edge $\langle u, C_v \rangle \in E_M$ if and only if PI node $u \in input(C_v)$; edge $\langle C_u, C_v \rangle \in E_M$ if and only if non-PI node $u \in input(C_v)$. Since each K-feasible cone can be implemented by a K-LUT, a K-cover can be implemented by a network of K-LUTs. Therefore, the *technology mapping* problem for K-LUT based FPGA, which is to transform N into a network of K-LUTs, is to find a K-cover N_M of N.

The *depth* of a network is the number of edges in its longest path. A technology mapping solution N_M is *depth optimal* if among all possible mapping solutions of N it has the minimum depth. If each level of K-LUT logic is assumed to contribute a constant amount of logic delay (known as the *unit delay* model), the minimum depth corresponds to the smallest logic propagation delay through the mapping solution, or in other words, the fastest K-LUT implementation of the network N. The problem solved by the *FlowMap* algorithm is *depth-optimal technology mapping for K-LUT based FPGAs*.

A Boolean network that is not K-bounded may not have a mapping solution as defined above. To make a network K-bounded, *gate decomposition* may be used to break larger gates into smaller ones. The *FlowMap* algorithm applies, as preprocessing, an algorithm named *DMIG* [3] that converts all gates into 2-input ones in a depth-optimal fashion, thus making the network K-bounded for $K \geq 2$. Different decomposition schemes may result in different K-bounded networks and consequently different mapping solutions; the optimality of *FlowMap* is with respect to a given K-bounded network Fig. 1 illustrates a Boolean network, its DAG, a covering with 3-feasible cones, and the resultant 3-LUT network. As illustrated, the cones in



FPGA Technology Mapping, Fig. 1 A Boolean network, its DAG. a 3-feasible cone covering, and a 3-LUT mapping

the covering may overlap; this is allowed and often beneficial. (When the mapped network is implemented, the overlapped portion of logic will be replicated into each of the K-LUTs that contain it)

Key Results

The *FlowMap* algorithm takes a two-phase approach. In the first phase, it determines for each non-PI node a preferred K-feasible cone as a candidate for the covering; the cones are computed such that if used, they will yield a depth-optimal mapping solution. This is the central piece of the algorithm. In the second phase, the cones necessary to form a cover are chosen to generate a mapping solution.

Structure of Depth-Optimal *K*-Covers

Let $M(v)$ denote a K-cover (or equivalently, K-LUT mapping solution) of the input network N_v of v. If v is a PI, $M(v)$ consists of v itself. (For simplicity, in the rest of the article, $M(v)$ shall be referred as *a K-cover of v*.) With that defined, first there is

Lemma 1 *If C_v is the K-feasible cone of v in a K-cover $M(v)$, then $M(v) = \{C_v\} + \cup\{M(u) : u \in input(C_v)\}$ where $M(u)$ is a certain K-cover of u. Conversely, if C_v is a K-feasible cone of v, and for each $u \in input(C_v)$, $M(u)$ a K-cover of u, then $M(v) = \{C_v\} + \cup\{M(u) : u \in input(C_v)\}$ is a K-cover of v.*

In other words, a K-cover of a node consists of a K-feasible cone of the node and a K-cover of each input of the cone. Note that for $u_1 \in input(C_v), u_2 \in input(C_v), M(u_1)$ and $M(u_2)$ may overlap, and an overlapped portion may or may not be covered the same way; the union above includes all *distinct* cones from all parts. Also note that for a given C_v, there can be different K-covers of v containing C_v, varying by the choice of $M(u)$ for each $u \in input(C_v)$.

Let $d(M(v))$ denote the depth of $M(v)$. Then

Lemma 2 *For K-cover $M(v) = \{C_v\} + \cup\{M(u) : u \in input(C_v)\}$,*

$$d(M(v)) = max\{d(M(u)) : u \in input(C_v)\} + 1.$$

In particular, let $M^*(u)$ denote a K-cover of u with minimum depth, then $d(M(v)) \geq max\{d(M^*(u)) : u \in input(C_v)\} + 1$; the equality holds when every $M(u)$ in $M(v)$ is of minimum depth.

Recall that C_v defines a K-feasible cut (X, X') where $X' = C_v, X = N_v - C_v$. Let $H(X, X')$ denote the *height* of the cut (X, X'), defined as $H(X, X') = max\{d(M^*(u)) : u \in input(X')\} + 1$. Clearly, $H(X, X')$ gives the minimum depth of any K-cover of v containing $C_v = X'$. Moreover, by properly choosing the cut, $H(X, X')$ height can be minimized, which leads to a K-cover with minimum depth:

Theorem 1 *If K feasible cut (X, X') of v has the minimum height among all K-feasible cuts of v, then the K-cover $M^*(v) = \{X'\} + \cup\{M^*(u) : u \in input(X')\}$ is of minimum depth among all K-covers of v.*

That is, a minimum height K-feasible cut defines a minimum depth K-cover. So the central task for depth-optimal technology mapping becomes the computation of a minimum height K-feasible cut for each PO node.

By definition, the height of a cut depends on the (depths of) minimum depth K-covers of nodes in $N_v - \{v\}$. This suggests a *dynamic programming* procedure that follows topological order, so that when the minimum depth K-cover of v is to be determined, a minimum depth K-cover of each node in $N_v - \{v\}$ is already known and the height of a cut can be readily determined. This is how the first phase of the *FlowMap* algorithm is carried out.

Minimum Height *K*-Feasible Cut Computation

The first phase of *FlowMap* was originally called the *labeling phase,* as it involves the computation of a *label* for each node in the K-bounded graph. The label of a non-PI node v, denoted $l(v)$, is defined as the minimum height of any cut of v. For convenience, the labels of PI nodes are defined to be 0.

The so-defined label has an important *monotonic* property.

Lemma 3 *Let* $p = max\{l(u) : u \in input(v)\}$, *then* $p \leq l(v) \leq p + 1$.

Note that this also implies that for any node $u \in N_v - \{v\}, l(u) \leq p$. Based on this, in order to find a minimum height K-feasible cut, it is sufficient to check if there is one of height p; if not, then any K-feasible cut will be of minimum height $(p + 1)$, and one always exists for a K-bounded graph.

The search for a K-feasible cut of a height p ($p > 0$; $p = 0$ is trivial) in *FlowMap* is done by transforming N_v into a *flow network* F_v and computing a *network flow* [5] on it (hence the name). The transformation is as follows. For each node $u \in N_v - \{v\}, l(u) < p, F_v$ has two nodes u_1 and u_2, linked by a *bridge* edge $\langle u_1, u_2\rangle$; F_v has a single *sink* node t for all other nodes in N_v, and a single *source* node s. For each PI node u of N_v, which corresponds to a bridge edge $\langle u_1, u_2\rangle$inF_v, F_v contains edge $\langle s, u_1\rangle$; for each edge $\langle u, w\rangle$ in N_v, if both u and w have bridge edges in F_v, then F_v contains edge $\langle u_2, w_1\rangle$; if u has a bridge edge but w does not, F_v contains edge $\langle u_2, t\rangle$; otherwise (neither has bridge) no corresponding edge is in F_v. The bridging edges have unit capacity; all others have infinite capacity. Noting that each edge in F_v with finite (unit) capacity corresponds to a node $u \in N_v$ with $l(u) < p$ and vice versa, and according to the max-flow min-cut theorem [5], it can be shown that.

Lemma 4 *Node v has a K-feasible cut of height p if and only if F_v has a maximum network flow of size no more than K.*

On the flow network F_v, a maximum flow can be computed by running the augmenting path algorithm [5]. Once a maximum flow is obtained, the *residual graph* of the flow network is disconnected, and the corresponding *min-cut* (X, X') can be identified as follows: $v \in X'$; for $u \in N_v - \{v\}$, if it is bridged in F_v, and u_1 can be reached in a depth-first search of the residual graph from s, then $u \in X$; otherwise $u \in X'$.

Note that as soon as the flow size exceeds K, the computation can stop, knowing there will not be a desired K-feasible cut. In this case, one can modify the flow network by bridging all nodes in $N_v - \{v\}$ allowing the inclusion of nodes u with $l(u) = p$ in the cut computation, and find a K-feasible cut with height $p + 1$ the same way.

An augmenting path is found in linear time to the number of edges, and there are at most K augmentations for each cut computation. Applying the algorithm to every node in topological order, one would have the following result.

Theorem 2 *In a K-bounded Boolean network of n nodes and m edges, the computation of a minimum height K-feasible cut for every node can be completed in $O(Kmn)$ time.*

The cut found by the algorithm has another property:

Lemma 5 *The cut (X, X') computed as above is the unique maximum volume min-cut; moreover, if (Y, Y') is another min-cut, then $Y' \subseteq X'$.*

Intuitively, a cut of larger volume defines a larger cone which covers more logic, therefore a cut of larger volume is preferred. Note however that Lemma 5 only claims maximum among min-cuts; if $n(X, X') < K$, there can be other cuts that are still K-feasible but with larger cut size and larger cut volume. A post-processing algorithm used by *FlowMap* tries to grow (X, X') by collapsing all nodes in X', plus one or more in the cut-set, into the sink, and repeat the flow computation; this will force a cut of larger volume, an improvement if it is still K-feasible.

K-Cover Construction

Once minimum height K-feasible cuts have been computed for all nodes, each node v has a K-feasible cone C_v defined by its cut, which has minimum depth. From here, constructing the K-cover $N_M = (V_M, E_M)$ is straight-forward. First, the cones of all PO nodes are included in V_M. Then, for any cone $C_v \in V_M$, cone C_u for each non-PI node $u \in input(C_v)$ is also included in V_M; so is every PI node $u \in input(C_v)$. Similarly, an edge $\langle C_u, C_v\rangle \in E_M$ for each non-

PI node $u \in input(C_v)$; an edge $\langle u, C_v \rangle \in E_M$ for each PI node $u \in input(C_v)$.

Lemma 6 *The K-cover constructed as above is depth optimal.*

This is a linear time procedure, therefore

Theorem 3 *The problem of depth-optimal technology mapping for K-LUT based FPGAs on a Boolean network of n nodes and m edges can be solved in $O(Kmn)$ time.*

Applications

The *FlowMap* algorithm has been used as a centerpiece or a framework for more complicated FPGA logic synthesis and technology mapping algorithms. There are many possible variations that can address various needs in its applications. Some are briefed below; details of such variations/applications can be found in [1,3].

Complicated Delay Models

With minimal change, the algorithm can be applied where non-unit delay model is used, allowing delay of the nodes and/or the edges to vary, as long as they are static. Dynamic delay models, where the delay of a net is determined by its post-mapping structure, cannot be applied to the algorithm. In fact, delay-optimal mapping under dynamic delay models is NP-hard [3].

Complicated Architectures

The algorithm can be adapted to FPGA architectures that are more sophisticated than homogeneous K-LUT arrays. For example, mapping for FPGA with two LUT sizes can be carried out by computing a cone for each size and dynamically choosing the best one.

Multiple Optimization Objectives

While the algorithm is for delay minimization, area minimization (in terms of the number of cones selected) as well as other objectives can also be incorporated, by adapting the criteria for cut selection. The original algorithm considers area minimization by maximizing the volume of the cuts; substantially, more minimization can be achieved by considering more K-feasible cuts and making smart choices to, e.g., increase sharing among input networks, allow cuts of larger heights along no-critical paths, etc. [4] Achieving area optimality, however, is NP-hard [3].

Integration with Other Optimizations

The algorithm can be combined with other types of optimizations, including retiming, logic resynthesis, and physical synthesis.

Cross-References

- Circuit Partitioning: A Network-Flow-Based Balanced Min-Cut Approach
- Performance-Driven Clustering
- Sequential Circuit Technology Mapping

Recommended Reading

The *FlowMap* algorithm, with more details and experimental results, was published in [2]. General information about FPGA can be found in [6]. A good source of concepts and algorithms of network flow is [5]. Comprehensive surveys of FPGA design automation, including many variations and applications of the *FlowMap* algorithm, as well as other algorithms, are presented in [1,3]. A general approach based on the relation ship among K-feasible cuts, K-covers and technology mapping solutions, enabling optimization for complicated objectives, is given in [4].

1. Chen D, Cong J, Pan P (2006) FPGA design automation: a survey. In: Foundations and trends in electronic design automation, vol 1, no 3. Now Publishers, Hanover
2. Cong J, Ding Y (1992) An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. In: Proceedings of IEEE/ACM international conference on computer-aided design, San Jose, pp 48–53
3. Cong J, Ding Y (1996) Combinational logic synthesis for LUT based field programmable gate arrays. ACM Trans Design Autom Electron Syst 1(2):145–204
4. Cong J, Wu C, Ding Y (1999) Cut ranking and pruning: enabling a general and efficient FPGA mapping solution. In: Proceedings of ACM/SIGDA seventh international symposium on FPGAs, Monterey, pp 29–35
5. Tarjan R (1983) Data structures and network algorithms. SIAM, Philadelphia
6. Trimberger S (1994) Field-programmable gate array technology. Springer, Boston

Fractional Packing and Covering Problems

George Karakostas
Department of Computing and Software,
McMaster University, Hamilton, ON, Canada

Keywords

Approximation; Covering; Dual; FPTAS; Packing

Years and Authors of Summarized Original Work

1991; Plotkin, Shmoys, Tardos
1995; Plotkin, Shmoys, Tardos

Problem Definition

This entry presents results on fast algorithms that produce approximate solutions to problems which can be formulated as linear programs (LP) and therefore can be solved exactly, albeit with slower running times. The general format of the family of these problems is the following: Given a set of m inequalities on n variables, and an oracle that produces the solution of an appropriate optimization problem over a convex set $P \in \mathbb{R}^n$, find a solution $x \in P$ that satisfies the inequalities, or detect that no such x exists. The basic idea of the algorithm will always be to start from an infeasible solution x, and use the optimization oracle to find a direction in which the violation of the inequalities can be decreased; this is done by calculating a vector y that is a *dual solution corresponding to* x. Then, x is carefully updated towards that direction, and the process is repeated until x becomes "approximately" feasible. In what follows, the particular problems tackled, together with the corresponding optimization oracle, as well as the different notions of "approximation" used are defined.

- The **fractional packing problem** and its oracle are defined as follows:

PACKING: Given an $m \times n$ matrix A, $b > 0$, and a convex set P in $\mathbb{R}^n$ such that $Ax \geq 0$, $\forall x \in P$, is there $x \in P$ such that $Ax \leq b$?

PACK_ORACLE: Given m-dimensional vector $y \geq 0$ and P as above, return $\bar{x} := \arg\min\{y^{\mathrm{T}} Ax : x \in P\}$.

- The **relaxed fractional packing problem** and its oracle are defined as follows:

RELAXED PACKING: Given $\varepsilon > 0$, an $m \times n$ matrix A, $b > 0$, and convex sets P and $\hat{P}$ in $\mathbb{R}^n$ such that $P \subseteq \hat{P}$ and $Ax \leq 0$, $\forall x \in \hat{P}$, find $x \in \hat{P}$ such that $Ax \leq (1+\varepsilon)b$, or show that $\nexists x \in P$ such that $Ax \leq b$.

REL_PACK_ORACLE: Given m-dimensional vector $y \geq 0$ and P, $\hat{P}$ as above, return $\bar{x} \in \hat{P}$ such that $y^{\mathrm{T}} A\bar{x} \leq \min\{y^{\mathrm{T}} Ax : x \in P\}$.

- The **fractional covering problem** and its oracle are defined as follows:

COVERING: Given an $m \times n$ matrix A, $b > 0$, and a convex set P in $\mathbb{R}^n$ such that $Ax \geq 0$, $\forall x \in P$, is there $x \in P$ such that $Ax \geq b$?

COVER_ORACLE: Given m-dimensional vector $y \geq 0$ and P as above, return $\bar{x} := \arg\max\{y^{\mathrm{T}} Ax : x \in P\}$.

- The **simultaneous packing and covering problem** and its oracle are defined as follows:

SIMULTANEOUS PACKING AND COVERING: Given $\hat{m} \times n$ and $(m-\hat{m}) \times n$ matrices $\hat{A}$, A respectively, $b > 0$ and $\hat{b} > 0$, and a convex set P in $\mathbb{R}^n$ such that $Ax \geq 0$ and $\hat{A}x \leq 0$, $\forall x \in P$, is there $x \in P$ such that $Ax \leq b$, and $\hat{A}x \geq \hat{b}$?

SIM_ORACLE: Given P as above, a constant ν and a dual solution $(y, \hat{y})$, return $\bar{x} \in P$ such that $A\bar{x} \leq \nu b$, and $y^{\mathrm{T}} A\bar{x} - \sum_{i \in I(\nu,\bar{x})} \hat{y}_i \hat{a}_i \bar{x} = \min\{y^{\mathrm{T}} Ax - \sum_{i \in I(\nu,x)} \hat{y}_i \hat{a}_i x : x \text{ a vertex of } P \text{ such that } Ax \leq \nu b\}$, where $I(\nu, x) := \{i : \hat{a}_i x \leq \nu b_i\}$.

- The **general problem** and its oracle are defined as follows:

GENERAL: Given an $m \times n$ matrix A, an arbitrary vector b, and a convex set P in $\mathbb{R}^n$, is there $x \in P$ such that $Ax \leq b$,?

GEN_ORACLE: Given m-dimensional vector $y \geq 0$ and P as above, return $\bar{x} := \arg\min\{y^T Ax : x \in P\}$.

Definitions and Notation

For an error parameter $\varepsilon > x0$, a point $x \in P$ is an *ε-approximation solution* for the fractional packing (or covering) problem if $Ax \leq (1+\varepsilon)b$ (or $Ax \geq (1-\varepsilon)b$). On the other hand, if $x \in P$ satisfies $Ax \leq b$ (or $Ax \geq b$), then x is an *exact solution*. For the GENERAL problem, given an error parameter $\varepsilon > 0$ and a positive tolerance vector d, $x \in P$ is an *ε-approximation solution* if $Ax \leq b + \varepsilon d$ and an *exact solution* if $Ax \leq b$. An *ε-relaxed decision procedure* for these problems either finds an ε-approximation solution or correctly reports that no exact solution exists. In general, for a minimization (maximization) problem, an $(1+\varepsilon)$-approximation ($(1-\varepsilon)$-*approximation*) *algorithm* returns a solution at most $(1+\varepsilon)$ (at least $(1-\varepsilon)$) times the optimal.

The algorithms developed work within time that depends polynomially on ε^{-1}, for any error parameter $\varepsilon > 0$. Their running time will also depend on the *width* ρ of the convex set P relative to the set of inequalities $Ax \leq b$ or $Ax \geq b$ defining the problem at hand. More specifically, the width ρ is defined as follows for each one of the problems considered here:

- PACKING: $\rho := \max_i \max_{x \in P} \frac{a_i x}{bi}$.
- RELAXED PACKING: $\hat{\rho} := \max_i \max_{x \in \hat{P}} \frac{a_i x}{b_i}$.
- COVERING: $\rho := \max_i \max_{x \in P} \frac{a_i x}{bi}$.
- SIMULTANEOUS PACKING AND COVERING: $\rho := \max_{x \in P} \max\{\max_i \frac{a_i x}{bi}, \max_i \frac{\hat{a}_i x}{\hat{b}_i}\}$.
- GENERAL: $\rho := \max_i \max_{x \in P} \frac{|a_i x - b_i|}{d_i} + 1$, where d is the tolerance vector defined above.

Key Results

Many of the results below were presented in [8] by assuming a model of computation with exact arithmetic on real numbers and exponentiation in a single step. But, as the authors mention [8], they can be converted to run on the RAM model by using approximate exponentiation, a version of the oracle that produces a *nearly* optimal solution, and a limit on the numbers used that is polynomial in the input length similar to the size of numbers used in exact linear programming algorithms. However, they leave as an open problem the construction of ε-approximate solutions using polylogarithmic precision for the general case of the problems they consider (as can be done, e.g., in the multicommodity flow case [4]).

Theorem 1 *For $0 < \varepsilon \leq 1$, there is a deterministic ε-relaxed decision procedure for the fractional packing problem that uses $O(\varepsilon^{-2}\rho \log(m\varepsilon^{-1}))$ calls to* PACK_ORACLE*, plus the time to compute Ax for the current iterate x between consecutive calls.*

For the case of P being written as a product of smaller-dimension polytopes, i.e., $P = P^1 \times \ldots \times P^k$, each P^l with width ρ^l $\left(\text{obviously} \rho \leq \sum_l \rho^l\right)$, and a separate PACK_ORACLE for each P^l,A^l, then randomization can be used to potentially speed up the algorithm. By using the notation $PACK\_ORACLE_l$ for the P^l,A^l oracle, the following holds:

Theorem 2 *For $0 < \varepsilon \leq 1$, there is a randomized ε-relaxed decision procedure for the fractional packing problem that is expected to use*

$$O\left(\varepsilon^{-2}\left(\sum_l \rho^l\right)\log(m\varepsilon^{-1}) + k\log(\rho\varepsilon^{-1})\right)$$

calls to $PACK\_ORACLE_l$ for some $l \in \{1,\ldots,k\}$ (possibly a different l in every call), plus the time to compute $\sum_l A^l x^l$ for the

current iterate $x = (x^1, x^2, \ldots, x^k)$ *between consecutive calls.*

Theorem 2 holds for RELAXED PACKING as well, if ρ is replaced by $\hat{\rho}$ and PACK_ORACLE by REL_PACK_ORACLE.

In fact, one needs only an *approximate* version of PACK_ORACLE. Let $C_{\mathcal{P}}(y)$ be the minimum cost $y^{\mathrm{T}}Ax$ achieved by PACK_ORACLE for a given y.

Theorem 3 *Let* PACK_ORACLE *be replaced by an oracle that, given vector* $y \geq 0$*, finds a point* $\bar{x} \in P$ *such that* $y^{\mathrm{T}}A\bar{x} \leq (1+\varepsilon/2)C_{\mathcal{P}}(y) + (\varepsilon/2)\lambda y^{\mathrm{T}}b$*, where* λ *is minimum so that* $Ax \leq \lambda b$ *is satisfied by the current iterate* x*. Then, Theorems 1 and 2 still hold.*

Theorem 3 shows that even if no efficient implementation exists for an oracle, as in, e.g., the case when this oracle solves an NP-hard problem, a fully polynomial approximation scheme for it suffices.

Similar results can be proven for the fractional covering problem ($COVER\_ORACLE_l$ is defined similarly to $PACK\_ORACLE_l$ above):

Theorem 4 *For* $0 < \varepsilon < 1$*, there is a deterministic* ε*-relaxed decision procedure for the fractional covering problem that uses* $O(m + \rho \log^2 m + \varepsilon^{-2}\rho \log(m\varepsilon^{-1}))$ *calls to* COVER_ORACLE*, plus the time to compute* Ax *for the current iterate* x *between consecutive calls.*

Theorem 5 *For* $0 < \varepsilon < 1$*, there is a randomized* ε*-relaxed decision procedure for the fractional packing problem that is expected to use* $O\left(mk + \left(\sum_i \rho^l\right)\log^2 m + k\log\varepsilon^{-1} + \varepsilon^{-2}\left(\sum_l \rho^l\right)\log(m\varepsilon^{-1})\right)$ *calls to* $COVER\_ORACLE_l$ *for some* $l \in \{1,\ldots,k\}$ *(possibly a different* l *in every call), plus the time to compute* $\sum_l A^l x^l$ *for the current iterate* $x = (x^1, x^2, \ldots, x^k)$ *between consecutive calls.*

Let $C_{\mathcal{C}}(y)$ be the maximum cost $y^{\mathrm{T}}Ax$ achieved by COVER_ORACLE for a given y.

Theorem 6 *Let* COVER_ORACLE *be replaced by an oracle that, given vector* $y \geq 0$*, finds a point* $\bar{x} \in P$ *such that* $y^{\mathrm{T}}A\bar{x} \geq (1-\varepsilon/2)C_{\mathcal{C}}(y) - (\varepsilon/2)\lambda y^{\mathrm{T}}b$*, where* λ *is maximum so that* $Ax \geq \lambda b$ *is satisfied by the current iterate* x*. Then, Theorems 4 and 5 still hold.*

For the simultaneous packing and covering problem, the following is proven:

Theorem 7 *For* $0 < \varepsilon \leq 1$*, there is a randomized* ε*-relaxed decision procedure for the simultaneous packing and covering problem that is expected to use* $O(m^2(\log^2 \rho)\varepsilon^{-2}\log(\varepsilon^{-1}m\log\rho))$ *calls to* SIM_ORACLE*, and a deterministic version that uses a factor of* $\log\rho$ *more calls, plus the time to compute* $\hat{A}x$ *for the current iterate* x *between consecutive calls.*

For the GENERAL problem, the following is shown:

Theorem 8 *For* $0 < \varepsilon < 1$*, there is a deterministic* ε*-relaxed decision procedure for the* GENERAL *problem that uses* $O(\varepsilon^{-2}\rho^2\log(m\rho\varepsilon^{-1}))$ *calls to* GEN_ORACLE*, plus the time to compute* Ax *for the current iterate* x *between consecutive calls.*

The running times of these algorithms are proportional to the width ρ, and the authors devise techniques to reduce this width for many special cases of the problems considered. One example of the results obtained by these techniques is the following: If a packing problem is defined by a convex set that is a product of k smaller-dimension convex sets, i.e., $P = P^1 \times \ldots \times P^k$, and the inequalities $\sum_l A^l x^l \leq b$, then there is a randomized ε-relaxed decision procedure that is expected to use $O(\varepsilon^{-2}k\log(m\varepsilon^{-1}) + k\log k)$ calls to a subroutine that finds a minimum-cost point in $\hat{P}^l = \{x^l \in P^l : A^l x^l \leq b\}$, $l = 1, \ldots, k$ and a deterministic version that uses $O(\varepsilon^{-2}k^2\log(m\varepsilon^{-1}))$ such calls, plus the time to compute Ax for the current iterate x between consecutive calls. This result can be applied to the multicommodity flow problem, but the required subroutine is a single-source minimum-cost flow computation, instead of a shortest-path calculation needed for the original algorithm.

Applications

The results presented above can be used in order to obtain fast approximate solutions to linear programs, even if these can be solved exactly by LP algorithms. Many approximation algorithms are based on the rounding of the solution of such programs, and hence one might want to solve them approximately (with the overall approximation factor absorbing the LP solution approximation factor), but more efficiently. Two such examples, which appear in [8], are mentioned here.

Theorems 1 and 2 can be applied for the improvement of the running time of the algorithm by Lenstra, Shmoys, and Tardos [5] for the scheduling of unrelated parallel machines without preemption $(R||C_{\max})$: N jobs are to be scheduled on M machines, with each job i scheduled on exactly one machine j with processing time p_{ij}, so that the maximum total processing time over all machines is minimized. Then, for any fixed $r > 1$, there is a deterministic $(1 + r)$-approximation algorithm that runs in $O(M^2 N \log^2 N \log M)$ time and a randomized version that runs in $O(MN \log M \log N)$ expected time. For the version of the problem with preemption, there are polynomial-time approximation schemes that run in $O(MN^2 \log^2 N)$ time and $O(MN \log N \log M)$ expected time in the deterministic and randomized case, respectively.

A well-known lower bound for the metric Traveling Salesman Problem (metric TSP) on N nodes is the Held-Karp bound [2], which can be formulated as the optimum of a linear program over the *subtour elimination polytope*. By using a randomized minimum-cut algorithm by Karger and Stein [3], one can obtain a randomized approximation scheme that computes the Held-Karp bound in $O(N^4 \log^6 N)$ expected time.

Open Problems

The main open problem is the further reduction of the running time for the approximate solution of the various fractional problems. One direction would be to improve the bounds for specific problems, as has been done very successfully for the multicommodity flow problem in a series of papers starting with Shahrokhi and Matula [9]. Currently, the best running times for several versions of the multicommodity flow problems are achieved by Madry [6]. Shahrokhi and Matula [9] also led to a series of results by Grigoriadis and Khachiyan developed independently to [8], starting with [1] which presents an algorithm with a number of calls smaller than the one in Theorem 1 by a factor of $\log(m\varepsilon^{-1})/\log m$. Considerable effort has been dedicated to the reduction of the dependence of the running time on the width of the problem or the reduction of the width itself (e.g., see [10] for sequential and parallel algorithms for mixed packing and covering), so this can be another direction of improvement.

A problem left open by [8] is the development of approximation schemes for the RAM model that use only *polylogarithmic in the input length* precision and work for the general case of the problems considered.

Cross-References

- Approximation Schemes for Makespan Minimization

Recommended Reading

1. Grigoriadis MD, Khachiyan LG (1994) Fast approximation schemes for convex programs with many blocks and coupling constraints. SIAM J Optim 4:86–107
2. Held M, Karp RM (1970) The traveling-salesman problem and minimum cost spanning trees. Oper Res 18:1138–1162
3. Karger DR, Stein C (1993) An $\tilde{O}(n^2)$ algorithm for minimum cut. In: Proceeding of 25th annual ACM symposium on theory of computing (STOC), San Diego, pp 757–765
4. Leighton FT, Makedon F, Plotkin SA, Stein C, Tardos É, Tragoudas S (1995) Fast approximation algorithms for multicommodity flow problems. J Comput Syst Sci 50(2):228–243
5. Lenstra JK, Shmoys DB, Tardos É (1990) Approximation algorithms for scheduling unrelated parallel machines. Math Program A 24:259–272
6. Madry A (2010) Faster approximation schemes for fractional multicommodity flow problems via

dynamic graph algorithms. In: Proceedings of 42nd ACM symposium on theory of computing (STOC), Cambridge, pp 121–130
7. Plotkin SA, Shmoys DB, Tardos É (1991) Fast approximation algorithms for fractional packing and covering problems. In: Proceedings of 32nd annual IEEE symposium on foundations of computer science (FOCS), San Juan, pp 495–504
8. Plotkin SA, Shmoys DB, Tardos É (1995) Fast approximation algorithms for fractional packing and covering problems. Math Oper Res 20(2):257–301. Preliminary version appeared in [6]
9. Shahrokhi F, Matula DW (1990) The maximum concurrent flow problem. J ACM 37:318–334
10. Young NE (2001) Sequential and parallel algorithms for mixed packing and covering. In: Proceedings of 42nd annual IEEE symposium on foundations of computer science (FOCS), Las Vegas, pp 538–546

Frequent Graph Mining

Takashi Washio
The Institute of Scientific and Industrial Research, Osaka University, Ibaraki, Osaka, Japan

Keywords

Anti-monotonicity; Canonical depth sequence; Canonical graph representation; Data-driven enumeration; DFS code; Frequent subgraph; Pattern growth

Years and Authors of Summarized Original Work

2000; Inokuchi, Washio
2002; Yan, Han
2004; Nijssen, Kok

Problem Definition

This problem is to enumerate all subgraphs appearing with frequencies not less than a threshold value in a given graph data set. Let $G(V, E, L, \ell)$ be a labeled graph where V is a set of vertices, $E \subseteq V \times V$ a set of edges, L a set of labels and $\ell : V \cup E \rightarrow L$ a labeling function. A labeled graph $g(v, e, L, \ell)$ is a subgraph of $G(V, E, L, \ell)$, *i.e.*, $g \sqsubseteq G$, if and only if a mapping $f : v \rightarrow V$ exists such that $\forall u_i \in v, f(u_i) \in V, \ell(u_i) = \ell(f(u_i))$, and $\forall (u_i, u_j) \in e, (f(u_i), f(u_j)) \in E, \ell(u_i, u_j) = \ell(f(u_i), f(u_j))$. Given a graph data set $D = \{G_i | i = 1, \ldots, n\}$, a support of g in D is a set of all G_i involving g in D, *i.e.*, $D(g) = \{G_i | g \sqsubseteq G_i \in D\}$. Under a given threshold frequency called a minimum support $minsup > 0$, g is said to be frequent, if the size of $D(g)$ *i.e.*, $|D(g)|$, is greater than or equal to $minsup$. Generic frequent graph mining is a problem to enumerate all frequent subgraphs g of D, while most algorithms focus on connected and undirected subgraphs. Some focus on induced subgraphs or limit the enumeration to closed frequent subgraphs where each of them is maximal in the frequent subgraphs having an identical support.

Key Results

Study of the frequent graph mining was initiated in the mid-1990s under motivation to analyze complex structured data acquired and accumulated in our society. Their major issue has been principles to efficiently extract frequent subgraphs embedded in a given graph data set. They invented many original canonical graph representations adapted to the data-driven extraction, which are different from these proposed in studies of efficient isomorphism checking [1] and graph enumeration without duplications [2].

Pioneering algorithms of frequent graph mining, SUBDUE [3] and GBI [4], did not solve the aforementioned standard problem but greedily extracted subgraphs concisely describing the original graph data under some measures such as minimum description length (MDL). The earliest algorithms for deriving a complete set of the frequent subgraphs are WARMR [5] and its extension FARMER [6]. They can flexibly focus on various types of frequent subgraphs for the enumeration by applying inductive logic program-

ming (ILP) in artificial intelligence, while they are not very scalable in the size of the enumerated subgraphs.

AGM proposed in 2000 [7, 8] was an epoch-making study in the sense that it combined frequent item set mining [9] and the graph enumeration and enhanced the scalability for practical applications. It introduced technical strategies of (1) incremental candidate enumeration based on anti-monotonicity of the subgraph frequency, (2) canonical graph representation to avoid duplicated candidate subgraph extractions, and (3) data-driven pruning of the candidates by the minimum support. The anti-monotonicity is a fundamental nature of the subgraph frequency that $|D(g_1)| \leq |D(g_2)|$ for any subgraphs g_1 and g_2 in D if $g_2 \sqsubseteq g_1$. Dozens of frequent graph mining algorithms have been studied along this line after 2000. In the rest of this entry, gSpan [10] and Gaston [11], considered to be the most efficient up to date, are explained.

gSpan

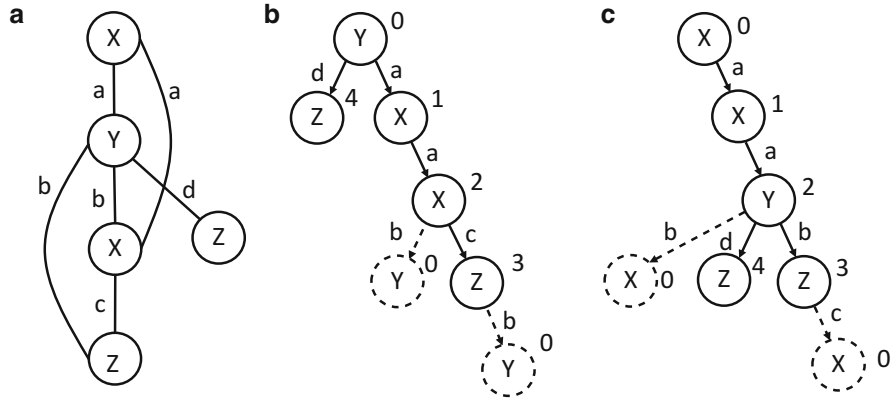
gSpan derives all frequent connected subgraphs in a given data set of connected and undirected graphs [10]. For the aforementioned strategy (1), it applies a pattern growth technique which is data-driven enumeration of candidate frequent subgraphs. It is performed by tracing vertices and edges of each data graph G in a DFS manner. Figure 1b is an example search tree generated by starting from the vertex labeled as Y in the graph (a). In a search tree T, the vertex for the next visit is the one reachable from the current vertex by passing through an edge untraced yet in G. If the vertex for the next visit is the one visited earlier in G, the edge is called a backward edge otherwise a forward edge. They are depicted by dashed and solid lines, respectively, in Fig. 1b. When no more untraced edges are available from the current vertex, the search backtracks to the nearest vertex having the untraced edges. Any subtree of T represents a subgraph of G. We denote the sets of the forward and the backward edges in T as $E_{f,T} = \{e | \forall i, j, i < j, e = (v_i, v_j) \in E\}$ and $E_{b,T} = \{e | \forall i, j, i > j, e = (v_i, v_j) \in E\}$, respectively, where i and j are integer indices numbered at the vertices in their visiting order in T.

There exist many trees T representing an identical graph G as another tree of the graph (a) shown in Fig. 1c. This ambiguity causing duplication and miss in the candidate graph enumeration is avoided by introducing the strategy (2). gSpan applied the following three types of partial orders of the edges in T. Given $e_1 = (v_{i_1}, v_{j_1})$ and $e_2 = (v_{i_2}, v_{j_2})$,

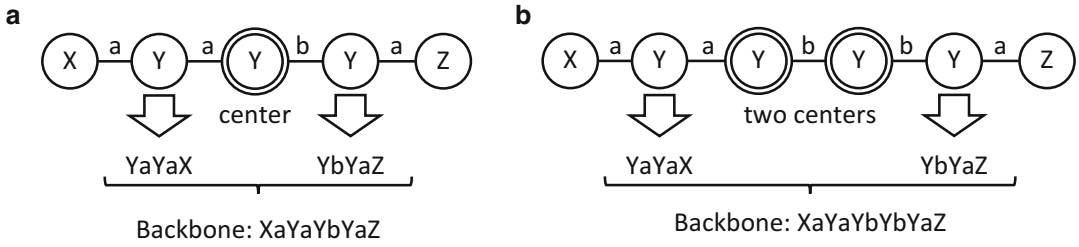
$$
\begin{aligned}
e_1 \prec_{f,T} e_2 \quad & \text{if and only if } j_1 < j_2 \text{ for } e_1, e_2 \in E_{f,T}, \\
e_1 \prec_{b,T} e_2 \quad & \text{if and only if (i) } i_1 < i_2 \text{ or (ii) } i_1 = i_2 \text{ and } j_1 < j_2, \text{ for } e_1, e_2 \in E_{b,T}, \\
e_1 \prec_{bf,T} e_2 \quad & \text{if and only if (i) } i_1 < j_2 \text{ for } e_1 \in E_{b,T}, e_2 \in E_{f,T} \\
& \text{or (ii) } j_1 \leq i_2 \text{ for } e_1 \in E_{f,T}, e_2 \in E_{b,T}.
\end{aligned}
$$

The combination of these partial orders is known to give a linear order of the edges. We also assume a total order of the labels in L and define a representation of T, a DFS code, as a sequence of 5-tuples $((v_i, v_j), \ell(v_i), \ell((v_i, v_j)), \ell(v_j))$ following the trace order of the DFS in T. A DFS code is smaller if smaller edges and smaller labels appear in earlier 5-tuples in the sequence. Accordingly, we define the search tree T having the minimum DFS code as a canonical representation of G. The search tree in Fig. 1c is canonical, since its DFS code is the minimum. As any subtree of a canonical T has its minimum DFS code, it is also canonical for its corresponding subgraph.

Moreover, gSpan applies the DFS which chooses the untraced edge having the smallest 5-tuples at the current vertex for visiting the next vertex. This focuses on the minimum DFS code and ensures to enumerate the canonical



Frequent Graph Mining, Fig. 1 A data graph and its search trees



Frequent Graph Mining, Fig. 2 Examples of paths and their backbones

subtree of every subgraph before finding its other noncanonical subtrees. This efficiently prunes an infrequent subgraph without matching its multiple DSF codes in (3). In this manner, the canonical graph representation of gSpan is fully adapted to its search algorithm.

Gaston

Gaston also derives all frequent connected subgraphs in a given data set of connected and undirected graphs [11]. It uses the polynomial time complexity of the enumeration of paths and free trees. Gaston uses a canonical path representation, a backbone, in (2). Two sequences of the labels of the vertices and edges starting from a center, which is a middle vertex, in the path to the both terminals are derived as shown in Fig. 2a, and the reverse of the lexicographically smaller sequence with the appended larger sequence is defined to be the backbone. In case of a path having an even number of vertices, two centers, which are two middle vertices, are used as shown in Fig. 2b. Starting from a single vertex, Gaston extends a path by adding a vertex to one of the terminals in the strategy (1). Finally, it efficiently counts the frequency of the extended path in the data set by using its backbone and prunes the infrequent paths in (3).

Gaston further enumerates free trees involving a frequent path as the longest path by iteratively adding vertices to the vertices in the free trees except for the terminal vertices of the path. Since the set of the free trees having a distinct backbone as its longest path is also distinct, the set does not intersect each other. This reduces the complexity of the enumeration in (1). Moreover, Gaston derives a canonical representation of a free tree, a canonical depth sequence, for (2) by transforming the tree to a rooted and ordered tree where the root is the center of its longest path, and its vertices and edges are arranged in a lexicographically descending order of the labels. If the two center exists in the path, the free tree is partitioned for each center, and each free tree

is represented by its canonical depth sequence. Similarly to the DFS code of gSpan, any subtree involving the root in this rooted and ordered tree is a canonical depth sequence. This is beneficial for (1), since all canonical depth sequences are incrementally obtained in the depth first search. Gaston efficiently prunes the infrequent free trees in the data set by using the canonical depth sequence in (3).

Gaston further enumerates cyclic subgraphs from a frequent free tree by iteratively adding edges bridging vertex pairs in the tree in (1). For (2), Gaston avoids duplicated enumerations of the cyclic subgraphs by using Nauty algorithm for the graph isomorphism checking [1]. It prunes the infrequent cyclic subgraphs of the data set in (3) and finally derives the frequent subgraphs. Gaston works very efficiently for the sparse data graphs, since the candidate cyclic subgraphs is less in such graphs.

URLs to Code and Data Sets

gSpan suite (http://www.cs.ucsb.edu/~xyan/software/gSpan.htm), Gaston suite (http://www.liacs.nl/~snijssen/gaston/). Other common suites can be found for various frequent substructure mining (http://hms.liacs.nl/index.html).

Cross-References

- Enumeration of Non-crossing Geometric Graphs
- Enumeration of Paths, Cycles, and Spanning Trees
- Frequent Pattern Mining
- Graph Isomorphism
- Matching in Dynamic Graphs
- Tree Enumeration

Recommended Reading

1. McKay BD, Piperno A (2013) Practical graph isomorphism, II. J Symb Comput 60:94–112
2. Roberts F, Tesman B (2011) Applied combinatorics, 2nd edn. CRC, New York
3. Cook J, Holder L (1994) Substructure discovery using minimum description length and background knowledge. J Artif Intell Res 1:231–255
4. Yoshida K, Motoda H, Indurkhya N (1994) Graph-based induction as a unified learning framework. J Appl Intell 4:297–328
5. Dehaspe L, Toivonen H (1999) Discovery of frequent datalog patterns. Data Min Knowl Discov 3(1):7–36
6. Nijssen S, Kok J (2001) Faster association rules for multiple relations. In: Proceedings of the IJCAI2001: 17th international joint conference on artificial intelligence, Seattle, vol 2, pp 891–896
7. Inokuchi A, Washio T, Motoda H (2000) An Apriori-based algorithm for mining frequent substructures from graph data. In: Proceedings of the PKDD2000: principles of data mining and knowledge discovery, 4th Europian conference, Lyon. Lecture notes in artificial intelligence (LNAI), vol 1910, pp 13–23
8. Inokuchi A, Washio T, Motoda H (2003) Complete mining of frequent patterns from graphs: mining graph data. Mach Learn 50:321–354
9. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the VLDB1994: 20th very large dada base conference, Santiago de Chile, pp 487–499
10. Yan X, Han J (2002) gspan: graph-based substructure pattern mining. In: Proceedings of the ICDM2002: 2nd IEEE conference on data mining, Maebashi City, pp 721–724
11. Nijssen S, Kok JN (2004) A quickstart in frequent structure mining can make a difference. In: Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining, Seattle, pp 647–652

Frequent Pattern Mining

Takeaki Uno
National Institute of Informatics, Chiyoda, Tokyo, Japan

Keywords

Apriori; Backtrack; Database; Data mining; FIMI; FP-growth; FP-tree; Itemset mining; Frequency; Monotone property

Years and Authors of Summarized Original Work

2004; Uno, Kiyomi, Arimura

Problem Definition

Pattern mining is a fundamental problem in data mining. The problem is to find all the patterns appearing in the given database frequently. For a set $E = \{1, \ldots, n\}$ of items, an *itemset* (also called a pattern) is a subset of E. Let $\mathcal{D}$ be a given *database* composed of *transactions* $R_1, \ldots, R_m$, $R_i \subseteq E$. For an itemset P, an *occurrence* of P is a transaction of $\mathcal{D}$ such that $P \subseteq R$, and the *occurrence set* $\mathrm{Occ}(P)$ is the set of occurrences of P. The *frequency* of P, also called *support*, is $|\mathrm{Occ}(P)|$ and denoted by $\mathrm{frq}(P)$. For a given constant σ called *minimum support*, an itemset P is *frequent* if $\mathrm{frq}(P) \geq \sigma$. For given a database and a minimum support, *frequent itemset mining* is the problem of enumerating all frequent itemsets in $\mathcal{D}$.

Key Results

The enumeration is solved in output polynomial time [1], and the space complexity is input polynomial [5]. Many algorithms have been proposed for practical efficiency on real-world data [7, 8, 10, 11] that are drastically fast. We show algorithm LCM that is the winner in the competition [6], and several techniques used in LCM.

Algorithms

There have been a lot of algorithms for this problem. The itemsets satisfy the following *monotone property*, and this is used in almost all existing algorithms.

Lemma 1 *For any itemsets P and Q such that $P \supseteq Q$, there holds $\mathrm{frq}(P) \leq \mathrm{frq}(Q)$. In particular, $\mathrm{Occ}(P) \subseteq \mathrm{Occ}(Q)$.* □

Using the monotone property, we can enumerate all frequent itemsets from $\emptyset$ by recursively adding items. □

Lemma 2 *Any frequent itemset P of size k is generated by adding an item to a frequent itemset of size $k-1$.*

Apriori

Apriori algorithm was proposed in the first paper of frequent pattern mining, by Agrawal et al. in 1993 [1]. The computational resources are not enough in the era, and the database could not fit memory, thus stored in an HDD. Apriori is designed to be efficient in such environments so that it scans the database only few times. Apriori is a breadth-first search algorithm that iteratively generates all frequent itemsets of size 1, size 2, and so on. Apriori generates candidate itemsets by adding an item to each frequent itemset of size $k-1$. From the monotone property, any frequent itemset of size k is in the candidate itemsets. Apriori then checks the inclusion relation between a transaction and all candidates. By doing this for all transactions, the frequencies of candidates are computed and infrequent candidates are removed. The algorithm is written as follows:

Algorithm Apriori($\mathcal{D}, \sigma$):

1. $\mathcal{P}_0 = \{\emptyset\}$; $k := 0$;
2. **while** $\mathcal{P}_k \neq \emptyset$ **do**
3. $\mathcal{P}_{k+1} := \emptyset$;
4. **for** each $P \in \mathcal{P}_k$, $\mathrm{frq}(P) := 0$; $\mathcal{P}_{k+1} := \mathcal{P}_{k+1} \cup \{P \cup \{i\} \mid i \in E\}$
5. **for** each $R \in \mathcal{D}$, $\mathrm{frq}(P) := \mathrm{frq}(P) + 1$ for all $P \in \mathcal{P}_{k+1}, P \subseteq R$.
6. remove all P from $\mathcal{P}_{k+1}$ satisfying $\mathrm{frq}(P) < \sigma$
7. **output** all $P \in \mathcal{P}_{k+1}$; $k := k + 1$
8. **end while**

The space complexity of Apriori is $O(n\mu)$ where μ is the number of frequent itemsets of $\mathcal{D}$. The time complexity is $O(n||\mathcal{D}||\mu)$ where $||\mathcal{D}|| = \sum_{R \in \mathcal{D}} |R|$ is the size of $\mathcal{D}$. Hence Apriori is output polynomial time.

Backtrack Algorithm

Backtrack algorithm is a depth-first search-based frequent itemset mining algorithm that is first proposed Bayardo et al. [5] in 1998. The amount of memory in a computer was rapidly increasing in the era, and thus the databases began to fit the memory. We can then reduce the memory space for storing candidate itemsets and thus huge amount of itemsets can be enumerated. Moreover, a technique so-called down project accelerates the computation. According to the monotone property, we can see that $P \cup \{i\}$ is included in a transaction R only if $R \in \mathrm{Occ}(P)$ holds. By using this, down project reduces the checks only with transactions in $\mathrm{Occ}(P)$. Particularly, we can see that $\mathrm{Occ}(P \cup \{i\}) = \mathrm{Occ}(P) \cap \mathrm{Occ}(\{i\})$. Moreover, we can reduce the check for the duplication by using a technique so-called tail extension. We denote the maximum item in P by $tail(P)$. Tail extension generates itemsets $P \cup \{i\}$ only with $i, i > tail(P)$. In this way, any frequent itemset P is generated uniquely from another frequent itemset; thus duplications are efficiently avoided, by recursively generating with tail extensions.

Algorithm BackTrack $(P, \mathrm{Occ}(P), \sigma)$:

1. **output** P
2. **for** each item $i > tail(P)$ **do**
3. **if** $|\mathrm{Occ}(P) \cap \mathrm{Occ}(\{i\})| \geq \sigma$, then **call** BackTrack $(P \cup \{i\}, \mathrm{Occ}(P) \cap \mathrm{Occ}(\{i\}), \sigma)$
4. **end for**

The space complexity of BackTrack is $O(||\mathcal{D}||)$; thereby BackTrack is polynomial space. The time complexity is $O(||\mathcal{D}||\mu)$, since step 3 is done by marking transactions in $\mathrm{Occ}(P)$ and checking whether each transaction of $\mathrm{Occ}(\{i\})$ is marked in constant time. Moreover, since the depth of recursion is at most n, the delay of BackTrack is $O(n||\mathcal{D}||)$. BackTrack with down project reduces practical computation time in order of magnitude, in implementation competitions FIMI03 and FIMI04 [6].

Database Reduction

The technique of database reduction was first developed in FP-growth by Han et al. [8] and modified in LCM by Uno et al. [11]. Database reduction drastically reduces practical computation time, as shown in the experiments in [6].

We observe that down project removes the unnecessary transactions from the database given to a recursive call, where unnecessary transactions are those never used in the recursion, and this fastens the computation. The idea of database reduction is to further remove unnecessary items from the database. The unnecessary items are (1) items i satisfying $i < tail(P)$ and (2) items i such that $P \cup \{i\}$ is not frequent. Items of (1) are never used because of the rule of tail extension. (2) comes from that $P \cup \{i\} \cup \{j\}$ is not frequent for any item j, by the monotone property. Thus, the removal of these items never disturbs the enumeration. The database obtained by removing unnecessary items from each transaction of $\mathrm{Occ}(P)$ is called the *conditional database*.

In the deep of recursion, the conditional database tends to have few items since $tail(P)$ is large and $\mathrm{frq}(P)$ is small. In such cases, several transactions would be identical. The computation for the identical transactions is the same, and thus we unify these transactions and put a mark of their quantity to the unified transaction as the representation of the multiplicity. For example, three transactions $R_1, R_2, R_3 = \{100, 105, 110\}$ are replaced by $R_j = \{100, 105, 110\}$ and a mark "three" is put to R_j. By this, the computation time on the bottom levels of the recursion is drastically shortened when σ is large. This is because conditional databases usually have k items in the bottom levels where k is a small constant and thus can have at most 2^k different transactions. The obtained database is called the *reduced database* and is denoted by $\mathcal{D}^*(P, \sigma)$.

The computation for the unification of identical transactions can be done by, for example, radix sort in $O(||\mathcal{D}^*(P, \sigma)||)$ time [11]. FP-growth further reduces by representing the database by a trie [7, 8]. However, experiments in [6] show that the overheads of trie are often larger than the gain; thus in many cases FP-growth is slower than LCM. The computation of frequent itemset mining generates recursions widely spread as the depth, thus so-called bottom expanded. In such case, the computation time on

the bottom levels dominates the total computation time [9]; thus database reduction performs very well.

Delivery

Delivery [10, 11] is a technique to compute $\mathrm{Occ}(P \cup \{i\})$ for all $i > tail(P)$ at once. Down project computes $\mathrm{Occ}(P \cup \{i\})$ in $O(|\mathrm{Occ}(\{i\})|)$ time and thus takes $O(||\mathcal{D}||)$ time for all i. Delivery computes $\mathrm{Occ}(P \cup \{i\})$ for all i at once in $(||\mathrm{Occ}(P)||)$ time. The idea is to find all $P \cup \{i\}$ that are included in R, for each transaction $R \in \mathrm{Occ}(P)$. Actually, $P \cup \{i\} \subseteq R$ iff $i \in R$; thus this is done by just scanning items $i > tail(P)$. The algorithm is described as follows:

1. $\mathrm{Occ}(P \cup \{i\}) := \emptyset$ for each $i > tail(P)$
2. **for** each $R \in \mathrm{Occ}(P)$ **do**
3. **for** each item $i \in R, i > tail(P)$, insert R to $\mathrm{Occ}(P \cup \{i\})$
4. **end for**

By using the reduced database $\mathcal{D}^*(P, \sigma)$, delivery is done in $O(||\mathcal{D}^*(P, \sigma)||)$ time. Note that the frequency is the sum of multiplications of transactions in the reduced database.

Generalizations and Extensions

The frequent itemset mining problem is extended by varying patterns and databases, such as trees in XML databases, labeled graphs in chemical compound databases, and so on. Let $\mathcal{L}$ be a class of structures, and $\preceq$ be an binary relation on $\mathcal{L}$. A member of $\mathcal{L}$ is called a *pattern*. Suppose that we are given a *database* $\mathcal{D}$ composed of records $R_1, \ldots, R_m$, $R_i \in \mathcal{L}$. Itemset mining is the case that $\mathcal{L} = 2^E$ and $a \preceq b$ holds iff $a \subseteq b$. For a pattern $P \in \mathcal{L}$, an *occurrence* of P is a record $R \in \mathcal{D}$ such that $P \preceq R$, and the other notations are defined in the same way. For given a database and a minimum support, *frequent pattern mining* is the problem of enumerating all the frequent patterns in $\mathcal{D}$.

When $\mathcal{L}$ is arbitrary, the frequent pattern mining is hard. Thus, we often assume that $(\mathcal{L}, \preceq)$ is a lattice, and there is an element $\perp$ of $\mathcal{L}$ such that $\perp \preceq P$ holds for any $P \in \mathcal{L}$. We then have the monotone property.

Lemma 3 *For any $P, Q \in \mathcal{L}$ satisfying $P \preceq Q$, there holds* $\mathrm{frq}(P) \geq \mathrm{frq}(Q)$. □

Let $\mathrm{suc}(P)$ (resp., $\mathrm{prc}(P)$) be the set of elements $Q \in \mathcal{L} \setminus \{P\}$ such that $P \preceq Q$ (resp., $Q \preceq P$) holds and no $X \in \mathcal{L} \setminus \{P, Q\}$ satisfies $P \preceq X \preceq Q$ (resp., $Q \preceq X \preceq P$). Using the monotone property, we can enumerate all frequent patterns from $\perp$ by recursively generating all elements of $\mathrm{suc}(P)$.

In this general setting, Apriori needs an assumption that $(\mathcal{L}, \preceq)$ is modular; thus for any P, Q such that $P \preceq Q$, the length of any maximal chain $P \preceq X_1 \cdots X_k \preceq Q$ is identical. By this assumption, we can define the size of a pattern P by the length of the maximal chain from $\perp$ to P. Apriori then works by replacing $\{P \cup \{i\} \mid i \in E\}$ of step 4 by $\mathrm{suc}(P)$.

Let T be the time to generate a pattern in $\mathrm{suc}(P)$ and T' be the time to evaluate $a \preceq b$. Note that T' may be large, for example, in the case that $\mathcal{L}$ is the set of graphs and T' is the time for graph isomorphism. Apriori generates $|\mathrm{suc}(P)|$ patterns for each pattern P, and we have to check whether each generated pattern is already in $\mathcal{P}_{i+1}$ by comparing P and each member of $\mathcal{P}_{i+1}$. Thus, the total computation time is $O(s(T + T'(\mu + |\mathcal{D}|)))$ where s is the maximum size of $\mathrm{suc}(P)$.

The depth-first search algorithm needs an alternative for tail extension. The alternative is given by reverse search technique proposed by Avis and Fukuda [4]. A pattern Q is generated from many patterns in $\mathrm{prc}(Q)$, and this makes duplications. We avoid this by defining the parent $P(Q)$ by one of $\mathrm{prc}(Q)$ and allow to generate Q only from $P(Q)$, so that Q is uniquely generated. For example, the same as tail extension, we define an order in $\mathrm{prc}(Q)$ and define $P(Q)$ by the minimum one in the order.

Algorithm Backtrack2 $(P, \mathrm{Occ}(P), \sigma)$:

1. **output** P
2. **for** each $Q \in \mathrm{suc}(P)$ **do**
3. compute $\mathrm{Occ}(Q)$ from $\mathrm{Occ}(P)$
4. **if** $\mathrm{frq}(Q) \geq \sigma$ and $P = P(Q)$ **then call** Backtrack2 $(Q, \mathrm{Occ}(Q), \sigma)$
5. **end for**

The time complexity of BackTrack2 is $O(s(T + T'' + T'|\mathcal{D}|)\mu)$ where T'' is the time to compute the parent of a pattern. The heaviest part of $T'|\mathcal{D}|$ is usually reduced by down project. The algorithm will be efficient if all patterns Q satisfying $P = P(Q)$ are efficiently enumerated. Such examples are sequences [12], trees [3], and motifs with wildcards [2].

Frequent Sequence Mining

$\mathcal{L}$ is composed of strings on alphabet Σ, and $a, b \in \mathcal{L}$ satisfy $a \preceq b$ iff a is a subsequence of b, i.e., a is obtained from b by deleting some letters. For a pattern P, $\mathrm{suc}(P)$ is the set of strings obtained by inserting a letter to P at some position.

We define the parent of P by the string obtained by removing the last letter from P. Then, the children of P is generated by appending a letter to the tail of P. Since $\preceq$ can be tested in linear time, BackTrack2 runs in $O(|\Sigma| \times ||\mathcal{D}||)$ time for each frequent sequence.

Frequent Ordered Tree Mining

$\mathcal{L}$ is composed of rooted trees such that each vertex has a label and an ordering of children. Such a tree is called a labeled ordered tree. $a, b \in \mathcal{L}$ satisfy $a \preceq b$ iff a is a subtree of b with correspondence keeping the children orders and vertex labels; a vertex of label "A" has to be mapped to a vertex having label "A," and children orders do not change. For a pattern P, $\mathrm{suc}(P)$ is the set of labeled ordered trees obtained by inserting a vertex as a leaf.

The rightmost path of an ordered tree is $\{v_1, \ldots, v_k\}$ where v_1 is the root, and v_i is the last child of v_{i-1}. We define the parent of P by that obtained by removing the rightmost leaf v_k. Then, the children of P is generated by appending a vertex with a label so that the vertex is the last child of a vertex in the rightmost path. Since $\preceq$ can be tested in linear time, BackTrack2 runs in $O(t|\Sigma| \times ||\mathcal{D}||)$ time for each frequent ordered tree, where t is the maximum height of the pattern tree.

Recommended Reading

1. Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo AI (1996) Fast discovery of association rules. In: Fayyad UM (ed) Advances in knowledge discovery and data mining. MIT, Menlo Park, pp 307–328
2. Arimura H, Uno T (2007) An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. J Comb Optim 13:243–262
3. Asai T, Arimura T, Uno T, Nakano S (2003) Discovering frequent substructures in large unordered trees. LNAI 2843:47–61
4. Avis D, Fukuda K (1996) Reverse search for enumeration. Discret Appl Math 65:21–46
5. Bayardo RJ Jr (1998) Efficiently mining long patterns from databases. SIGMOD Rec 27:85–93
6. Goethals B (2003) The FIMI repository. http://fimi.cs.helsinki.fi/
7. Grahne G, Zhu J (2003) Efficiently using prefix-trees in mining frequent itemsets. In: IEEE ICDM'03 workshop FIMI'03, Melbourne
8. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. ACM SIGMOD Rec 29:1–12
9. Uno T (1998) New approach for speeding up enumeration algorithms. In: Chwa K-Y, Ibarra OH (eds) Algorithms and computation. LNCS, vol 1533. Springer, Berlin/Heidelberg, pp 287–296
10. Uno T, Asai T, Uchida Y, Arimura H (2004) An efficient algorithm for enumerating closed patterns in transaction databases. In: Suzuki E, Arikawa S (eds) Discovery science. LNCS, vol 3245. Springer, Berlin/Heidelberg, pp 16–31
11. Uno T, Kiyomi M, Arimura H (2004) LCM ver.2: efficient mining algorithms for frequent/closed/maximal itemsets. In: IEEE ICDM'04 workshop FIMI'04, Brighton
12. Wang J, Han J (2004) BIDE: efficient mining of frequent closed sequences. In: ICDE'04, Boston, pp 79–90

Fully Dynamic All Pairs Shortest Paths

Giuseppe F. Italiano
Department of Computer and Systems Science, University of Rome, Rome, Italy
Department of Information and Computer Systems, University of Rome, Rome, Italy

Years and Authors of Summarized Original Work

2004; Demetrescu, Italiano

Problem Definition

The problem is concerned with efficiently maintaining information about all-pairs shortest paths in a dynamically changing graph. This problem has been investigated since the 1960s [17, 18, 20], and plays a crucial role in many applications, including network optimization and routing, traffic information systems, databases, compilers, garbage collection, interactive verification systems, robotics, dataflow analysis, and document formatting.

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is said to be *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only. In this entry, fully dynamic algorithms for maintaining shortest paths on general directed graphs are presented.

In the *fully dynamic All Pairs Shortest Path (APSP) problem* one wishes to maintain a directed graph $G = (V, E)$ with real-valued edge weights under an intermixed sequence of the following operations:

***Update*(x, y, w):** update the weight of edge (x, y) to the real value w; this includes as a special case both edge insertion (if the weight is set from $+\infty$ to $w < +\infty$) and edge deletion (if the weight is set to $w = +\infty$);

***Distance*(x, y):** output the shortest distance from x to y.

***Path*(x, y):** report a shortest path from x to y, if any.

More formally, the problem can be defined as follows.

Problem 1 (Fully Dynamic All-Pairs Shortest Paths)
INPUT: A weighted directed graph $G = (V, E)$, and a sequence σ of operations as defined above.
OUTPUT: A matrix D such entry $D[x, y]$ stores the distance from vertex x to vertex y throughout the sequence σ of operations.

Throughout this entry, m and n denotes respectively the number of edges and vertices in G.

Demetrescu and Italiano [3] proposed a new approach to dynamic path problems based on maintaining classes of paths characterized by local properties, i.e., properties that hold for all proper subpaths, even if they may not hold for the entire paths. They showed that this approach can play a crucial role in the dynamic maintenance of shortest paths.

Key Results

Theorem 1 *The fully dynamic shoretest path problem can be solved in* $O(n^2 \log^3 n)$ *amortized time per update during any intermixed sequence of operations. The space required is* $O(mn)$.

Using the same approach, Thorup [22] has shown how to slightly improve the running times:

Theorem 2 *The fully dynamic shoretest path problem can be solved in* $O(n^2(\log n +$

$\log^2(m/n)))$ *amortized time per update during any intermixed sequence of operations. The space required is* $O(mn)$.

Applications

Dynamic shortest paths find applications in many areas, including network optimization and routing, transportation networks, traffic information systems, databases, compilers, garbage collection, interactive verification systems, robotics, dataflow analysis, and document formatting.

Open Problems

The recent work on dynamic shortest paths has raised some new and perhaps intriguing questions. First, can one reduce the space usage for dynamic shortest paths to $O(n^2)$? Second, and perhaps more importantly, can one solve efficiently fully dynamic *single-source* reachability and shortest paths on general graphs? Finally, are there any general techniques for making increase-only algorithms fully dynamic? Similar techniques have been widely exploited in the case of fully dynamic algorithms on undirected graphs [11–13].

Experimental Results

A thorough empirical study of the algorithms described in this entry is carried out in [4].

Data Sets

Data sets are described in [4].

Cross-References

- Dynamic Trees
- Fully Dynamic Connectivity
- Fully Dynamic Higher Connectivity
- Fully Dynamic Higher Connectivity for Planar Graphs
- Fully Dynamic Minimum Spanning Trees
- Fully Dynamic Planarity Testing
- Fully Dynamic Transitive Closure

Recommended Reading

1. Ausiello G, Italiano GF, Marchetti-Spaccamela A, Nanni U (1991) Incremental algorithms for minimal length paths. J Algorithm 12(4):615–638
2. Demetrescu C (2001) Fully dynamic algorithms for path problems on directed graphs. PhD thesis, Department of Computer and Systems Science, University of Rome "La Sapienza", Rome
3. Demetrescu C, Italiano GF (2004) A new approach to dynamic all pairs shortest paths. J Assoc Comput Mach 51(6):968–992
4. Demetrescu C, Italiano GF (2006) Experimental analysis of dynamic all pairs shortest path algorithms. ACM Trans Algorithms 2(4):578–601
5. Demetrescu C, Italiano GF (2005) Trade-offs for fully dynamic reachability on dags: breaking through the O(n2) barrier. J Assoc Comput Mach 52(2):147–156
6. Demetrescu C, Italiano GF (2006) Fully dynamic all pairs shortest paths with real edge weights. J Comput Syst Sci 72(5):813–837
7. Even S, Gazit H (1985) Updating distances in dynamic graphs. Method Oper Res 49:371–387
8. Frigioni D, Marchetti-Spaccamela A, Nanni U (1998) Semi-dynamic algorithms for maintaining single source shortest paths trees. Algorithmica 22(3):250–274
9. Frigioni D, Marchetti-Spaccamela A, Nanni U (2000) Fully dynamic algorithms for maintaining shortest paths trees. J Algorithm 34:351–381
10. Henzinger M, King V (1995) Fully dynamic biconnectivity and transitive closure. In: Proceedings of the 36th IEEE symposium on foundations of computer science (FOCS'95), Los Alamos. IEEE Computer Society, pp 664–672
11. Henzinger M, King V (2001) Maintaining minimum spanning forests in dynamic graphs. SIAM J Comput 31(2):364–374
12. Henzinger MR, King V (1999) Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J ACM 46(4):502–516
13. Holm J, de Lichtenberg K, Thorup M (2001) Polylogarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J ACM 48:723–760
14. King V (1999) Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In: Proceedings of the 40th IEEE symposium on foundations of computer science (FOCS'99), Los Alamos. IEEE Computer Society, pp 81–99

15. King V, Sagert G (2002) A fully dynamic algorithm for maintaining the transitive closure. J Comput Syst Sci 65(1):150–167
16. King V, Thorup M (2001) A space saving trick for directed dynamic transitive closure and shortest path algorithms. In: Proceedings of the 7th annual international computing and combinatorics conference (COCOON). LNCS, vol 2108. Springer, Berlin, pp 268–277
17. Loubal P (1967) A network evaluation procedure. Highw Res Rec 205:96–109
18. Murchland J (1967) The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph. Technical report, LBS-TNT-26, London Business School, Transport Network Theory Unit, London
19. Ramalingam G, Reps T (1996) An incremental algorithm for a generalization of the shortest path problem. J Algorithm 21:267–305
20. Rodionov V (1968) The parametric problem of shortest distances. USSR Comput Math Math Phys 8(5):336–343
21. Rohnert H (1985) A dynamization of the all-pairs least cost problem. In: Proceedings of the 2nd annual symposium on theoretical aspects of computer science, (STACS'85). LNCS, vol 182. Springer, Berlin, pp 279–286
22. Thorup M (2004) Fully-dynamic all-pairs shortest paths: faster and allowing negative cycles. In: Proceedings of the 9th Scandinavian workshop on algorithm theory (SWAT'04), Humlebaek. Springer, Berlin, pp 384–396
23. Thorup M (2005) Worst-case update times for fully-dynamic all-pairs shortest paths. In: Proceedings of the 37th ACM symposium on theory of computing (STOC 2005). ACM, New York

Fully Dynamic Connectivity

Valerie King
Department of Computer Science, University of Victoria, Victoria, BC, Canada

Keywords

Fully dynamic graph algorithm for maintaining connectivity; Incremental algorithms for graphs

Years and Authors of Summarized Original Work

2001; Holm, de Lichtenberg, Thorup

Problem Definition

Design a data structure for an undirected graph with a fixed set of nodes which can process queries of the form "Are nodes i and j connected?" and updates of the form "Insert edge $\{i,j\}$"; "Delete edge $\{i,j\}$." The goal is to minimize update and query times, over the worst-case sequence of queries and updates. Algorithms to solve this problem are called "fully dynamic" as opposed to "partially dynamic" since both insertions and deletions are allowed.

Key Results

Holm et al. [4] gave the first deterministic fully dynamic graph algorithm for maintaining connectivity in an undirected graph with polylogarithmic amortized time per operation, specifically, $O(\log^2 n)$ amortized cost per update operation and $O(\log n / \log\log n)$ worst-case per query, where n is the number of nodes. The basic technique is extended to maintain minimum spanning trees in $O(\log^4 n)$ amortized cost per update operation and 2-edge connectivity and biconnectivity in $O(\log^5 n)$ amortized time per operation.

The algorithm relies on a simple novel technique for maintaining a spanning forest in a graph which enables efficient search for a replacement edge when a tree edge is deleted. This technique ensures that each nontree edge is examined no more than $\log_2 n$ times. The algorithm relies on previously known tree data structures, such as top trees or ET-trees to store and quickly retrieve information about the spanning trees and the nontree edges incident to them.

Algorithms to achieve a query time $O(\log n / \log\log\log n)$ and expected amortized update time $O(\log n(\log\log n)^3)$ for connectivity and $O(\log^3 n \log\log n)$ expected amortized update time for 2-edge and biconnectivity were given in [6]. Lower bounds showing a continuum of tradeoffs for connectivity between query and update times in the cell probe model which match the known upper bounds were proved in [5]. Specifically, if t_u and t_q are the amortized update

and query time, respectively, then $t_q \cdot \log(t_u/t_q) = \Omega(\log n)$ and $t_u \cdot \log(t_q/t_u) = \Omega(\log n)$.

A previously known, somewhat different, randomized method for computing dynamic connectivity with $O(\log^3 n)$ amortized expected update time can be found in [2], improved to $O(\log^2 n)$ in [3]. A method which minimizes worst-case rather than amortized update time is given in [1] $O(\sqrt{n})$ time per update for connectivity as well as 2-edge connectivity and bipartiteness.

Open Problems

Can the worst-case update time be reduced to $O(n^{1/2})$, with polylogarithmic query time?

Can the lower bounds on the trade-offs in [6] be matched for all possible query costs?

Applications

Dynamic connectivity has been used as a subroutine for several static graph algorithms, such as the maximum flow problem in a static graph [7], and for speeding up numerical studies of the Potts spin model.

URL to Code

See http://www.mpi-sb.mpg.de/LEDA/friends/dyngraph.html for software which implements the algorithm in [2] and other older methods.

Cross-References

- Fully Dynamic All Pairs Shortest Paths
- Fully Dynamic Transitive Closure

Recommended Reading

1. Eppstein D, Galil Z, Italiano GF, Nissenzweig A (1997) Sparsification-a technique for speeding up dynamic graph algorithms. J ACM 44(5):669–696.1
2. Henzinger MR, King V (1999) Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J ACM 46(4):502–536. (Presented at ACM STOC 1995)
3. Henzinger MR, Thorup M (1997) Sampling to provide or to bound: with applications to fully dynamic graph algorithms. Random Struct Algorithms 11(4):369–379. (Presented at ICALP 1996)
4. Holm J, De Lichtenberg K, Thorup M (2001) Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-Edge, and biconnectivity. J ACM 48(4):723–760. (Presented at ACM STOC 1998)
5. Iyer R, Karger D, Rahul H, Thorup M (2001) An experimental study of poly-logarithmic fully-dynamic connectivity algorithms. J Exp Algorithmics 6(4). (Presented at ALENEX 2000)
6. Pătraşcu M, Demaine E (2006) Logarithmic lower bounds in the cell-probe model. SIAM J Comput 35(4):932–963. (Presented at ACM STOC 2004)
7. Thorup M (2000) Near-optimal fully-dynamic graph connectivity. In: Proceedings of the 32nd ACM symposium on theory of computing, Portland. ACM STOC, pp 343–350
8. Thorup M (2000) Dynamic Graph Algorithms with Applications. In: Halldórsson MM (ed) 7th Scandinavian workshop on algorithm theory (SWAT), Norway, 5–7 July 2000, pp 1–9
9. Zaroliagis CD (2002) Implementations and experimental studies of dynamic graph algorithms. In: Experimental algorithmics, Dagstuhl seminar, Sept 2000. Lecture notes in computer science, vol 2547. Springer. Journal article: J Exp Algorithmics 229–278 (2000)

Fully Dynamic Connectivity: Upper and Lower Bounds

Giuseppe F. Italiano
Department of Computer and Systems Science, University of Rome, Rome, Italy
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords

Dynamic connected components; Dynamic spanning forests

Years and Authors of Summarized Original Work

2000; Thorup

Problem Definition

The problem is concerned with efficiently maintaining information about connectivity in a dynamically changing graph. A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is said to be *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

In the fully dynamic connectivity problem, one wishes to maintain an undirected graph $G = (V, E)$ under an intermixed sequence of the following operations:

***Connected*(u, v):** Return *true* if vertices u and v are in the same connected component of the graph. Return *false* otherwise.

***Insert*(x, y):** Insert a new edge between the two vertices x and y.

***Delete*(x, y):** Delete the edge between the two vertices x and y.

Key Results

In this section, a high level description of the algorithm for the fully dynamic connectivity problem in undirected graphs described in [11] is presented: the algorithm, due to Holm, de Lichtenberg and Thorup, answers connectivity queries in $O(\log n / \log \log n)$ worst-case running time while supporting edge insertions and deletions in $O(\log^2 n)$ amortized time.

The algorithm maintains a spanning forest F of the dynamically changing graph G. Edges in F are referred to as *tree edges*. Let e be a tree edge of forest F, and let T be the tree of F containing it. When e is deleted, the two trees T_1 and T_2 obtained from T after the deletion of e can be reconnected if and only if there is a non-tree edge in G with one endpoint in T_1 and the other endpoint in T_2. Such an edge is called a *replacement edge* for e. In other words, if there is a replacement edge for e, T is reconnected via this replacement edge; otherwise, the deletion of e creates a new connected component in G.

To accommodate systematic search for replacement edges, the algorithm associates to each edge e a level $\ell(e)$ and, based on edge levels, maintains a set of sub-forests of the spanning forest F: for each level i, forest F_i is the sub-forest induced by tree edges of level $\geq i$. Denoting by L denotes the maximum edge level, it follows that:

$$F = F_0 \supseteq F_1 \supseteq F_2 \supseteq \cdots \supseteq F_L \,.$$

Initially, all edges have level 0; levels are then progressively increased, but never decreased. The changes of edge levels are accomplished so as to maintain the following invariants, which obviously hold at the beginning.

Invariant (1): F is a maximum spanning forest of G if edge levels are interpreted as weights.

Invariant (2): The number of nodes in each tree of F_i is at most $n/2^i$.

Invariant (1) should be interpreted as follows. Let (u, v) be a non-tree edge of level $\ell(u, v)$ and let $u \cdots v$ be the unique path between u and v in F (such a path exists since F is a spanning forest of G). Let e be any edge in $u \cdots v$ and let $\ell(e)$ be its level. Due to (1), $\ell(e) \geq \ell(u, v)$. Since this holds for each edge in the path, and by construction $F_{\ell(u,v)}$ contains all the tree edges of level $\geq \ell(u, v)$, the entire path is contained in $F_{\ell(u,v)}$, i.e., u and v are connected in $F_{\ell(u,v)}$.

Invariant (2) implies that the maximum number of levels is $L \leq \lfloor \log_2 n \rfloor$.

Note that when a new edge is inserted, it is given level 0. Its level can be then increased at most $\lfloor \log_2 n \rfloor$ times as a consequence of edge deletions. When a tree edge $e = (v, w)$ of level $\ell(e)$ is deleted, the algorithm looks for a replacement edge at the highest possible level, if any.

Due to invariant (1), such a replacement edge has level $\ell \leq \ell(e)$. Hence, a replacement subroutine `Replace`$((u, w),\ell(e))$ is called with parameters e and $\ell(e)$. The operations performed by this subroutine are now sketched.

`Replace`$((u, w), \ell)$ finds a replacement edge of the highest level $\leq \ell$, if any. If such a replacement does not exist in level ℓ, there are two cases: if $\ell > 0$, the algorithm recurses on level $\ell - 1$; otherwise, $\ell = 0$, and the deletion of (v, w) disconnects v and w in G.

During the search at level ℓ, suitably chosen tree and non-tree edges may be promoted at higher levels as follows. Let T_v and T_w be the trees of forest F_ℓ obtained after deleting (v, w) and let, w.l.o.g., T_v be smaller than T_w. Then T_v contains at most $n/2^{\ell+1}$ vertices, since $T_v \cup T_w \cup \{(v, w)\}$ was a tree at level ℓ and due to invariant (2). Thus, edges in T_v of level ℓ can be promoted at level $\ell + 1$ by maintaining the invariants. Non-tree edges incident to T_v are finally visited one by one: if an edge does connect T_v and T_w, a replacement edge has been found and the search stops, otherwise its level is increased by 1.

Trees of each forest are maintained so that the basic operations needed to implement edge insertions and deletions can be supported in $O(\log n)$ time. There are few variants of basic data structures that can accomplish this task, and one could use the Euler Tour trees (in short ET-tree), first introduced in [17], for this purpose.

In addition to inserting and deleting edges from a forest, ET-trees must also support operations such as finding the tree of a forest that contains a given vertex, computing the size of a tree, and, more importantly, finding tree edges of level ℓ in T_v and non-tree edges of level ℓ incident to T_v. This can be done by augmenting the ET-trees with a constant amount of information per node: the interested reader is referred to [11] for details.

Using an amortization argument based on level changes, the claimed $O(\log^2 n)$ bound on the update time can be proved. Namely, inserting an edge costs $O(\log n)$, as well as increasing its level. Since this can happen $O(\log n)$ times, the total amortized insertion cost, inclusive of level increases, is $O(\log^2 n)$. With respect to edge deletions, cutting and linking $O(\log n)$ forest has a total cost $O(\log^2 n)$; moreover, there are $O(\log n)$ recursive calls to `Replace`, each of cost $O(\log n)$ plus the cost amortized over level increases. The ET-trees over $F_0 = F$ allows it to answer connectivity queries in $O(\log n)$ worst-case time. As shown in [11], this can be reduced to $O(\log n / \log\log n)$ by using a $\Theta(\log n)$-ary version of ET-trees.

Theorem 1 *A dynamic graph G with n vertices can be maintained upon insertions and deletions of edges using $O(\log^2 n)$ amortized time per update and answering connectivity queries in $O(\log n / \log\log n)$ worst-case running time.*

Later on, Thorup [18] gave another data structure which achieves slightly different time bounds:

Theorem 2 *A dynamic graph G with n vertices can be maintained upon insertions and deletions of edges using $O(\log n \cdot (\log\log n)^3)$ amortized time per update and answering connectivity queries in $O(\log n / \log\log\log n)$ time.*

The bounds given in Theorems 1 and 2 are not directly comparable, because each sacrifices the running time of one operation (either query or update) in order to improve the other.

The best known lower bound for the dynamic connectivity problem holds in the bit-probe model of computation and is due to Pătraşcu and Tarniţă [16]. The bit-probe model is an instantiation of the cell-probe model with one-bit cells. In this model, memory is organized in cells, and the algorithms may read or write a cell in constant time. The number of cell probes is taken as the measure of complexity. For formal definitions of this model, the interested reader is referred to [13].

Theorem 3 *Consider a bit-probe implementation for dynamic connectivity, in which updates take expected amortized time t_u, and queries take expected time t_q. Then, in the average case of an input distribution, $t_u = \Omega\left(log^2 n / log^2(t_u + t_q)\right)$. In particular*

$$\max\{t_u, t_q\} = \Omega\left(\left(\frac{\log n}{\log\log n}\right)^2\right).$$

In the bit-probe model, the best upper bound per operation is given by the algorithm of Theorem 2, namely it is $O(\log^2 n/\log\log\log n)$. Consequently, the gap between upper and lower bound appears to be limited essentially to doubly logarithmic factors only.

Applications

Dynamic graph connectivity appears as a basic subproblem of many other important problems, such as the dynamic maintenance of minimum spanning trees and dynamic edge and vertex connectivity problems. Furthermore, there are several applications of dynamic graph connectivity in other disciplines, ranging from Computational Biology, where dynamic graph connectivity proved to be useful for the dynamic maintenance of protein molecular surfaces as the molecules undergo conformational changes [6], to Image Processing, when one is interested in maintaining the connected components of a bitmap image [3].

Open Problems

The work on dynamic connectivity raises some open and perhaps intruiguing questions. The first natural open problem is whether the gap between upper and lower bounds can be closed. Note that the lower bound of Theorem 3 seems to imply that different trade-offs between queries and updates could be possible: can we design a data structure with $o(\log n)$ time per update and $O(\text{poly}(\log n))$ per query? This would be particulary interesting in applications where the total number of queries is substantially larger than the number of updates.

Finally, is it possible to design an algorithm with matching $O(\log n)$ update and query bounds for general graphs? Note that this is possible in the special case of plane graphs [5].

Experimental Results

A thorough empirical study of dynamic connectivity algorithms has been carried out in [1, 12].

Data Sets

Data sets are described in [1, 12].

Cross-References

- Dynamic Trees
- Fully Dynamic All Pairs Shortest Paths
- Fully Dynamic Higher Connectivity
- Fully Dynamic Higher Connectivity for Planar Graphs
- Fully Dynamic Minimum Spanning Trees
- Fully Dynamic Planarity Testing
- Fully Dynamic Transitive Closure

Recommended Reading

1. Alberts D, Cattaneo G, Italiano GF (1997) Anempirical study of dynamic graph algorithms. ACM J Exp Algorithms 2
2. Beame P, Fich FE (2002) Optimal bounds for the predecessor problem and related problems. J Comput Syst Sci 65(1):38–72
3. Eppstein D (1997) Dynamic connectivity in digital images. Inf Process Lett 62(3):121–126
4. Eppstein D, Galil Z, Italiano GF, Nissenzweig A (1997) Sparsification – a technique for speeding up dynamic graph algorithms. J Assoc Comput Mach 44(5):669–696
5. Eppstein D, Italiano GF, Tamassia R, Tarjan RE, Westbrook J, Yung M (1992) Maintenance of a minimum spanning forest in a dynamic plane graph. J Algorithms 13:33–54
6. Eyal E, Halperin D (2005) Improved maintenance of molecular surfacesusing dynamic graph connectivity. In: Proceedings of the 5th international workshop on algorithms in bioinformatics (WABI 2005), Mallorca, pp 401–413
7. Frederickson GN (1985) Data structures for on-line updating of minimum spanning trees. SIAM J Comput 14:781–798
8. Frederickson GN (1991) Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. In: Proceedings of the 32nd symposium on foundations of computer science, pp 632–641

9. Henzinger MR, Fredman ML (1998) Lower bounds for fully dynamic connectivity problems in graphs. Algorithmica 22(3):351–362
10. Henzinger MR, King V (1999) Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J ACM 46(4):502–516
11. Holm J, de Lichtenberg K, Thorup M (2001) Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J ACM 48:723–760
12. Iyer R, Karger D, Rahul H, Thorup M (2001) An experimental study of polylogarithmic, fully dynamic, connectivity algorithms. ACM J Exp Algorithmics 6
13. Miltersen PB (1999) Cell probe complexity – a survey. In: 19th conference on the foundations of software technology and theoretical computer science (FSTTCS). Advances in Data Structures Workshop
14. Miltersen PB, Subramanian S, Vitter JS, Tamassia R (1994) Complexity models for incremental computation. In: Ausiello G, Italiano GF (eds) Special issue on dynamic and on-line algorithms. Theor Comput Sci 130(1):203–236
15. Pătraşcu M, Demain ED (2004) Lower bounds for dynamic connectivity. In: Proceedings of the 36th ACM symposium on theory of computing (STOC), pp 546–553
16. Pätrascu M, Tarnita C (2007) On dynamic bit-probe complexity. Theor Comput Sci Spec Issue ICALP'05 380:127–142; In: Italiano GF, Palamidessi C (eds) A preliminary version in proceedings of the 32nd international colloquium on automata, languages and programming (ICALP'05), pp 969–981
17. Tarjan RE, Vishkin U (1985) An efficient parallel biconnectivity algorithm. SIAM J Comput 14:862–874
18. Thorup M (2000) Near-optimal fully-dynamic graph connectivity. In: Proceedings of the 32nd ACM symposium on theory of computing (STOC), pp 343–350

Fully Dynamic Higher Connectivity

Giuseppe F. Italiano
Department of Computer and Systems Science, University of Rome, Rome, Italy
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords

Fully dynamic edge connectivity; Fully dynamic vertex connectivity

Years and Authors of Summarized Original Work

1997; Eppstein, Galil, Italiano, Nissenzweig

Problem Definition

The problem is concerned with efficiently maintaining information about edge and vertex connectivity in a dynamically changing graph. Before defining formally the problems, a few preliminary definitions follow.

Given an undirected graph $G = (V, E)$, and an integer $k \geq 2$, a pair of vertices $\langle u, v \rangle$ is said to be *k-edge-connected* if the removal of any $(k-1)$ edges in G leaves u and v connected. It is not difficult to see that this is an equivalence relationship: the vertices of a graph G are partitioned by this relationship into equivalence classes called *k-edge-connected components*. G is said to be *k-edge-connected* if the removal of any $(k-1)$ edges leaves G connected. As a result of these definitions, G is k-edge-connected if and only if any two vertices of G are k-edge-connected. An edge set $E' \subseteq E$ is an *edge-cut for vertices x and y* if the removal of all the edges in E' disconnects G into two graphs, one containing x and the other containing y. An edge set $E' \subseteq E$ is an *edge-cut for G* if the removal of all the edges in E' disconnects G into two graphs. An edge-cut E' for G (for x and y, respectively) is *minimal* if removing any edge from E' reconnects G (for x and y, respectively). The cardinality of an edge-cut E', denoted by $|E'|$, is given by the number of edges in E'. An edge-cut E' for G (for x and y, respectively) is said to be a *minimum cardinality edge-cut* or in short a *connectivity edge-cut* if there is no other edge-cut E'' for G (for x and y respectively) such that $|E''| < |E'|$. Connectivity edge-cuts are of course minimal edge-cuts. Note that G is k-edge-connected if and only if a connectivity edge-cut for G contains at least k edges, and vertices x and y are k-edge-connected if and only if a connectivity edge-cut for x and y contains at least k edges. A connectivity edge-cut of cardinality 1 is called a *bridge*.

The following theorem due to Ford and Fulkerson, and Elias, Feinstein and Shannon (see [7]) gives another characterization of k-edge connectivity.

Theorem 1 (Ford and Fulkerson, Elias, Feinstein and Shannon) *Given a graph G and two vertices x and y in G, x and y are k-edge-connected if and only if there are at least k edge-disjoint paths between x and y.*

In a similar fashion, a vertex set $V' \subseteq V - \{x, y\}$ is said to be a *vertex-cut* for vertices x and y if the removal of all the vertices in V' disconnects x and y. $V' \subset V$ is a *vertex-cut* for vertices G if the removal of all the vertices in V' disconnects G.

The cardinality of a vertex-cut V', denoted by $|V'|$, is given by the number of vertices in V'. A vertex-cut V' for x and y is said to be a *minimum cardinality vertex-cut* or in short a *connectivity vertex-cut* if there is no other vertex-cut V'' for x and y such that $|V''| < |V'|$. Then x and y are k-vertex-connected if and only if a connectivity vertex-cut for x and y contains at least k vertices. A graph G is said to be *k-vertex-connected* if all its pairs of vertices are k-vertex-connected. A connectivity vertex-cut of cardinality 1 is called an *articulation point*, while a connectivity vertex-cut of cardinality 2 is called a *separation pair*. Note that for vertex connectivity it is no longer true that the removal of a connectivity vertex-cut splits G into two sets of vertices.

The following theorem due to Menger (see [7]) gives another characterization of k-vertex connectivity.

Theorem 2 (Menger) *Given a graph G and two vertices x and y in G, x and y are k-vertex-connected if and only if there are at least k vertex-disjoint paths between x and y.*

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

In the *fully dynamic k-edge connectivity problem* one wishes to maintain an undirected graph $G = (V, E)$ under an intermixed sequence of the following operations:

- *k-EdgeConnected(u, v):* Return *true* if vertices u and v are in the same k-edge-connected component. Return *false* otherwise.
- *Insert(x, y):* Insert a new edge between the two vertices x and y.
- *Delete(x, y):* Delete the edge between the two vertices x and y.

In the *fully dynamic k-vertex connectivity problem* one wishes to maintain an undirected graph $G = (V, E)$ under an intermixed sequence of the following operations:

- *k-VertexConnected(u, v):* Return *true* if vertices u and v are k-vertex-connected. Return *false* otherwise.
- *Insert(x, y):* Insert a new edge between the two vertices x and y.
- *Delete(x, y):* Delete the edge between the two vertices x and y.

Key Results

To the best knowledge of the author, the most efficient fully dynamic algorithms for k-edge and k-vertex connectivity were proposed in [3, 12]. Their running times are characterized by the following theorems.

Theorem 3 *The fully dynamic k-edge connectivity problem can be solved in:*

1. $O(\log^4 n)$ *time per update and* $O(\log^3 n)$ *time per query, for* $k = 2$

2. $O(n^{2/3})$ *time per update and query, for* $k = 3$
3. $O(n\alpha(n))$ *time per update and query, for* $k = 4$
4. $O(n \log n)$ *time per update and query, for* $k \geq 5$.

Theorem 4 *The fully dynamic k-vertex connectivity problem can be solved in:*

1. $O(\log^4 n)$ *time per update and* $O(\log^3 n)$ *time per query, for* $k = 2$
2. $O(n)$ *time per update and query, for* $k = 3$
3. $O(n\alpha(n))$ *time per update and query, for* $k = 4$.

Applications

Vertex and edge connectivity problems arise often in issues related to network reliability and survivability. In computer networks, the vertex connectivity of the underlying graph is related to the smallest number of nodes that might fail before disconnecting the whole network. Similarly, the edge connectivity is related to the smallest number of links that might fail before disconnecting the entire network. Analogously, if two nodes are k-vertex-connected then they can remain connected even after the failure of up to $(k-1)$ other nodes, and if they are k-edge-connected then they can survive the failure of up to $(k-1)$ links. It is important to investigate the dynamic versions of those problems in contexts where the networks are dynamically evolving, say, when links may go up and down because of failures and repairs.

Open Problems

The work of Eppstein et al. [3] and Holm et al. [12] raises some intriguing questions. First, while efficient dynamic algorithms for k-edge connectivity are known for general k, no efficient fully dynamic k-vertex connectivity is known for $k \geq 5$. To the best of the author's knowledge, in this case even no static algorithm is known. Second, fully dynamic 2-edge and 2-vertex connectivity can be solved in polylogarithmic time per update, while the best known update bounds for higher edge and vertex connectivity are polynomial: Can this gap be reduced, i.e., can one design polylogarithnmic algorithms for fully dynamic 3-edge and 3-vertex connectivity?

Cross-References

- Dynamic Trees
- Fully Dynamic All Pairs Shortest Paths
- Fully Dynamic Connectivity
- Fully Dynamic Higher Connectivity for Planar Graphs
- Fully Dynamic Minimum Spanning Trees
- Fully Dynamic Planarity Testing
- Fully Dynamic Transitive Closure

Recommended Reading

1. Dinitz EA (1993) Maintaining the 4-edge-connected components of a graph on-line. In: Proceedings of the 2nd Israel symposium on theory of computing and systems, Natanya, pp 88–99
2. Dinitz EA, Karzanov AV, Lomonosov MV (1990) On the structure of the system of minimal edge cuts in a graph. In: Fridman AA (ed) Studies in discrete optimization. Nauka, Moscow, pp 290–306 (in Russian)
3. Eppstein D, Galil Z, Italiano GF, Nissenzweig A (1997) Sparsification – a technique for speeding up dynamic graph algorithms. J Assoc Comput Mach 44(5):669–696
4. Frederickson GN (1997) Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. SIAM J Comput 26(2):484–538
5. Galil Z, Italiano GF (1992) Fully dynamic algorithms for 2-edge connectivity. SIAM J Comput 21:1047–1069
6. Galil Z, Italiano GF (1993) Maintaining the 3-edge-connected components of a graph on-line. SIAM J Comput 22:11–28
7. Harary F (1969) Graph theory. Addison-Wesley, Reading
8. Henzinger MR (1995) Fully dynamic biconnectivity in graphs. Algorithmica 13(6):503–538
9. Henzinger MR (2000) Improved data structures for fully dynamic biconnectivity. SIAM J Comput 29(6):1761–1815
10. Henzinger M, King V (1995) Fully dynamic biconnectivity and transitive closure. In: Proceedings of the 36th IEEE symposium on foundations of computer science (FOCS'95), Milwaukee, pp 664–672

F

11. Henzinger MR, King V (1999) Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J ACM 46(4):502–516
12. Holm J, de Lichtenberg K, Thorup M (2001) Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J ACM 48:723–760
13. Karzanov AV, Timofeev EA (1986) Efficient algorithm for finding all minimal edge cuts of a nonoriented graph. Cybernetics 22:156–162
14. La Poutré JA (1992) Maintenance of triconnected components of graphs. In: La Poutré JA (ed) Proceedings of the 19th international colloquium on automata, languages and programming. Lecture notes in computer science, vol 623. Springer, Berlin, pp 354–365
15. La Poutré JA (2000) Maintenance of 2- and 3-edge-connected components of graphs II. SIAM J Comput 29(5):1521–1549
16. La Poutré JA, van Leeuwen J, Overmars MH (1993) Maintenance of 2- and 3-connected components of graphs, part I: 2- and 3-edge-connected components. Discr Math 114:329–359
17. La Poutré JA, Westbrook J (1994) Dynamic two-connectivity with backtracking. In: Proceedings of the 5th ACM-SIAM symposium on discrete algorithms, Arlington, pp 204–212
18. Westbrook J, Tarjan RE (1992) Maintaining bridge-connected and biconnected components on-line. Algorithmica 7:433–464

Fully Dynamic Higher Connectivity for Planar Graphs

Giuseppe F. Italiano
Department of Computer and Systems Science, University of Rome, Rome, Italy
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords

Fully dynamic edge connectivity; Fully dynamic vertex connectivity

Years and Authors of Summarized Original Work

1998; Eppstein, Galil, Italiano, Spencer

Problem Definition

In this entry, the problem of maintaining a dynamic planar graph subject to edge insertions and edge deletions that preserve planarity but that can change the embedding is considered. In particular, in this problem one is concerned with the problem of efficiently maintaining information about edge and vertex connectivity in such a dynamically changing planar graph. The algorithms to solve this problem must handle insertions that keep the graph planar without regard to any particular embedding of the graph. The interested reader is referred to the chapter ▸ Fully Dynamic Planarity Testing of this encyclopedia for algorithms to learn how to check efficiently whether a graph subject to edge insertions and deletions remains planar (without regard to any particular embedding).

Before defining formally the problems considered here, a few preliminary definitions follow.

Given an undirected graph $G = (V, E)$, and an integer $k \geq 2$, a pair of vertices $\langle u, v \rangle$ is said to be *k-edge-connected* if the removal of any $(k-1)$ edges in G leaves u and v connected. It is not difficult to see that this is an equivalence relationship: the vertices of a graph G are partitioned by this relationship into equivalence classes called *k-edge-connected components*. G is said to be *k-edge-connected* if the removal of any $(k-1)$ edges leaves G connected. As a result of these definitions, G is k-edge-connected if and only if any two vertices of G are k-edge-connected. An edge set $E' \subseteq E$ is an *edge-cut for vertices x and y* if the removal of all the edges in E' disconnects G into two graphs, one containing x and the other containing y. An edge set $E' \subseteq E$ is an *edge-cut for G* if the removal of all the edges in E' disconnects G into two graphs. An edge-cut E' for G (for x and y, respectively) is *minimal* if removing any edge from E' reconnects G (for x and y, respectively). The cardinality of an edge-cut E', denoted by $|E'|$, is given by the number of edges in E'. An edge-cut E' for G (for x and y, respectively) is said to be a *minimum cardinality edge-cut* or in short a *connectivity edge-cut* if there is no other edge-cut E'' for G (for x and y, respectively) such that $|E''| < |E'|$. Connec-

tivity edge-cuts are of course minimal edge-cuts. Note that G is k-edge-connected if and only if a connectivity edge-cut for G contains at least k edges, and vertices x and y are k-edge-connected if and only if a connectivity edge-cut for x and y contains at least k edges. A connectivity edge-cut of cardinality 1 is called a *bridge*.

In a similar fashion, a vertex set $V' \subseteq V - \{x, y\}$ is said to be a *vertex-cut* for vertices x and y if the removal of all the vertices in V' disconnects x and y. $V' \subset V$ is a *vertex-cut* for vertices G if the removal of all the vertices in V' disconnects G.

The cardinality of a vertex-cut V', denoted by $|V'|$, is given by the number of vertices in V'. A vertex-cut V' for x and y is said to be a *minimum cardinality vertex-cut* or in short a *connectivity vertex-cut* if there is no other vertex-cut V'' for x and y such that $|V''| < |V'|$. Then x and y are k-vertex-connected if and only if a connectivity vertex-cut for x and y contains at least k vertices. A graph G is said to be *k-vertex-connected* if all its pairs of vertices are k-vertex-connected. A connectivity vertex-cut of cardinality 1 is called an *articulation point*, while a connectivity vertex-cut of cardinality 2 is called a *separation pair*. Note that for vertex connectivity it is no longer true that the removal of a connectivity vertex-cut splits G into two sets of vertices.

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

In the *fully dynamic k-edge connectivity problem for a planar graph* one wishes to maintain an undirected planar graph $G = (V, E)$ under an intermixed sequence of edge insertions, edge deletions and queries about the k-edge connectivity of the underlying planar graph. Similarly, in the *fully dynamic k-vertex connectivity problem for a planar graph* one wishes to maintain an undirected planar graph $G = (V, E)$ under an intermixed sequence of edge insertions, edge deletions and queries about the k-vertex connectivity of the underlying planar graph.

Key Results

The algorithms in [2, 3] solve efficiently the above problems for small values of k:

Theorem 1 *One can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow queries that test the 2-edge connectivity of the graph, or test whether two vertices belong to the same 2-edge-connected component, in $O(\log n)$ amortized time per insertion or query, and $O(\log^2 n)$ per deletion.*

Theorem 2 *One can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow testing of the 3-edge and 4-edge connectivity of the graph in $O(n^{1/2})$ time per update, or testing of whether two vertices are 3- or 4-edge-connected, in $O(n^{1/2})$ time per update or query.*

Theorem 3 *One can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow queries that test the 3-vertex connectivity of the graph, or test whether two vertices belong to the same 3-vertex-connected component, in $O(n^{1/2})$ amortized time per update or query.*

Note that these theorems improve on the bounds known for the same problems on general graphs, reported in the chapter ▸ Fully Dynamic Higher Connectivity

Applications

The interest reader is referred to the chapter ▸ Fully Dynamic Higher Connectivity for applications of dynamic edge and vertex connectivity.

The case of planar graphs is especially important, as these graphs arise frequently in applications.

Open Problems

A number of problems related to the work of Eppstein et al. [2, 3] remain open. First, can the running times per operation be improved? Second, as in the case of general graphs, also for planar graphs fully dynamic 2-edge connectivity can be solved in polylogarithmic time per update, while the best known update bounds for higher edge and vertex connectivity are polynomial: Can this gap be reduced, i.e., can one design polylogarithnmic algorithms at least for fully dynamic 3-edge and 3-vertex connectivity? Third, in the special case of planar graphs can one solve fully dynamic k-vertex connectivity for general k?

Cross-References

- ▸ Dynamic Trees
- ▸ Fully Dynamic All Pairs Shortest Paths
- ▸ Fully Dynamic Connectivity
- ▸ Fully Dynamic Higher Connectivity
- ▸ Fully Dynamic Minimum Spanning Trees
- ▸ Fully Dynamic Planarity Testing
- ▸ Fully Dynamic Transitive Closure

Recommended Reading

1. Galil Z, Italiano GF, Sarnak N (1999) Fully dynamic planarity testing with applications. J ACM 48:28–91
2. Eppstein D, Galil Z, Italiano GF, Spencer TH (1996) Separator based sparsification I: planarity testing and minimum spanning trees. J Comput Syst Sci Spec Issue STOC 93 52(1):3–27
3. Eppstein D, Galil Z, Italiano GF, Spencer TH (1999) Separator based sparsification II: edge and vertex connectivity. SIAM J Comput 28:341–381
4. Giammarresi D, Italiano GF (1996) Decremental 2- and 3-connectivity on planar graphs. Algorithmica 16(3):263–287
5. Hershberger J, Rauch M, Suri S (1994) Data structures for two-edge connectivity in planar graphs. Theor Comput Sci 130(1):139–161

Fully Dynamic Minimum Spanning Trees

Giuseppe F. Italiano
Department of Computer and Systems Science, University of Rome, Rome, Italy
Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords

Dynamic minimum spanning forests

Years and Authors of Summarized Original Work

2000; Holm, de Lichtenberg, Thorup

Problem Definition

Let $G = (V, E)$ be an undirected weighted graph. The problem considered here is concerned with maintaining efficiently information about a minimum spanning tree of G (or minimum spanning forest if G is not connected), when G is subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. One expects from the dynamic algorithm to perform update operations faster than recomputing the entire minimum spanning tree from scratch.

Throughout, an algorithm is said to be *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

Key Results

The dynamic minimum spanning forest algorithm presented in this section builds upon the dynamic connectivity algorithm described in the entry ▸ Fully Dynamic Connectivity. In particu-

lar, a few simple changes to that algorithm are sufficient to maintain a minimum spanning forest of a weighted undirected graph upon deletions of edges [13]. A general reduction from [11] can then be applied to make the deletions-only algorithm fully dynamic.

This section starts by describing a decremental algorithm for maintaining a minimum spanning forest under deletions only. Throughout the sequence of deletions, the algorithm maintains a minimum spanning forest F of the dynamically changing graph G. The edges in F are referred to as *tree edges* and the other edges (in $G - F$) are referred to as *non-tree edges*. Let e be an edge being deleted. If e is a non-tree edge, then the minimum spanning forest does not need to change, so the interesting case is when e is a tree edge of forest F. Let T be the tree of F containing e. In this case, the deletion of e disconnects the tree T into two trees T_1 and T_2: to update the minimum spanning forest, one has to look for the minimum weight edge having one endpoint in T_1 and the other endpoint in T_2. Such an edge is called a *replacement edge* for e.

As for the dynamic connectivity algorithm, to search for replacement edges, the algorithm associates to each edge e a level $\ell(e)$ and, based on edge levels, maintains a set of sub-forests of the minimum spanning forest F: for each level i, forest F_i is the sub-forest induced by tree edges of level $\geq i$. Denoting by L the maximum edge level, it follows that:

$$F = F_0 \supseteq F_1 \supseteq F_2 \supseteq \cdots \supseteq F_L.$$

Initially, all edges have level 0; levels are then progressively increased, but never decreased. The changes of edge levels are accomplished so as to maintain the following invariants, which obviously hold at the beginning.

Invariant (1): F is a maximum spanning forest of G if edge levels are interpreted as weights.

Invariant (2): The number of nodes in each tree of F_i is at most $n/2^i$.

Invariant (3): Every cycle $\mathcal{C}$ has a non-tree edge of maximum weight and minimum level among all the edges in $\mathcal{C}$.

Invariant (1) should be interpreted as follows. Let (u,v) be a non-tree edge of level $\ell(u, v)$ and let $u \cdots v$ be the unique path between u and v in F (such a path exists since F is a spanning forest of G). Let e be any edge in $u \cdots v$ and let $\ell(e)$ be its level. Due to (1), $\ell(e) \geq \ell(u, v)$. Since this holds for each edge in the path, and by construction $F_{\ell(u,v)}$ contains all the tree edges of level $\geq \ell(u, v)$, the entire path is contained in $F_{\ell(u,v)}$, i.e., u and v are connected in $F_{\ell(u,v)}$.

Invariant (2) implies that the maximum number of levels is $L \leq \lfloor \log_2 n \rfloor$.

Invariant (3) can be used to prove that, among all the replacement edges, the lightest edge is on the maximum level. Let e_1 and e_2 be two replacement edges with $w(e_1) < w(e : 2)$, and let $\mathcal{C}_i$ be the cycle induced by e_i in F, $i = 1, 2$. Since F is a minimum spanning forest, e_i has maximum weight among all the edges in $\mathcal{C}_i$. In particular, since by hypothesis $w(e_1) < w(e : 2)$, e_2 is also the heaviest edge in cycle $\mathcal{C} = (\mathcal{C}_1 \cup \mathcal{C}_2) \setminus (\mathcal{C}_1 \cap \mathcal{C}_2)$. Thanks to Invariant (3), e_2 has minimum level in $\mathcal{C}$, proving that $\ell(e_2) \leq \ell(e_1)$. Thus, considering non-tree edges from higher to lower levels is correct.

Note that initially, an edge is is given level 0. Its level can be then increased at most $\lfloor \log_2 n \rfloor$ times as a consequence of edge deletions. When a tree edge $e = (v, w)$ of level $\ell(e)$ is deleted, the algorithm looks for a replacement edge at the highest possible level, if any. Due to invariant (1), such a replacement edge has level $\ell \leq \ell(e)$. Hence, a replacement subroutine `Replace`$((u, w), \ell(e))$ is called with parameters e and $\ell(e)$. The operations performed by this subroutine are now sketched.

`Replace`$((u, w), \ell)$ finds a replacement edge of the highest level $\leq \ell$, if any, considering edges in order of increasing weight. If such a replacement does not exist in level ℓ, there are two cases: if $\ell > 0$, the algorithm recurses on level $\ell - 1$; otherwise, $\ell = 0$, and the deletion of (v,w) disconnects v and w in G.

It is possible to show that `Replace` returns a replacement edge of minimum weight on the highest possible level, yielding the following lemma:

Lemma 1 *There exists a deletions-only minimum spanning forest algorithm that can be initialized on a graph with n vertices and m edges and supports any sequence of edge deletions in* $O(m \log^2 n)$ *total time.*

The description of a fully dynamic algorithm which performs updates in $O(\log^4 n)$ time now follows. The reduction used to obtain a fully dynamic algorithm is a slight generalization of the construction proposed by Henzinger and King [11] and works as follows.

Lemma 2 *Suppose there is a deletions-only minimum spanning tree algorithm that, for any* k *and* ℓ*, can be initialized on a graph with* k *vertices and* ℓ *edges and supports any sequence of* $\Omega(\ell)$ *deletions in total time* $O(\ell \cdot t(k, \ell))$*, where* t *is a non-decreasing function. Then there exists a fully-dynamic minimum spanning tree algorithm for a graph with n nodes starting with no edges, that, for m edges, supports updates in time*

$$O\left(\log^3 n + \sum_{i=1}^{3+\log_2 m} \sum_{j=1}^{i} t\left(min\{n, 2^j\}, 2^j\right)\right).$$

The interested reader is referred to references [11] and [13] for the description of the construction that proves Lemma 2. From Lemma 1 one gets $t(k, \ell) = O(\log^2 k)$. Hence, combining Lemmas 1 and 2, the claimed result follows:

Theorem 3 *There exists a fully-dynamic minimum spanning forest algorithm that, for a graph with n vertices, starting with no edges, maintains a minimum spanning forest in* $O(\log^4 n)$ *amortized time per edge insertion or deletion.*

There is a lower bound of $\Omega(\log n)$ for dynamic minimum spanning tree, given by Eppstein et al. [6], which uses the following argument. Let A be an algorithm for maintaining a minimum spanning tree of an arbitrary (multi)graph G. Let A be such that change weight(e, Δ) returns the edge f that replace e in the minimum spanning tree, if e is replaced. Clearly, any dynamic spanning tree algorithm can be modified to return f. One can use algorithm A to sort n positive numbers $x_1, x_2, \ldots, x_n$, as follows. Construct a multigraph G consisting of two nodes connected by $(n + 1)$ edges $e_0, e_1, \ldots, e_n$, such that edge e_0 has weight 0 and edge e_i has weight x_i. The initial spanning tree is e_0. Increase the weight of e_0 to $+\infty$. Whichever edge replaces e_0, say e_i, is the edge of minimum weight. Now increase the weight of e_i to $+\infty$: the replacement of e_i gives the second smallest weight. Continuing in this fashion gives the numbers sorted in increasing order. A similar argument applies when only edge decreases are allowed. Since Paul and Simon [14] have shown that any sorting algorithm needs $\Omega(n \log n)$ time to sort n numbers on a unit-cost random access machine whose repertoire of operations include additions, subtractions, multiplications and comparisons with 0, but not divisions or bit-wise Boolean operations, the following theorem follows.

Theorem 4 *Any unit-cost random access algorithm that performs additions, subtractions, multiplications and comparisons with 0, but not divisions or bit-wise Boolean operations, requires* $\Omega(\log n)$ *amortized time per operation to maintain a minimum spanning tree dynamically.*

Applications

Minimum spanning trees have applications in many areas, including network design, VLSI, and geometric optimization, and the problem of maintaining minimum spanning trees dynamically arises in such applications.

Algorithms for maintaining a minimum spanning forest of a graph can be used also for maintaining information about the connected components of a graph. There are also other applications of dynamic minimum spanning trees algorithms, which include finding the k smallest spanning trees [3–5, 8, 9], sampling spanning

trees [7] and dynamic matroid intersection problems [10]. Note that the first two problems are not necessarily dynamic: however, efficient solutions for these problems need dynamic data structures.

Open Problems

The first natural open question is to ask whether the gap between upper and lower bounds for the dynamic minimum spanning tree problem can be closed. Note that this is possible in the special case of plane graphs [6].

Second, the techniques for dynamic minimum spanning trees can be extended to dynamic 2-edge and 2-vertex connectivity, which indeed can be solved in polylogarithmic time per update. Can one extend the same technique also to higher forms of connectivity? This is particularly important, since the best known update bounds for higher edge and vertex connectivity are polynomial, and it would be useful to design polylogarithnmic algorithms at least for fully dynamic 3-edge and 3-vertex connectivity.

Experimental Results

A thorough empirical study on the performance evaluation of dynamic minimum spanning trees algorithms has been carried out in [1, 2].

Data Sets

Data sets are described in [1, 2].

Cross-References

- Dynamic Trees
- Fully Dynamic All Pairs Shortest Paths
- Fully Dynamic Connectivity
- Fully Dynamic Higher Connectivity
- Fully Dynamic Higher Connectivity for Planar Graphs
- Fully Dynamic Planarity Testing
- Fully Dynamic Transitive Closure

Recommended Reading

1. Alberts D, Cattaneo G, Italiano GF (1997) An empirical study of dynamic graph algorithms. ACM J Exp Algorithm 2
2. Cattaneo G, Faruolo P, Ferraro Petrillo U, Italiano GF (2002) Maintaining dynamic minimum spanning trees: an experimental study. In: Proceeding 4th workshop on algorithm engineering and experiments (ALENEX 02), 6–8 Jan, pp 111–125
3. Eppstein D (1992) Finding the *k* smallest spanning trees. BIT 32:237–248
4. Eppstein D (1994) Tree-weighted neighbors and geometric *k* smallest spanning trees. Int J Comput Geom Appl 4:229–238
5. Eppstein D, Galil Z, Italiano GF, Nissenzweig A (1997) Sparsification – a technique for speeding up dynamic graph algorithms. J Assoc Comput Mach 44(5):669–696
6. Eppstein D, Italiano GF, Tamassia R, Tarjan RE, Westbrook J, Yung M (1992) Maintenance of a minimum spanning forest in a dynamic plane graph. J Algorithms 13:33–54
7. Feder T, Mihail M (1992) Balanced matroids. In: Proceeding 24th ACM symposium on theory of computing, Victoria, 04–06 May, pp 26–38
8. Frederickson GN (1985) Data structures for on-line updating of minimum spanning trees. SIAM J Comput 14:781–798
9. Frederickson GN (1991) Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. In: Proceeding 32nd symposium on foundations of computer science, San Juan, 01–04 Oct, pp 632–641
10. Frederickson GN, Srinivas MA (1989) Algorithms and data structures for an expanded family of matroid intersection problems. SIAM J Comput 18:112–138
11. Henzinger MR, King V (2001) Maintaining minimum spanning forests in dynamic graphs. SIAM J Comput 31(2):364–374
12. Henzinger MR, King V (1999) Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J ACM 46(4):502–516
13. Holm J, de Lichtenberg K, Thorup M (2001) Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J ACM 48:723–760
14. Paul J, Simon W (1980) Decision trees and random access machines. In: Symposium über Logik und Algorithmik; See also Mehlhorn K (1984) Sorting and searching. Springer, Berlin, pp 85–97
15. Tarjan RE, Vishkin U (1985) An efficient parallel biconnectivity algorithm. SIAM J Comput 14:862–874

Fully Dynamic Planarity Testing

Giuseppe F. Italiano
Department of Computer and Systems Science, University of Rome, Rome, Italy
Department of Information and Computer Systems, University of Rome, Rome, Italy

Years and Authors of Summarized Original Work

1999; Galil, Italiano, Sarnak

Problem Definition

In this entry, the problem of maintaining a dynamic planar graph subject to edge insertions and edge deletions that preserve planarity but that can change the embedding is considered. Before formally defining the problem, few preliminary definitions follow.

A graph is *planar* if it can be embedded in the plane so that no two edges intersect. In a dynamic framework, a planar graph that is committed to an embedding is called *plane*, and the general term *planar* is used only when changes in the embedding are allowed. An edge insertion that preserves the embedding is called *embedding-preserving*, whereas it is called *planarity-preserving* if it keeps the graph planar, even though its embedding can change; finally, an edge insertion is called *arbitrary* if it is not known to preserve planarity. Extensive work on dynamic graph algorithms has used ad hoc techniques to solve a number of problems such as minimum spanning forests, 2-edge-connectivity and planarity testing for plane graphs (with embedding-preserving insertions) [5–7, 9–12]: this entry is concerned with more general planarity-preserving updates.

The work of Galil et al. [8] and of Eppstein et al. [3] provides a general technique for dynamic planar graph problems, including those mentioned above: in all these problems, one can deal with either arbitrary or planarity-preserving insertions and therefore allow changes of the embedding.

The *fully dynamic planarity testing problem* can be defined as follows. One wishes to maintain a (not necessarily planar) graph subject to *arbitrary* edge insertions and deletions, and allow queries that test whether the graph is currently planar, or whether a potential new edge would violate planarity.

Key Results

Eppstein et al. [3] provided a way to apply the sparsification technique [2] to families of graphs that are already sparse, such as planar graphs.

The new ideas behind this technique are the following. The notion of a certificate can be expanded to a definition for graphs in which a subset of the vertices are denoted as *interesting*; these *compressed certificates* may reduce the size of the graph by removing uninteresting vertices. Using this notion, one can define a type of sparsification based on *separators*, small sets of vertices the removal of which splits the graph into roughly equal size components. Recursively finding separators in these components gives a *separator tree* which can also be used as a *sparsification tree*; the interesting vertices in each certificate will be those vertices used in separators at higher levels of the tree. The notion of a *balanced separator tree*, which also partitions the interesting vertices evenly in the tree, is introduced: such a tree can be computed in linear time, and can be maintained dynamically. Using this technique, the following results can be achieved.

Theorem 1 *One can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow queries that test whether a new edge would violate planarity, in amortized time* $O(n^{1/2})$ *per update or query.*

This result can be improved, in order to allow arbitrary insertions or deletions, even if they might let the graph become nonplanar, using the following approach. The data structure above can be used to maintain a planar subgraph of the given

graph. Whenever one attempts to insert a new edge, and the resulting graph would be nonplanar, the algorithm does not actually perform the insertion, but instead adds the edge to a list of *nonplanar edges*. Whenever a query is performed, and the list of nonplanar edges is nonempty, the algorithm attempts once more to add those edges one at a time to the planar subgraph. The time for each successful addition can be charged to the insertion operation that put that edge in the list of nonplanar edges. As soon as the algorithm finds some edge in the list that can not be added, it stops trying to add the other edges in the list. The time for this failed insertion can be charged to the query the algorithm is currently performing. In this way the list of nonplanar edges will be empty if and only if the graph is planar, and the algorithm can test planarity even for updates in nonplanar graphs.

Theorem 2 *One can maintain a graph, subject to arbitrary insertions and deletions, and allow queries that test whether the graph is presently planar or whether a new edge would violate planarity, in amortized time* $O(n^{1/2})$ *per update or query.*

Applications

Planar graphs are perhaps one of the most important interesting subclasses of graphs which combine beautiful structural results with relevance in applications. In particular, planarity testing is a basic problem, which appears naturally in many applications, such as VLSI layout, graphics, and computer aided design. In all these applications, there seems to be a need for dealing with dynamic updates.

Open Problems

The $O(n^{1/2})$ bound for planarity testing is amortized. Can we improve this bound or make it worst-case?

Finally, the complexity of the algorithms presented here, and the large constant factors involved in some of the asymptotic time bounds, make some of the results unsuitable for practical applications. Can one simplify the methods while retaining similar theoretical bounds?

Cross-References

- Dynamic Trees
- Fully Dynamic All Pairs Shortest Paths
- Fully Dynamic Connectivity
- Fully Dynamic Higher Connectivity
- Fully Dynamic Higher Connectivity for Planar Graphs
- Fully Dynamic Minimum Spanning Trees
- Fully Dynamic Transitive Closure

Recommended Reading

1. Cimikowski R (1994) Branch-and-bound techniques for the maximum planar subgraph problem. Int J Comput Math 53:135–147
2. Eppstein D, Galil Z, Italiano GF, Nissenzweig A (1997) Sparsification – a technique for speeding up dynamic graph algorithms. J Assoc Comput Mach 44(5):669–696
3. Eppstein D, Galil Z, Italiano GF, Spencer TH (1996) Separator based sparsification I: planarity testing and minimum spanning trees. J Comput Syst Sci Spec Issue STOC 93 52(1):3–27
4. Eppstein D, Galil Z, Italiano GF, Spencer TH (1999) Separator based sparsification II: edge and vertex connectivity. SIAM J Comput 28: 341–381
5. Eppstein D, Italiano GF, Tamassia R, Tarjan RE, Westbrook J, Yung M (1992) Maintenance of a minimum spanning forest in a dynamic plane graph. J Algorithms 13:33–54
6. Frederickson GN (1985) Data structures for on-line updating of minimum spanning trees, with applications. SIAM J Comput 14:781–798
7. Frederickson GN (1997) Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. SIAM J Comput 26(2):484–538
8. Galil Z, Italiano GF, Sarnak N (1999) Fully dynamic planarity testing with applications. J ACM 48:28–91
9. Giammarresi D, Italiano GF (1996) Decremental 2- and 3-connectivity on planar graphs. Algorithmica 16(3):263–287
10. Hershberger J, Suri MR, Suri S (1994) Data structures for two-edge connectivity in planar graphs. Theor Comput Sci 130(1):139–161
11. Italiano GF, La Poutré JA, Rauch M (1993) Fully dynamic planarity testing in planar embedded graphs.

In: 1st annual European symposium on algorithms, Bad Honnef, 30 Sept–2 Oct
12. Tamassia R (1988) A dynamic data structure for planar graph embedding. In: 15th international colloquium automata, languages, and programming. LNCS, vol 317. Springer, Berlin, pp 576–590

Fully Dynamic Transitive Closure

Valerie King
Department of Computer Science, University of Victoria, Victoria, BC, Canada

Keywords

All-pairs dynamic reachability; Fully dynamic graph algorithm for maintaining transitive closure; Incremental algorithms for digraphs;

Years and Authors of Summarized Original Work

1999; King

Problem Definition

Design a data structure for a directed graph with a fixed set of node which can process queries of the form "Is there a path from i to j ?" and updates of the form: "Insert edge (i, j)"; "Delete edge (i, j)". The goal is to minimize update and query times, over the worst case sequence of queries and updates. Algorithms to solve this problem are called "fully dynamic" as opposed to "partially dynamic" since both insertions and deletions are allowed.

Key Results

This work [4] gives the first deterministic fully dynamic graph algorithm for maintaining the transitive closure in a directed graph. It uses $O(n^2 \log n)$ amortized time per update and $O(1)$ worst case query time where n is number of nodes in the graph. The basic technique is extended to give fully dynamic algorithms for approximate and exact all-pairs shortest paths problems.

The basic building block of these algorithms is a method of maintaining all-pairs shortest paths with insertions and deletions for distances up to d. For each vertex v, a single-source shortest path tree of depth d which reach v ("In_v") and another tree of vertices which are reached by v ("Out_v") are maintained during any sequence of deletions. Each insert of a set of edges incident to v results in the rebuilding of In_v and Out_v I. For each pair of vertices x, y and each length, a count is kept of the number of v such that there is a path from x in In_v to y in Out_v of that length.

To maintain transitive closure, log n levels of these trees are maintained for trees of depth 2, where the edges used to construct a forest on one level depend on the paths in the forest of the previous level.

Space required was reduced from $O(n^3)$ to $O(n^2)$ in [6]. A log n factor was shaved off [7, 10]. Other tradeoffs between update and query time are given in [1, 7–10]. A deletions only randomized transitive closure algorithm running in $O(mn)$ time overall is given by [8] where m is the initial number of edges in the graph. A simple monte carlo transitive closure algorithm for acyclic graphs is presented in [5]. Dynamic single source reachability in a digraph is presented in [8, 9]. All-pairs shortest paths can be maintained with nearly the same update time [2].

Applications

None

Open Problems

Can reachability from a single source in a directed graph be maintained in $o(mn)$ time over a worst case sequence of m deletions?

Can strongly connected components be maintained in $o(mn)$ time over a worst case sequence of m deletions?

Experimental Results

Experimental results on older techniques can be found in [3].

Cross-References

- All Pairs Shortest Paths in Sparse Graphs
- All Pairs Shortest Paths via Matrix Multiplication
- Fully Dynamic All Pairs Shortest Paths
- Fully Dynamic Connectivity

Recommended Reading

1. Demestrescu C, Italiano GF (2005) Trade-offs for fully dynamic transitive closure on DAG's: breaking through the $O(n^2)$ barrier, (presented in FOCS 2000). J ACM 52(2):147–156
2. Demestrescu C, Italiano GF (2004) A new approach to dynamic all pairs shortest paths, (presented in STOC 2003). J ACM 51(6):968–992
3. Frigioni D, Miller T, Nanni U, Zaroliagis CD (2001) An experimental study of dynamic algorithms for transitive closure. ACM J Exp Algorithms 6(9)
4. King V (1999) Fully dynamic algorithms for maintaining all-pairs shortest paths andtransitive closure in digraphs. In: Proceedings of the 40th annual IEEE symposium on foundation of computer science (ComiIEEE FOCS). IEEE Computer Society, New York, pp 81–91
5. King V, Sagert G (2002) A fully dynamic algorithm for maintaining the transitive closure (presented in FOCS 1999). JCCS 65(1):150–167
6. King V, Thorup M (2001) A space saving trick for dynamic transitive closure and shortest path algorithms. In: Proceedings of the 7th annual international conference of computing and cominatorics (COCOON). Lecture notes computer science, vol 2108/2001. Springer, Heidelberg, pp 269–277
7. Roditty L (2003) A faster and simpler fully dynamic transitive closure. In: Proceedings of the 14th annual ACM-SIAM symposium on discrete algorithms (ACMIEEE SODA). ACM, Baltimore, pp 404–412
8. Roditty L, Zwick U (2002) Improved dynamic reachability algorithms for directed graphs. In: Proceedings of the 43rd annual symposium on foundation of computer science (IEEE FOCS). IEEE Computer Society, Vancouver, pp 679–688
9. Roditty L, Zwick U (2004) A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In: Proceedings of the 36th ACM symposium on theory of computing (ACMSTOC). ACM, Chicago, pp 184–191
10. Sankowski S (2004) Dynamic transitive closure via dynamic matrix inverse. In: Proceedings of the 45th annual symposium on foundations of computer science (IEEE FOCS). IEEE Computer Society, Rome, pp 509–517

G

Gate Sizing

Vijay Sundararajan
Broadcom Corp, Fremont, CA, USA

Keywords

Fast and exact transistor sizing

Years and Authors of Summarized Original Work

2002; Sundararajan, Sapatnekar, Parhi

Problem Definition

For a detailed exposition of the solution approach presented in this entry, please refer to [15]. As evidenced by the successive announcement of ever-faster computer systems in the past decade, increasing the speed of VLSI systems continues to be one of the major requirements for VLSI system designers today. Faster integrated circuits are making possible newer applications that were traditionally considered difficult to implement in hardware. In this scenario of increasing circuit complexity, reduction of circuit delay in integrated circuits is an important design objective. Transistor sizing is one such task that has been employed for speeding up circuits for quite some time now [6]. Given the circuit topology, the delay of a combinational circuit can be controlled by varying the sizes of transistors in the circuit. Here, the size of a transistor is measured in terms of its channel width, since the channel lengths of MOS transistors in a digital circuit are generally uniform. In any case, what really matters is the ratio of channel width to channel length, and if channel lengths are not uniform, this ratio can be considered as the size. In coarse terms, the circuit delay can usually be reduced by increasing the sizes of certain transistors in the circuit from the minimum size. Hence, making the circuit faster usually entails the penalty of increased circuit area relative to a minimum-sized circuit, and the area-delay trade-off involved here is the problem of transistor size optimization. A related problem to transistor sizing is called gate sizing, where a logic gate in a circuit is modeled as an equivalent inverter and the sizing optimization is carried out on this modified circuit with equivalent inverters in place of more complex gates. There is, therefore, a reduction in the number of size parameters corresponding to every gate in the circuit. Needless to say, this is an easier problem to solve than the general transistor sizing problem. Note that gate sizing mentioned here is distinct from library-specific gate sizing that is a discrete optimization problem targeted to selecting appropriate gate sizes from an underlying cell library. The gate sizing problem targeted here is one of continuous gate sizing where the gate sizes are allowed to vary in a continuous manner between a minimum and a maximum size. There has been a large amount of work done on transistor sizing

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

[1–3, 5, 6, 9, 10, 12, 13], that underlines the importance of this optimization technique. Starting from a minimum-sized circuit, TILOS, [6], uses a greedy strategy for transistor sizing by iteratively sizing transistors in the critical path. A sensitivity factor is calculated for every transistor in the critical path to quantify the gain in circuit speed achieved by a unit upsizing of the transistor. The most sensitive transistor is then bumped up in size by a small constant factor to speed up the circuit. This process is repeated iteratively until the timing requirements are met. The technique is extremely simple to implement and has run-time behavior proportional to the size of the circuit. Its chief drawback is that it does not have guaranteed convergence properties and hence is not an exact optimization technique.

Key Results

The solution presented in the entry heretofore referred to as MINFLOTRANSIT was a novel way of solving the transistor sizing problem exactly and in an extremely fast manner. Even though the entry treats transistor sizing, in the description, the results apply as well to the less general problem of continuous gate sizing as described earlier. The proposed approach has some similarity in form to [2, 5, 8] which will be subsequently explained, but the similarity in content is minimal and the details of implementation are vastly different.

In essence, the proposed technique and the techniques in [2, 5, 8] are iterative relaxation approaches that involve a two-step optimization strategy. The first step involves a delay budgeting step where optimal delays are computed for transistors/gates. The second step involves sizing transistors optimally under this "constant delay" model to achieve these delay budgets. The two steps are iteratively alternated until the solution converges, i.e., until the delay budgets calculated in the first step are exactly satisfied by the transistor sizes determined by the second step.

The primary features of the proposed approach are:

- It is computationally fast and is comparable to TILOS in its run-time behavior.
- It can be used for true transistor sizing as well as the relaxed problem of gate sizing. Additionally, the approach can easily incorporate wire sizing [15].
- It can be adapted for more general delay models than the Elmore delay model [15].

The starting point for the proposed approach is a fast guess solution. This could be obtained, for example, from a circuit that has been optimized using TILOS to meet the given delay requirements. The proposed approach, as outlined earlier, is an iterative relaxation procedure that involves an alternating two-phase relaxed optimization sequence that is repeated iteratively until convergence is achieved. The two phases in the proposed approach are:

- The **D-phase** where transistor sizes are assumed fixed and transistor delays are regarded as variable parameters. Irrespective of the delay model employed, this phase can be formulated as the dual of a min-cost network flow problem. Using $|V|$ to denote the number of transistors and $|E|$ the number of wires in the circuit, this step in our application has worst-case complexity of $O(|V||E|\log(\log|V|))$ [7].
- The **W-phase** where transistor/gate delays are assumed fixed and their sizes are regarded as variable parameters. As long as the gate delay can be expressed as a separable function of the transistor sizes, this step can be solved as a Simple Monotonic Program (SMP) [11]. The complexity of SMP is similar to an all-pairs shortest-path algorithm in a directed graph, [4, 11], i.e., $O(V||E|)$.

The objective function for the problem is the minimization of circuit area. In the W-phase, this

Gate Sizing, Table 1 Comparison of TILOS and MINFLOTRANSIT on a Sun Ultrasparc 10 workstation for ISCAS85 and MCNC91 benchmarks for 0.13 um technology. The delay specs are with respect to a minimum-sized circuit. The optimization approach followed here was gate sizing

Circuit	# Gates	Area saved over TILOS (%)	Delay specs. (D_{min})	CPU time (TILOS) (s)	CPU time (OURS) (s)
Adder32	480	≤ 1	0.5	2.2	5
Adder256	3,840	≤ 1	0.5	262	608
Cm163a	65	2.1	0.55	0.13	0.32
Cm162a	71	10.4	0.5	0.23	0.96
Parity8	89	37	0.45	0.68	2.15
Frg1	177	1.9	0.7	0.55	1.49
Population	518	6.7	0.4	57	179
Pmult8	1,431	5	0.5	637	1476
Alu2	826	2.6	0.6	28	71
C432	160	9.4	0.4	0.5	4.8
C499	202	7.2	0.57	1.47	11.26
C880	383	4	0.4	2.7	8, 2
C1355	546	9.5	0.4	29	76
C1908	880	4.6	0.4	36	84
C2670	1,193	9.1	0.4	27	69
C3540	1,669	7.7	0.4	226	651
C5315	2,307	2	0.4	90	201
C6288	2,416	16.5	0.4	1,677	4,138
C7552	3,512	3.3	0.4	320	683

objective is addressed directly, and in the D-phase the objective is chosen to facilitate a move in the solution space in a direction that is known to lead to a reduction in the circuit area.

Applications

The primary application of the solution provided here is circuit and system optimization in automated VLSI design. The solution provided here can enable electronic design automation (EDA) tools that take a holistic approach toward transistor sizing. This will in turn enable making custom circuit design flows more realizable in practice. The mechanics of some of the elements of the solution provided here especially the **D-phase** have been used to address other circuit optimization problems [14].

Open Problems

The related problem of discrete gate sizing optimization matching gate sized to available gate sizes from a standard cell library is a provably hard optimization problem which could be aided by the development of efficient heuristics and probabilistic algorithms.

Experimental Results

A relative comparison of MINFLOTRANSIT with TILOS is provided in Table 1 for gate sizing of ISACS85 and mcnc91 benchmark circuits. As can be seen a significant performance improvement is observed with a tolerable loss in execution time.

Cross-References

- ▸ Circuit Retiming
- ▸ Wire Sizing

Recommended Reading

1. Chen CP, Chu CN, Wong DF (1998) Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation. In: Proceedings of the 1998 IEEE/ACM international conference on computer-aided design, San Jose, pp 617–624
2. Chen HY, Kang SM (1991) iCOACH: a circuit optimization aid for CMOS high-performance circuits. Intergr VLSI J 10(2):185–212
3. Conn AR, Coulman PK, Haring RA, Morrill GL, Visweshwariah C, Wu CW (1998) Jiffy Tune: circuit optimization using time-domain sensitivities. IEEE Trans Comput Aided Des Intergr Circuits Syst 17(12):1292–1309
4. Cormen TH, Leiserson CE, Rivest RL (1990) Introduction to algorithms. McGraw-Hill, New York
5. Dai Z, Asada K (1989) MOSIZ: a two-step transistor sizing algorithm based on optimal timing assignment method for multi-stage complex gates. In: Proceedings of the 1989 custom integrated circuits conference, New York, pp 17.3.1–17.3.4
6. Fishburn JP, Dunlop AE (1985) TILOS: a posynomial programming approach to transistor sizing. In: Proceedings of the 1985 international conference on computer-aided design, Santa Clara, pp 326–328
7. Goldberg AV, Grigoriadis MD, Tarjan RE (1991) Use of dynamic trees in a network simplex algorithm for the maximum flow problem. Math Program 50(3):277–290
8. Grodstein J, Lehman E, Harkness H, Grundmann B, Watanabe Y (1995) A delay model for logic synthesis of continuously sized networks. In: Proceedings of the 1995 international conference on computer-aided design, San Jose, pp 458–462
9. Marple DP (1986) Performance optimization of digital VLSI circuits. Technical report CSL-TR-86-308, Stanford University
10. Marple DP (1989) Transistor size optimization in the tailor layout system. In: Proceedings of the 26th ACM/IEEE design automation conference, Las Vegas, pp 43–48
11. Papaefthymiou MC (1998) Asymptotically efficient retiming under setup and hold constraints. In: Proceedings of the IEEE/ACM international conference on computer-aided design, San Jose, pp 288–295
12. Sapatnekar SS, Rao VB, Vaidya PM, Kang SM (1993) An exact solution to the transistor sizing problem for CMOS circuits using convex optimization. IEEE Trans Comput Aided Des 12(11): 1621–1634
13. Shyu JM, Sangiovanni-Vincentelli AL, Fishburn JP, Dunlop AE (1988) Optimization-based transistor sizing. IEEE J Solid State Circuits 23(2):400–409
14. Sundararajan V, Parhi K (1999) Low power synthesis of dual threshold voltage CMOS VLSI circuits. In: Proceedings of the international symposium on low power electronics and design, San Diego, pp 139–144
15. Sundararajan V, Sapatnekar SS, Parhi KK (2002) Fast and exact transistor sizing based on iterative relaxation. Comput-Aided Design Intergr Circuits Syst IEEE Trans 21(5):568–581

General Equilibrium

Li-Sha Huang
Department of Computer Science and Technology, Tsinghua University, Beijing, China

Keywords

Competitive market equilibrium

Years and Authors of Summarized Original Work

2002; Deng, Papadimitriou, Safra

Problem Definition

This problem is concerned with the computational complexity of finding an exchange market equilibrium. The exchange market model consists of a set of agents, each with an initial endowment of commodities, interacting through a market, trying to maximize each's utility function. The equilibrium prices are determined by a clearance condition. That is, all commodities are bought, collectively, by all the utility maximizing agents, subject to their budget constraints (determined by the values of their initial endowments of commodities at the market price). The work of Deng, Papadimitriou and Safra [3] studies the complexity, approximability, inapproximability, and communication complexity of finding equilibrium prices. The work shows the NP-hardness

of approximating the equilibrium in a market with indivisible goods. For markets with divisible goods and linear utility functions, it develops a pseudo-polynomial time algorithm for computing an ϵ-equilibrium. It also gives a communication complexity lower bound for computing Pareto allocations in markets with non-strictly concave utility functions.

Market Model

In a pure exchange economy, there are m traders, labeled by $i = 1, 2, ..., m$, and n types of commodities, labeled by $j = 1, 2, ..., n$. The commodities could be divisible or indivisible. Each trader i comes to the market with initial endowment of commodities, denoted by a vector $w_i \in \mathbb{R}^n_+$, whose j-th entry is the amount of commodity j held by trader i.

Associate each trader i a *consumption set* X_i to represents the set of possible commodity bundles for him. For example, when there are n_1 divisible commodities and $(n - n_1)$ indivisible commodities, X_i can be $\mathbb{R}^{n_1}_+ \times \mathbb{Z}^{n-n_1}_+$. Each trader has a utility function $X_i \mapsto \mathbb{R}_+$ to present his utility for a bundle of commodities. Usually, the utility function is required to be concave and nondecreasing.

In the market, each trader acts as both a buyer and a seller to maximize his utility. At a certain price $p \in \mathbb{R}^n_+$, trader i is is solving the following optimization problem, under his budget constraint:

$$\max \ u_i(x_i) \ \ s.t. \quad x_i \in X_i \text{ and } \langle p, x_i \rangle \leq \langle p, w_i \rangle.$$

Definition 1 An equilibrium in a pure exchange economy is a price vector $\bar{p} \in \mathbb{R}^n_+$ and bundles of commodities $\{\bar{x}_i \in \mathbb{R}^n_+, i = 1, ..., m\}$, such that

$$\bar{x_i} \in \text{argmax}\{u_i(x_i) | x_i \in X_i \text{ and } \langle x_i, \bar{p} \rangle \leq \langle w_i, \bar{p} \rangle\},$$
$$\forall 1 \leq i \leq m$$
$$\sum_{i=1}^{m} \bar{x}_{ij} \leq \sum_{i=1}^{m} w_{ij}, \forall 1 \leq j \leq n.$$

The concept of approximate equilibrium was introduced in [3]:

Definition 2 ([3)] An ϵ-approximate equilibrium in an exchange market is a price vector $\bar{p} \in \mathbb{R}^n_+$ and bundles of goods $\{\bar{x}_i \in \mathbb{R}^n_+, i = 1, ..., m\}$, such that

$$u_i(\bar{x}_i) \geq \frac{1}{1+\epsilon} \max\{u_i(x_i) | x_i \in X_i, \langle x_i, \bar{p} \rangle \leq \langle w_i, \bar{p} \rangle\}, \forall i \quad (1)$$

$$\langle \bar{x}_i, \bar{p} \rangle \leq (1+\epsilon) \langle w_i, \bar{p} \rangle, \forall i \quad (2)$$

$$\sum_{i=1}^{m} \bar{x}_{ij} \leq (1+\epsilon) \sum_{i=1}^{m} w_{ij}, \forall j \ . \quad (3)$$

Key Results

A linear market is a market in which all the agents have linear utility functions. The deficiency of a market is the smallest $\epsilon \geq 0$ for which an ϵ-approximate equilibrium exists.

Theorem 1 *The deficiency of a linear market with indivisible goods is NP-hard to compute, even if the number of agents is two. The deficiency is also NP-hard to approximate within 1/3.*

Theorem 2 *There is a polynomial-time algorithm for finding an equilibrium in linear markets with bounded number of divisible goods. Ditto for a polynomial number of agents.*

Theorem 3 *If the number of goods is bounded, there is a polynomial-time algorithm which, for any linear indivisible market for which a price equilibrium exists, and for any $\epsilon > 0$, finds an ϵ-approximate equilibrium.*

If the utility functions are strictly concave and the equilibrium prices are broadcasted to all agents, the equilibrium allocation can be computed distributely without any communication, since each agent's basket of goods is uniquely determined. However, if the utility functions are not strictly concave, e.g., linear functions, communications are needed to coordinate the agents' behaviors.

Theorem 4 *Any protocol with binary domains for computing Pareto allocations of m agents and n divisible commodities with concave utility functions (resp. ϵ-Pareto allocations for indivisible commodities, for any $\epsilon < 1$) must have market communication complexity $\Omega(m \log(m+n))$ bits.*

Applications

This concept of market equilibrium is the outcome of a sequence of efforts trying to fully understand the laws that govern human commercial activities, starting with the "invisible hand" of Adam Smith, and finally, the mathematical conclusion of Arrow and Debreu [1] that there exists a set of prices that bring supply and demand into equilibrium, under quite general conditions on the agent utility functions and their optimization behavior.

The work of Deng, Papadimitriou and Safra [3] explicitly called for an algorithmic complexity study of the problem, and developed interesting complexity results and approximation algorithms for several classes of utility functions. There has since been a surge of algorithmic study for the computation of the price equilibrium problem with continuous variables, discovering and rediscovering polynomial time algorithms for many classes of utility functions, see [2, 4–9].

Significant progress has been made in the above directions but only as a first step. New ideas and methods have already been invented and applied in reality. The next significant step will soon manifest itself with many active studies in microeconomic behavior analysis for E-commercial markets. Nevertheless the algorithmic analytic foundation in [3] will be an indispensable tool for further development in this reincarnated exciting field.

Open Problems

The most important open problem is what is the computational complexity for finding the equilibrium price, as guaranteed by the Arrow–Debreu theorem. To the best of the author's knowledge, only the markets whose set of equilibria is convex can be solved in polynomial time with current techniques. And approximating equilibria in some markets with disconnected set of equilibria, e.g., Leontief economies, are shown to be PPAD-hard. Is the convexity or (weakly) gross substitutability a necessary condition for a market to be polynomial-time solvable?

Second, how to handle the dynamic case is especially interesting in theory, mathematical modeling, and algorithmic complexity as bounded rationality. Great progress must be made in those directions for any theoretical work to be meaningful in practice.

Third, incentive compatible mechanism design protocols for the auction models have been most actively studied recently, especially with the rise of E-Commerce. Especially at this level, a proper approximate version of the equilibrium concept handling price dynamics should be especially important.

Cross-References

- Complexity of Core
- Leontief Economy Equilibrium
- Non-approximability of Bimatrix Nash Equilibria

Recommended Reading

1. Arrow KJ, Debreu G (1954) Existence of an equilibrium for a competitive economy. Econometrica 22(3):265–290
2. Codenotti B, McCune B, Varadarajan K (2005) Market equilibrium via the excess demand function. In: Proceedings STOC'05. ACM, Baltimore, pp 74–83
3. Deng X, Papadimitriou C, Safra S (2002) On the complexity of price equilibria. J Comput Syst Sci 67(2):311–324
4. Devanur NR, Papadimitriou CH, Saberi A, Vazirani VV (2002) Market equilibria via a primal-dual-type algorithm. In: Proceedings of FOCS'02. IEEE Computer Society, Vancouver, pp 389–395
5. Eaves BC (1985) Finite solution for pure trade markets with Cobb-Douglas utilities. Math Program Study 23:226–239

6. Garg R, Kapoor S (2004) Auction algorithms for market equilibrium. In: Proceedings of STOC'04. ACM, Chicago, pp 511–518
7. Jain K (2004) A polynomial time algorithm for computing the Arrow-Debreu market equilibrium for linear utilities. In: Proceeding of FOCS'04. IEEE Computer Society, Rome, pp 286–294
8. Nenakhov E, Primak M (1983) About one algorithm for finding the solution of the Arrow-Debreu model. Kibernetica 3:127–128
9. Ye Y (2008) A path to the Arrow-Debreu competitive market equilibrium. Math Program 111(1–2):315–348

Generalized Steiner Network

Julia Chuzhoy
Toyota Technological Institute, Chicago, IL, USA

Keywords

Survivable network design

Years and Authors of Summarized Original Work

2001; Jain

Problem Definition

The generalized Steiner network problem is a network design problem, where the input consists of a graph together with a collection of connectivity requirements, and the goal is to find the cheapest subgraph meeting these requirements.

Formally, the input to the generalized Steiner network problem is an undirected multigraph $G = (V, E)$, where each edge $e \in E$ has a non-negative cost $c(e)$, and for each pair of vertices $i, j \in V$, there is a connectivity requirement $r_{i,j} \in \mathbb{Z}$. A feasible solution is a subset $E' \subseteq E$ of edges, such that every pair $i, j \in V$ of vertices is connected by at least $r_{i,j}$ edge-disjoint path in graph $G' = (V, E')$. The generalized Steiner network problem asks to find a solution E' of minimum cost $\sum_{e \in E'} c(e)$.

This problem generalizes several classical network design problems. Some examples include minimum spanning tree, Steiner tree and Steiner forest. The most general special case for which a 2-approximation was previously known is the Steiner forest problem [1, 4].

Williamson et al. [8] were the first to show a non-trivial approximation algorithm for the generalized Steiner network problem, achieving a $2k$-approximation, where $k = \max_{i,j \in V}\{r_{i,j}\}$. This result was improved to $O(\log k)$-approximation by Goemans et al. [3].

Key Results

The main result of [6] is a factor-2 approximation algorithm for the generalized Steiner network problem. The techniques used in the design and the analysis of the algorithm seem to be of independent interest.

The 2-approximation is achieved for a more general problem, defined as follows. The input is a multigraph $G = (V, E)$ with costs $c(\cdot)$ on edges, and connectivity requirement function $f : 2^V \to \mathbb{Z}$. Function f is weakly submodular, i.e., it has the following properties:

1. $f(V) = 0$.
2. For all $A, B \subseteq V$, at least one of the following two conditions holds:
   - $f(A) + f(B) \leq f(A \setminus B) + f(B \setminus A)$.
   - $f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$.

For any subset $S \subseteq V$ of vertices, let $\delta(S)$ denote the set of edges with exactly one endpoint in S. The goal is to find a minimum-cost subset of edges $E' \subseteq E$, such that for every subset $S \subseteq V$ of vertices, $|\delta(S) \cap E'| \geq f(S)$.

This problem can be equivalently expressed as an integer program. For each edge $e \in E$, let x_e be the indicator variable of whether e belongs to the solution.

$$\text{(IP)} \qquad \min \sum_{e \in E} c(e) x_e$$

subject to:

$$\sum_{e \in \delta(S)} x_e \geq f(S) \qquad \forall S \subseteq V \qquad (1)$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \qquad (2)$$

It is easy to see that the generalized Steiner network problem is a special case of (IP), where for each $S \subseteq V$, $f(S) = \max_{i \in S, j \notin S}\{r_{i,j}\}$.

Techniques

The approximation algorithm uses the LP-rounding technique. The initial linear program (LP) is obtained from (IP) by replacing the integrality constraint (2) with:

$$0 \leq x_e \leq 1 \qquad \forall e \in E \qquad (3)$$

It is assumed that there is a separation oracle for (LP). It is easy to see that such an oracle exists if (LP) is obtained from the generalized Steiner network problem. The key result used in the design and the analysis of the algorithm is summarized in the following theorem.

Theorem 1 *In any basic solution of (LP), there is at least one edge $e \in E$ with $x_e \geq 1/2$.*

The approximation algorithm works by iterative LP-rounding. Given a basic optimal solution of (LP), let $E^* \subseteq E$ be the subset of edges e with $x_e \geq 1/2$. The edges of E^* are removed from the graph (and are eventually added to the solution), and the problem is then solved recursively on the residual graph, by solving (LP) on $G^* = (V, E \setminus E^*)$, where for each subset $S \subseteq V$, the new requirement is $f(S) - |\delta(S) \cap E^*|$. The main observation that leads to factor-2 approximation is the following: if E' is a 2-approximation for the residual problem, then $E' \cup E^*$ is a 2-approximation for the original problem.

Given any solution to (LP), set $S \subseteq V$ is called *tight* iff constraint (1) holds with equality for S. The proof of Theorem 1 involves constructing a large *laminar family* of tight sets (a family where for every pair of sets, either one set contains the other, or the two sets are disjoint). After that a clever accounting scheme that charges edges to the sets of the laminar family is used to show that there is at least one edge $e \in E$ with $x_e \geq 1/2$.

Applications

Generalized Steiner network is a very basic and natural network design problem that has many applications in different areas, including the design of communication networks, VLSI design and vehicle routing. One example is the design of survivable communication networks, which remain functional even after the failure of some network components (see [5] for more details).

Open Problems

The 2-approximation algorithm of Jain [6] for generalized Steiner network is based on LP-rounding, and it has high running time. It would be interesting to design a combinatorial approximation algorithm for this problem.

It is not known whether a better approximation is possible for generalized Steiner network. Very few hardness of approximation results are known for this type of problems. The best current hardness factor stands on 1.01063 [2], and this result is valid even for the special case of Steiner tree.

Cross-References

- Steiner Forest
- Steiner Trees

Recommended Reading

1. Agrawal A, Klein P, Ravi R (1995) When trees collide: an approximation algorithm for the generalized Steiner problem on networks. J SIAM Comput 24(3):440–456
2. Chlebik M, Chlebikova J (2002) Approximation hardness of the Steiner tree problem on graphs. In: 8th Scandinavian workshop on algorithm theory LNCS, vol 2368, pp 170–179
3. Goemans MX, Goldberg AV, Plotkin SA, Shmoys DB, Tardos É, Williamson DP (1994) Improved approximation algorithms for network design problems. In: Proceedings of the fifth annual ACM-SIAM symposium on discrete algorithms (SODA), pp 223–232

4. Goemans MX, Williamson DP (1995) A general approximation technique for constrained forest problems. SIAM J Comput 24(2):296–317
5. Grötschel M, Monma CL, Stoer M (1995) Design of survivable networks. In: Network models, handbooks in operations research and management science. North Holland Press, Amsterdam
6. Jain K (2001) A factor 2 approximation algorithm for the generalized Steiner network problem. Combinatorica 21(1):39–60
7. Vazirani VV (2001) Approximation algorithms. Springer, Berlin
8. Williamson DP, Goemans MX, Mihail M, Vazirani VV (1995) A primal-dual approximation algorithm for generalized Steiner network problems. Combinatorica 15(3):435–454

Generalized Two-Server Problem

René A. Sitters
Department of Econometrics and Operations Research, VU University, Amsterdam, The Netherlands

Keywords

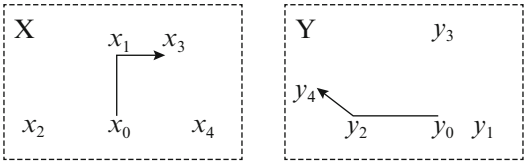
CNN problem

Years and Authors of Summarized Original Work

2006; Sitters, Stougie

Problem Definition

In the *generalized two-server problem*, we are given two servers: one moving in a metric space $\mathbb{X}$ and one moving in a metric space $\mathbb{Y}$. They are to serve requests $r \in \mathbb{X} \times \mathbb{Y}$which arrive one by one. A request $r = (x, y)$ is served by moving either the $\mathbb{X}$-server to point x or the $\mathbb{Y}$-server to point y. The decision as to which server to move to the next request is irrevocable and has to be taken without any knowledge about future requests. The objective is to minimize the total distance traveled by the two servers (Fig. 1).



Generalized Two-Server Problem, Fig. 1 In this example, both servers move in the plane and start from the configuration (x_0, y_0). The $\mathbb{X}$-server moves through requests 1 and 3, and the $\mathbb{Y}$-server takes care of requests 2 and 4. The cost of this solution is the sum of the path-lengths

Online Routing Problems

The generalized two-server problem belongs to a class of routing problems called *metrical service systems* [4, 10]. Such a system is defined by a metric space $\mathbb{M}$ of all possible system configurations, an initial configuration $\mathcal{C}_0$, and a set $\mathcal{R}$ of possible requests, where each request $r \in \mathcal{R}$ is a subset of $\mathbb{M}$. Given a sequence, $r_1, r_2 \ldots, r_n$, of requests, a feasible solution is a sequence, $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$, of configurations such that $C_i \in r_i$ for all $i \in \{1,\ldots,n\}$.

When we model the generalized two-server problem as a metrical service system we have $\mathbb{M} = \mathbb{X} \times \mathbb{Y}$ and $\mathcal{R} = \{\{x \times \mathbb{Y}\} \cup \{\mathbb{X} \times y\} | x \in \mathbb{X}, y \in \mathbb{Y}\}$. In the classical *two-server problem*, both servers move in the same space and receive the same requests, that is, $\mathbb{M} = \mathbb{X} \times \mathbb{X}$ and $\mathcal{R} = \{\{x \times \mathbb{Y}\} \cup \{\mathbb{X} \times x\} | x \in \mathbb{X}\}$.

The performance of algorithms for online optimization problems is often measured using *competitive analysis*. We say that an algorithm is α*-competitive* ($\alpha \geq 1$) for some minimization problem if for every possible instance the cost of the algorithm's solution is at most α times the cost of an optimal solution for the instance.

A standard algorithm that performs provably well for several elementary routing problems is the so-called work function algorithm [2, 5, 8]; after each request, the algorithm moves to a configuration with low cost and which is not too far from the current configuration. More precisely, if the system's configuration after serving a sequence σ is $\mathcal{C}$ and $r \subseteq \mathbb{M}$ is the next request, then the work function algorithm with parameter $\lambda \geq 1$ moves to a configuration $\mathcal{C}' \in r$ that minimizes

$$\lambda W_{\sigma,r}(\mathcal{C}') + d(\mathcal{C}, \mathcal{C}'),$$

where $d(\mathcal{C}, \mathcal{C}')$ is the distance between configurations $\mathcal{C}$ and $\mathcal{C}'$, and $W_{\sigma,r}(\mathcal{C}')$ is the cost of an optimal solution that serves all requests (in order) in σ plus request r with the restriction that it ends in configuration $\mathcal{C}'$.

Key Results

The main result in [11] is a sufficient condition for a metrical service system to have a constant-competitive algorithm. Additionally, the authors show that this condition holds for the generalized two-server problem.

For a fixed metrical service system $\mathcal{S}$ with metric space $\mathbb{M}$, denote by $A(\mathcal{C}, \sigma)$ the cost of algorithm A on input sequence σ, starting in configuration $\mathcal{C}$. Let $\mathrm{OPT}(\mathcal{C}, \sigma)$ be the cost of the corresponding optimal solution. We say that a path T in $\mathbb{M}$ *serves* a sequence σ if it visits all requests in order. Hence, a feasible path is a path that serves the sequence and starts in the initial configuration.

Paths T_1 and T_2 are said to be *independent* if they are far apart in the following way: $|T_1| + |T_2| < d(\mathcal{C}_1^g, \mathcal{C}_1^t) + d(\mathcal{C}_1^g, \mathcal{C}_1^t)$, where $\mathcal{C}_i^g$ and $\mathcal{C}_i^t$ are, respectively, the start and end point of path $T_i (i \in \{1,2\})$. Notice, for example, that two intersecting paths are not independent.

Theorem 1 *Let $\mathcal{S}$ be a metrical service system with metric space $\mathbb{M}$. Suppose there exists an algorithm A and constants $\alpha \geq 1$, $\beta \geq 0$, and $m \geq 2$ such that for any point $\mathcal{C} \in \mathbb{M}$, sequence σ and pairwise independent paths $T_1, T_2, \ldots, T_m$ that serve σ*

$$A(\mathcal{C}, \sigma) \leq \alpha \mathrm{OPT}(\mathcal{C}, \sigma) + \beta \sum_{i=1}^{m} |T_i|. \quad (1)$$

Then there exists an algorithm B that is constant competitive for $\mathcal{S}$.

The proof in [11] of the theorem above provides an explicit formulation of B. This algorithm combines algorithm A with the work function algorithm and operates in phases. In each phase, it applies algorithm A until its cost becomes too large compared to the optimal cost. Then, it makes one step of the work function algorithm and a new phase starts. In each phase, algorithm A makes a restart, that is, it takes the final configuration of the previous phase as the initial configuration, whereas the work function algorithm remembers the whole request sequence.

For the generalized two-server problem the so-called balance algorithm satisfies condition (1). This algorithm stores the cumulative costs of the two servers and with each request it moves the server that minimizes the maximum of the two new values. The balance algorithm itself is not constant competitive but Theorem 1 says that, if we combine it in a clever way with the work function algorithm, then we get an algorithm that is constant competitive.

Applications

A set of metrical service systems can be combined to get what is called in [9] the *sum system*. A request of the sum system consists of one request for each system, and to serve it we need to serve at least one of the individual requests. The generalized two-server problem should be considered as one of the simplest sum systems since the two individual problems are completely trivial: There is one server and each request consists of a single point.

Sum systems are particularly interesting to model systems for information storage and retrieval. To increase stability or efficiency, one may store copies of the same information in multiple systems (e.g., databases, hard disks). To retrieve one piece of information, we may read it from any system. However, to read information it may be necessary to change the configuration of the system. For example, if the database is stored in a binary search tree, then it is efficient to make online changes to the structure of the tree, that is, to use dynamic search trees [12].

Open Problems

A proof that the work function algorithm is competitive for the generalized two-server problem (as conjectured in [9] and [11]) is still lacking.

Also, a randomized algorithm with a smaller competitive ratio than that of [11] is not known. No results (except for a lower bound) are known for the generalized problem with more than two servers. It is not even clear if the work function algorithm may be competitive here.

There are systems for which the work function algorithm is not competitive. It would be interesting to have a nontrivial property that implies competitiveness of the work function algorithm.

Cross-References

- ▸ Algorithm DC-TREE for k-Servers on Trees
- ▸ Metrical Task Systems
- ▸ Online Paging and Caching
- ▸ Work-Function Algorithm for k-Servers

Recommended Reading

1. Borodin A, El-Yaniv R (1998) Online computation and competitive analysis. Cambridge University Press, Cambridge
2. Burley WR (1996) Traversing layered graphs using the work function algorithm. J Algorithms 20:479–511
3. Chrobak M (2003) Sigact news online algorithms column 1. ACM SIGACT News 34:68–77
4. Chrobak M, Larmore LL (1992) Metrical service systems: deterministic strategies. Tech. Rep. UCR-CS-93-1, Department of Computer Science, University of California at Riverside
5. Chrobak M, Sgall J (2004) The weighted 2-server problem. Theor Comput Sci 324:289–312
6. Chrobak M, Karloff H, Payne TH, Vishwanathan S (1991) New results on server problems. SIAM J Discret Math 4:172–181
7. Fiat A, Ricklin M (1994) Competitive algorithms for the weighted server problem. Theor Comput Sci 130:85–99
8. Koutsoupias E, Papadimitriou CH (1995) On the k-server conjecture. J ACM 42:971–983
9. Koutsoupias E, Taylor DS (2004) The CNN problem and other k-server variants. Theor Comput Sci 324:347–359
10. Manasse MS, McGeoch LA, Sleator DD (1990) Competitive algorithms for server problems. J Algorithms **11**, 208–230
11. Sitters RA, Stougie L (2006) The generalized two-server problem. J ACM 53:437–458
12. Sleator DD, Tarjan RE (1985) Self-adjusting binary search trees. J ACM 32:652–686

Generalized Vickrey Auction

Makoto Yokoo
Department of Information Science and Electrical Engineering, Kyushu University, Nishi-ku, Fukuoka, Japan

Keywords

Generalized Vickrey auction; GVA; VCG; Vickrey–Clarke–Groves mechanism

Years and Authors of Summarized Original Work

1995; Varian

Problem Definition

Auctions are used for allocating goods, tasks, resources, etc. Participants in an auction include an auctioneer (usually a seller) and bidders (usually buyers). An auction has well-defined rules that enforce an agreement between the auctioneer and the winning bidder. Auctions are often used when a seller has difficulty in estimating the value of an auctioned good for buyers.

The Generalized Vickrey Auction protocol (GVA) [5] is an auction protocol that can be used for combinatorial auctions [3] in which multiple items/goods are sold simultaneously. Although conventional auctions sell a single item at a time, combinatorial auctions sell multiple items/goods. These goods may have interdependent values, e.g., these goods are complementary/substitutable and bidders can bid on any combination of goods. In a combinatorial auction, a bidder can express complementary/substitutable preferences over multiple bids. By taking into account complementary/substitutable preferences, the participants' utilities and the revenue of the seller can be increased. The GVA is one instance of the Clarke mechanism [2, 4]. It is also called the Vickrey–Clarke–Groves mechanism (VCG). As

its name suggests, it is a generalized version of the well-known Vickrey (or second-price) auction protocol [6], proposed by an American economist W. Vickrey, a 1996 Nobel Prize winner.

Assume there is a set of bidders $N = \{1, 2, \ldots, n\}$ and a set of goods $M = \{1, 2, \ldots, m\}$. Each bidder i has his/her preferences over a bundle, i.e., a subset of goods $B \subseteq M$. Formally, this can be modeled by supposing that bidder i privately observes a parameter, or signal, θ_i, which determines his/her preferences. The parameter θ_i is called the *type* of bidder i. A bidder is assumed to have a *quasilinear*, *private value* defined as follows.

Definition 1 (Utility of a Bidder) The utility of bidder i, when i obtains $B \subseteq M$ and pays p_i, is represented as $v(B, \theta_i) - p_i$.

Here, the valuation of a bidder is determined independently of other bidders' valuations. Also, the utility of a bidder is linear in terms of the payment. Thus, this model is called a quasilinear, private value model.

Definition 2 (Incentive Compatibility) An auction protocol is (dominant-strategy) *incentive compatible* (or *strategy-proof*) if declaring the true type/evaluation values is a dominant strategy for each bidder, i.e., an optimal strategy regardless of the actions of other bidders.

A combination of dominant strategies of all bidders is called a *dominant-strategy equilibrium*.

Definition 3 (Individual Rationality) An auction protocol is *individually rational* if no participant suffers any loss in a dominant-strategy equilibrium, i.e., the payment never exceeds the evaluation value of the obtained goods.

Definition 4 (Pareto Efficiency) An auction protocol is *Pareto efficient* when the sum of all participants' utilities (including that of the auctioneer), i.e., the social surplus, is maximized in a dominant-strategy equilibrium.

The goal is to design an auction protocol that is incentive compatible, individually rational, and Pareto efficient. It is clear that individual rationality and Pareto efficiency are desirable. Regarding the incentive compatibility, the *revelation principle* states that in the design of an auction protocol, it is possible to restrict attention only to incentive compatible protocols without loss of generality [4]. In other words, if a certain property (e.g., Pareto efficiency) can be achieved using some auction protocol in a dominant-strategy equilibrium, then the property can also be achieved using an incentive-compatible auction protocol.

Key Results

A *feasible* allocation is defined as a vector of n bundles $\vec{B} = \langle B_1, \ldots, B_n \rangle$, where $\bigcup_{j \in N} B_j \subseteq M$ and for all $j \neq j'$, $B_j \cap B_{j'} = \emptyset$ hold.

The GVA protocol can be described as follows.

1. Each bidder i declares his/her type $\hat{\theta}_i$, which can be different from his/her true type θ_i.
2. The auctioneer chooses an optimal allocation $\vec{B}^*$ according to the declared types. More precisely, the auctioneer chooses $\vec{B}^*$ defined as follows:

$$\vec{B}^* = \arg\max_{\vec{B}} \sum_{j \in N} v\left(B_j, \hat{\theta}_j\right).$$

3. Each bidder i pays p_i, which is defined as follows ($B_j^{\sim i}$ and B_j^* are the jth element of $\vec{B}^{\sim i}$ and $\vec{B}^*$, respectively):

$$p_i = \sum_{j \in N \setminus \{i\}} v\left(B_j^{\sim i}, \hat{\theta}_j\right) - \sum_{j \in N \setminus \{i\}} v\left(B_j^*, \hat{\theta}_j\right),$$
$$\text{where } \vec{B}^{\sim i} = \arg\max_{\vec{B}} \sum_{j \in N \setminus \{i\}} v\left(B_j, \hat{\theta}_j\right). \tag{1}$$

The first term in Eq. (1) is the social surplus when bidder i does not participate. The second term is the social surplus except bidder i when i does participate. In the GVA, the payment of bidder i can be considered as the decreased amount of

the other bidders' social surplus resulting from his/her participation.

A description of how this protocol works is given below.

Example 1 Assume there are two goods a and b, and three bidders, 1, 2, and 3, whose types are θ_1, θ_2, and θ_3, respectively. The evaluation value for a bundle $v(B, \theta_i)$ is determined as follows.

	$\{a\}$	$\{b\}$	$\{a, b\}$
θ_1	\$6	\$0	\$6
θ_2	\$0	\$0	\$8
θ_3	\$0	\$5	\$5

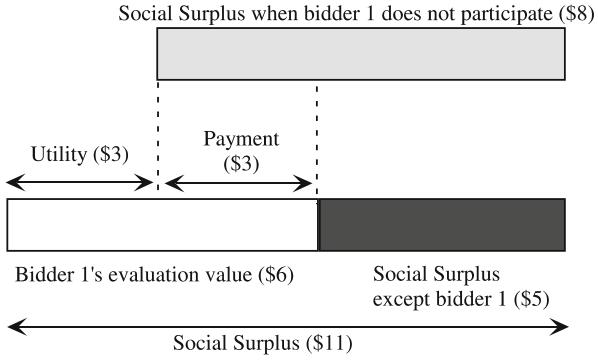
Here, bidder 1 wants good a only, and bidder 3 wants good b only. Bidder 2's utility is all-or-nothing, i.e., he/she wants both goods at the same time and having only one good is useless.

Assume each bidder i declares his/her true type θ_i. The optimal allocation is to allocate good a to bidder 1 and b to bidder 3, i.e., $\vec{B}^* = \langle \{a\}, \{\}, \{b\} \rangle$. The payment of bidder 1 is calculated as follows. If bidder 1 does not participate, the optimal allocation would have been allocating both items to bidder 2, i.e., $\vec{B}^{\sim 1} = \langle \{\}, \{a, b\}, \{\} \rangle$ and the social surplus, i.e., $\sum_{j \in N \setminus \{1\}} v\left(B_j^{\sim 1}, \hat{\theta}_j\right)$ is equal to \$8. When bidder 1 does participate, bidder 3 obtains $\{b\}$, and the social surplus except for bidder 1, i.e., $\sum_{j \in N \setminus \{1\}} v\left(B_j^*, \hat{\theta}_j\right)$, is 5. Therefore, bidder 1 pays the difference $\$8 - \$5 = \$3$. The obtained utility of bidder 1 is $\$6 - \$3 = \$3$. The payment of bidder 3 is calculated as $\$8 - \$6 = \$2$.

The intuitive explanation of why truth telling is the dominant strategy in the GVA is as follows. In the GVA, goods are allocated so that the social surplus is maximized. In general, the utility of society as a whole does not necessarily mean maximizing the utility of each participant. Therefore, each participant might have an incentive for lying if the group decision is made so that the social surplus is maximized.

However, the payment of each bidder in the GVA is cleverly determined so that the utility of each bidder is maximized when the social surplus is maximized. Figure 1 illustrates the relationship between the payment and utility of bidder 1 in Example 1. The payment of bidder 1 is defined as the difference between the social surplus when bidder 1 does not participate (i.e., the length of the upper shaded bar) and the social surplus except bidder 1 when bidder 1 does participate (the length of the lower black bar), i.e., $\$8 - \$5 = \$3$.

On the other hand, the utility of bidder 1 is the difference between the evaluation value of the obtained item and the payment, which equals $\$6 - \$3 = \$3$. This amount is equal to the difference between the total length of the lower bar and the upper bar. Since the length of the upper bar is determined independently of bidder 1's declaration, bidder 1 can maximize his/her utility by maximizing the length of the lower bar. However, the length of the lower bar represents the social surplus. Thus, bidder 1 can maximize his/her utility when the social surplus is maximized. Therefore, bidder 1 does not have an incentive for lying since the group decision is made so that the social surplus is maximized.



Generalized Vickrey Auction, Fig. 1 Utilities and Payments in the GVA

Theorem 1 *The GVA is incentive compatible.*

Proof Since the utility of bidder i is assumed to be quasilinear, it can be represented as

$$\begin{aligned} v(B_i, \theta_i) - p_i &= v(B_i, \theta_i) \\ &- \left[\sum_{j \in N \setminus \{i\}} v\left(B_j^{\sim i}, \hat{\theta}_j\right) - \sum_{j \in N \setminus \{i\}} v\left(B_j^*, \hat{\theta}_j\right) \right] \\ &= \left[v(B_i, \theta_i) + \sum_{j \in N \setminus \{i\}} v\left(B_j^*, \hat{\theta}_j\right) \right] \\ &- \sum_{j \in N \setminus \{i\}} v\left(B_j^{\sim i}, \hat{\theta}_j\right) \end{aligned} \tag{2}$$

The second term in Eq. (2) is determined independently of bidder i's declaration. Thus, bidder 1 can maximize his/her utility by maximizing the first term. However, $\vec{B}^*$ is chosen so that $\sum_{j \in N} v\left(B_j, \hat{\theta}_j\right)$ is maximized. Therefore, bidder i can maximize his/her utility by declaring $\hat{\theta}_i = \theta_i$, i.e., by declaring his/her true type. □

Theorem 2 *The GVA is individually rational.*

Proof This is clear from Eq. (2), since the first term is always larger than (or at least equal to) the second term. □

Theorem 3 *The GVA is Pareto efficient.*

Proof From Theorem 1, truth telling is a dominant-strategy equilibrium. From the way of choosing the allocation, the social surplus is maximized if all bidders declare their true types. □

Applications

The GVA can be applied to combinatorial auctions, which have lately attracted considerable attention [3]. The US Federal Communications Commission has been conductingauctions for allocating spectrum rights. Clearly, there exist interdependencies among the values of spectrum rights. For example, a bidder may desire licenses for adjoining regions simultaneously, i.e., these licenses are complementary. Thus, the spectrum auctions is a promising application field of combinatorial auctions and have been a major driving force for activating the research on combinatorial auctions.

Open Problems

Although the GVA has these good characteristics (Pareto efficiency, incentive compatibility, and individual rationality), these characteristics cannot be guaranteed when bidders can submit *false-name* bids. Furthermore, [1] pointed out several other limitations such as vulnerability to the collusion of the auctioneer and/or losers.

Also, to execute the GVA, the auctioneer must solve a complicated optimization problem. Various studies have been conducted to introduce search techniques, which were developed in the artificial intelligence literature, for solving this optimization problem [3].

Cross-References

▸ False-Name-Proof Auction

Recommended Reading

1. Ausubel LM, Milgrom PR (2002) Ascending auctions with package bidding. Front Theor Econ 1(1). Article 1
2. Clarke EH (1971) Multipart pricing of public goods. Public Choice 2:19–33
3. Cramton P, Steinberg R, Shoham Y (eds) (2005) Combinatorial auctions. MIT, Cambridge
4. Mas-Colell A, Whinston MD, Green JR (1995) Microeconomic theory. Oxford University Press, Oxford
5. Varian HR (1995) Economic mechanism design for computerized agents. In: Proceedings of the 1st Usenix workshop on electronic commerce
6. Vickrey W (1961) Counter speculation, auctions, and competitive sealed tenders. J Financ 16:8–37

Geographic Routing

Aaron Zollinger
Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA

Keywords

Directional routing; Geometric routing; Location-based routing; Position-based routing

Years and Authors of Summarized Original Work

2003; Kuhn, Wattenhofer, Zollinger

Problem Definition

Geographic routing is a type of routing particularly well suited for dynamic ad hoc networks. Sometimes also called directional, geometric, location-based, or position-based routing, it is based on two principal assumptions. First, it is assumed that every node knows its own and its network neighbors' positions. Second, the source of a message is assumed to be informed about the position of the destination. Geographic routing is defined on a Euclidean graph, that is a graph whose nodes are embedded in the Euclidean plane. Formally, geographic ad hoc routing algorithms can be defined as follows:

Definition 1 (Geographic Ad Hoc Routing Algorithm) Let $G = (V, E)$ be a Euclidean graph. The task of a geographic ad hoc routing algorithm $\mathcal{A}$ is to transmit a message from a source $s \in V$ to a destination $t \in V$ by sending packets over the edges of G while complying with the following conditions:

- All nodes $v \in V$ know their geographic positions as well as the geographic positions of all their neighbors in G.
- The source s is informed about the position of the destination t.
- The control information which can be stored in a packet is limited by $O(\log n)$ bits, that is, only information about a constant number of nodes is allowed.
- Except for the temporary storage of packets before forwarding, a node is not allowed to maintain any information.

Geographic routing is particularly interesting, as it operates without any routing tables whatsoever. Furthermore, once the position of the destination is known, all operations are strictly local, that is, every node is required to keep track only of its direct neighbors. These two factors – absence of necessity to keep routing tables up to date and independence of remotely occurring topology changes – are among the foremost reasons why geographic routing is exceptionally suitable for operation in ad hoc networks. Furthermore, in a sense, geographic routing can be considered a lean version of source routing appropriate for dynamic networks: While in source routing the complete hop-by-hop route to be followed by the message is specified by the source, in geographic routing the source simply addresses the message with the position of the destination. As the destination can generally be expected to move slowly compared to the frequency of topology changes between the source and the destination, it makes sense to keep track of the position of the destination instead of maintaining network topology information up to date; if the destination does not move too fast, the message is delivered regardless of possible topology changes among intermediate nodes.

The cost bounds presented in this entry are achieved on *unit disk graphs*. A unit disk graph is defined as follows:

Definition 2 (Unit Disk Graph) Let $V \subset \mathbb{R}^2$ be a set of points in the 2-dimensional plane. The graph with edges between all nodes with distance at most 1 is called the unit disk graph of V.

Unit disk graphs are often employed to model wireless ad hoc networks.

The routing algorithms considered in this entry operate on planar graphs, graphs that contain no two intersecting edges. There exist strictly local algorithms constructing such planar graphs given a unit disk graph. The edges of planar graphs partition the Euclidean plane into contiguous areas, so-called faces. The algorithms cited in this entry are based on these faces.

Key Results

The first geographic routing algorithm shown to always reach the destination was Face Routing introduced in [14].

Theorem 1 *If the source and the destination are connected, Face Routing executed on an arbitrary planar graph always finds a path to the destination. It thereby takes at most $O(n)$ steps, where n is the total number of nodes in the network.*

There exists however a geographic routing algorithm whose cost is bounded not only with respect to the total number of nodes, but in relation to the *shortest path* between the source and the destination: The GOAFR$^+$ algorithm [15, 16, 18, 24] (pronounced as "gopher-plus") combines *greedy routing* – where every intermediate node relays the message to be routed to its neighbor located nearest to the destination – with face routing. Together with the locally computable *Gabriel Graph* planarization technique, the effort expended by the GOAFR$^+$ algorithm is bounded as follows:

Theorem 2 *Let c be the cost of an optimal path from s to t in a given unit disk graph. GOAFR$^+$ reaches t with cost $O(c^2)$ if s and t are connected. If s and t are not connected, GOAFR$^+$ reports so to the source.*

On the other hand it can be shown that – on certain worst-case graphs – no geographic routing algorithm operating in compliance with the above definition can perform asymptotically better than GOAFR$^+$:

Theorem 3 *There exist graphs where any deterministic (randomized) geographic ad hoc routing algorithm has (expected) cost $\Omega(c^2)$.*

This leads to the following conclusion:

Theorem 4 *The cost expended by GOAFR$^+$ to reach the destination on a unit disk graph is asymptotically optimal.*

In addition, it has been shown that the GOAFR$^+$ algorithm is not only guaranteed to have low worst-case cost but that it also performs well in average-case networks with nodes randomly placed in the plane [15, 24].

Applications

By its strictly local nature geographic routing is particularly well suited for application in potentially highly dynamic wireless ad hoc networks. However, also its employment in dynamic networks in general is conceivable.

Open Problems

A number of problems related to geographic routing remain open. This is true above all with respect to the dissemination within the network of information about the destination position and on the other hand in the context of node mobility as well as network dynamics. Various approaches to these problems have been described in [7] as well as in chapters 11 and 12 of [24]. More generally, taking geographic routing one step further towards its application in practical wireless ad hoc networks [12, 13] is a field yet largely open. A more specific open problem is finally posed by the question whether geographic routing can be adapted to networks with nodes embedded in three-dimensional space.

Experimental Results

First experiences with geographic and in particular face routing in practical networks have

been made [12, 13]. More specifically, problems in connection with graph planarization that can occur in practice were observed, documented, and tackled.

Cross-References

- Local Computation in Unstructured Radio Networks
- Planar Geometric Spanners
- Routing in Geometric Networks

Recommended Reading

1. Barrière L, Fraigniaud P, Narayanan L (2001) Robust position-based routing in wireless ad hoc networks with unstable transmission ranges. In: Proceedings of the 5th international workshop on discrete algorithms and methods for mobile computing and communications (DIAL-M). ACM Press, New York, pp 19–27
2. Bose P, Brodnik A, Carlsson S, Demaine E, Fleischer R, López-Ortiz A, Morin P, Munro J (2000) Online routing in convex subdivisions. In: International symposium on algorithms and computation (ISAAC). LNCS, vol 1969. Springer, Berlin/New York, pp 47–59
3. Bose P, Morin P (1999) Online routing in triangulations. In: Proceedings of 10th internatrional symposium on algorithms and computation (ISAAC). LNCS, vol 1741. Springer, Berlin, pp 113–122
4. Bose P, Morin P, Stojmenovic I, Urrutia J (1999) Routing with guaranteed delivery in ad hoc wireless networks. In: Proceedings of the 3rd international workshop on discrete algorithms and methods for mobile computing and communications (DIAL-M), pp 48–55
5. Datta S, Stojmenovic I, Wu J (2002) Internal node and shortcut based routing with guaranteed delivery in wireless networks. In: Cluster computing, vol 5. Kluwer Academic, Dordrecht, pp 169–178
6. Finn G (1987) Routing and addressing problems in large metropolitan-scale internetworks. Technical report ISI/RR-87–180, USC/ISI
7. Flury R, Wattenhofer R (2006) MLS: an efficient location service for mobile ad hoc networks. In: Proceedings of the 7th ACM international symposium on mobile ad-hoc networking and computing (MobiHoc), Florence
8. Fonseca R, Ratnasamy S, Zhao J, Ee CT, Culler D, Shenker S, Stoica I (2005) Beacon vector routing: scalable point-to-point routing in wireless sensornets. In: 2nd symposium on networked systems design & implementation (NSDI), Boston
9. Gao J, Guibas L, Hershberger J, Zhang L, Zhu A (2001) Geometric spanner for routing in mobile networks. In: Proceedings of 2nd ACM international symposium on mobile ad-hoc networking and computing (MobiHoc), Long Beach
10. Hou T, Li V (1986) Transmission range control in multihop packet radio networks. IEEE Trans Commun 34:38–44
11. Karp B, Kung H (2000) GPSR: greedy perimeter stateless routing for wireless networks. In: Proceedings of 6th annual international conference on mobile computing and networking (MobiCom), pp 243–254
12. Kim YJ, Govindan R, Karp B, Shenker S (2005) Geographic routing made practical. In: Proceedings of the second USENIX/ACM symposium on networked system design and implementation (NSDI 2005), Boston
13. Kim YJ, Govindan R, Karp B, Shenker S (2005) On the pitfalls of geographic face routing. In: Proceedings of the ACM joint workshop on foundations of mobile computing (DIALM-POMC), Cologne
14. Kranakis E, Singh H, Urrutia J (1999) Compass routing on geometric networks. In: Proceedings of 11th Canadian conference on computational geometry, Vancouver, pp 51–54
15. Kuhn F, Wattenhofer R, Zhang Y, Zollinger A (2003) Geometric routing: of theory and practice. In: Proceedings of the 22nd ACM symposium on the principles of distributed computing (PODC)
16. Kuhn F, Wattenhofer R, Zollinger A (2002) Asymptotically optimal geometric mobile ad-hoc routing. In: Proceedings of 6th international workshop on discrete algorithms and methods for mobile computing and communications (Dial-M). ACM, New York, pp 24–33
17. Kuhn F, Wattenhofer R, Zollinger A (2003) Ad-hoc networks beyond unit disk graphs. In: 1st ACM joint workshop on foundations of mobile computing (DIALM-POMC), San Diego
18. Kuhn F, Wattenhofer R, Zollinger A (2003) Worst-case optimal and average-case efficient geometric ad-hoc routing. In: Proceedings of 4th ACM international symposium on mobile ad-hoc networking and computing (MobiHoc)
19. Leong B, Liskov B, Morris R (2006) Geographic routing without planarization. In: 3rd symposium on networked systems design & implementation (NSDI), San Jose
20. Leong B, Mitra S, Liskov B (2005) Path vector face routing: geographic routing with local face information. In: 13th IEEE international conference on network protocols (ICNP), Boston
21. Takagi H, Kleinrock L (1984) Optimal transmission ranges for randomly distributed packet radio terminals. IEEE Trans Commun 32:246–257
22. Urrutia J (2002) Routing with guaranteed delivery in geometric and wireless networks. In: Stojmenovic I (ed) Handbook of wireless networks and mobile computing, ch. 18. Wiley, Hoboken, pp. 393–406

23. Wattenhofer M, Wattenhofer R, Widmayer P (2005) Geometric routing without geometry. In: 12th colloquium on structural information and communication complexity (SIROCCO), Le Mont Saint-Michel
24. Zollinger A (2005) Networking unleashed: geographic routing and topology control in ad hoc and sensor networks. PhD thesis, ETH Zurich, Switzerland Dissertation, ETH 16025

Geometric Approaches to Answering Queries

Aleksandar Nikolov
Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Keywords

Convex geometry; Convex optimization; Differential privacy; Query release

Years and Authors of Summarized Original Work

2014; Nikolov, Talwar, Zhang
2015; Dwork, Nikolov, Talwar

Problem Definition

The central problem of private data analysis is to extract meaningful information from a statistical database without revealing too much about any particular individual represented in the database. Here, by a *statistical database*, we mean a multiset $D \in \mathcal{X}^n$ of n rows from the *data universe* $\mathcal{X}$. The notation $|D| \triangleq n$ denotes the *size* of the database. Each row represents the information belonging to a single individual. The universe $\mathcal{X}$ depends on the domain. A natural example to keep in mind is $\mathcal{X} = \{0,1\}^d$, i.e., each row of the database gives the values of d binary attributes for some individual.

Differential privacy formalizes the notion that an adversary should not learn too much about any individual as a result of a private computation. The formal definition follows.

Definition 1 ([8]) A randomized algorithm $\mathcal{A}$ satisfies (ε, δ)-differential privacy if for any two databases D and D' that differ in at most a single row (i.e., $|D \triangle D'| \leq 1$), and any measurable event S in the range of $\mathcal{A}$,

$$\Pr[\mathcal{A}(D) \in S] \leq e^{\varepsilon} \Pr[\mathcal{A}(D') \in S] + \delta.$$

Above, probabilities are taken over the internal coin tosses of $\mathcal{A}$.

Differential privacy guarantees to a data owner that allowing her data to be used for analysis does not risk much more than she would if she did not allow her data to be used.

In the sequel, we shall call databases D and D' that differ in a single row *neighboring databases*, denoted $D \sim D'$. Usually, the parameter ε is set to be a small constant so that $e^{\varepsilon} \approx 1 + \varepsilon$, and δ is set to be no bigger than n^{-2} or even $n^{-\omega(1)}$. The case of $\delta = 0$ often requires different techniques from the case $\delta > 0$; as is common in the literature, we shall call the two cases *pure differential privacy* and *approximate differential privacy*.

Query Release

In the query release problem, we are given a set $\mathcal{Q}$ of queries, where each $q \in \mathcal{Q}$ is a function $q : \mathcal{X}^n \to \mathbb{R}$. Our goal is to design a differentially private algorithm $\mathcal{A}$ which takes as input a database D and outputs a list of answers to the queries in $\mathcal{Q}$. We shall call such an algorithm a *(query answering) mechanism*. Here, we treat the important special case of query release for sets of *linear queries*. A linear query q is specified by a function $q : \mathcal{X} \to [-1, 1]$, and, slightly abusing notation, we define the value of the query as $q(D) \triangleq \sum_{x \in D} q(e)$. When $q : \mathcal{X} \to \{0,1\}$ is a predicate, $q(D)$ is a *counting query*: it simply counts the number of rows of D that satisfy the predicate.

It is easy to see that a differentially private algorithm (with any reasonable choice of ε and δ) cannot answer a nontrivial set of queries exactly. For this reason, we need to have a measure of error, and here we introduce the two most commonly used ones: average and worst-case error. Assume that on an input database D and a set of

linear queries $\mathcal{Q}$, the algorithm $\mathcal{A}$ gives answer $\tilde{q}^{\mathcal{A}}(D)$ for query $q \in \mathcal{Q}$. The *average error*of $\mathcal{A}$ on the query set $\mathcal{Q}$ for databases of size at most n is equal to

$$\mathrm{err}_{\mathrm{avg}}(\mathcal{A}, \mathcal{Q}, n) \triangleq \max_{D:|D|\leq n} \sqrt{\frac{1}{|\mathcal{Q}|}\mathbb{E}\left[\sum_{q\in\mathcal{Q}} |\tilde{q}^{\mathcal{A}}(D) - q(D)|^2\right]}.$$

The *worst-case error* is equal to

$$\mathrm{err}_{\mathrm{wc}}(\mathcal{A}, \mathcal{Q}, n) \triangleq \max_{D:|D|\leq n} \mathbb{E} \max_{q\in\mathcal{Q}} |\tilde{q}^{\mathcal{A}}(D) - q(D)|.$$

In both definitions above, expectations are taken over the coin throws of $\mathcal{A}$. We also define $\mathrm{err}_{\mathrm{avg}}(\mathcal{A}, \mathcal{Q}) = \sup_n \mathrm{err}_{\mathrm{avg}}(\mathcal{A}, \mathcal{Q}, n)$, and respectively $\mathrm{err}_{\mathrm{wc}}(\mathcal{A}, \mathcal{Q}) = \sup_n \mathrm{err}_{\mathrm{wc}}(\mathcal{A}, \mathcal{Q}, n)$, to be the maximum error over all database sizes. The objective in the query release problem is to *minimize error subject to privacy constraints*.

Marginal Queries

An important class of counting queries are the *marginal queries*. A k-way marginal query $\mathrm{mar}_{S,\alpha} : \{0,1\}^d \to \{0,1\}$ is specified by a subset of attributes $S \subseteq \{1, \ldots, d\}$ of size k and a vector $\alpha \in \{0,1\}^S$. The query evaluates to 1 on those rows that agree with α on all attributes in S, i.e., $\mathrm{mar}_{S,\alpha}(\chi) = \bigwedge_{i\in S} \chi_i = \alpha_i$ for any $\chi \in \{0,1\}^d$. Recall that, using the notation we introduced above, this implies that $\mathrm{mar}_{S,\alpha}(D)$ counts the number of rows in the database D that agree with α on S. Marginal queries capture contingency tables in statistics and OLAP cubes in databases. They are widely used in the sciences and are released by a number of official agencies.

Matrix Notation

It will be convenient to encode the query release problem for linear queries using matrix notation. A common and very useful representation of a database $D \in \mathcal{X}^n$ is the *histogram representation*: the histogram of D is a vector $x \in \mathbb{P}^{\mathcal{X}}$ ($\mathbb{P}$ is the set of nonnegative integers) such that for any $\chi \in \mathcal{X}$, x_χ is equal to the number of copies of χ in D. Notice that $\|x\|_1 = n$ and also that if x and x' are, respectively, the histograms of two neighboring databases D and D', then $\|x - x'\|_1 \leq 1$ (here $\|x\|_1 = \sum_\chi |x_\chi|$ is the standard ℓ_1 norm). Linear queries are a linear transformation of x. More concretely, let us define the *query matrix* $A \in [-1,1]^{\mathcal{Q}\times\mathcal{X}}$ associated with a set of linear queries $\mathcal{Q}$ by $a_{q,\chi} = q(\chi)$. Then it is easy to see that the vector Ax gives the answers to the queries $\mathcal{Q}$ on a database D with histogram x.

Key Results

A central object of study in geometric approaches to the query release problem is a convex body associated with a set of linear queries. Before introducing some of the main results and algorithms, we define this body.

The Sensitivity Polytope

Let A be the query matrix for some set of queries $\mathcal{Q}$, and let x and x' be the histograms of two neighboring databases, respectively, D and D'. Above, we observed that $D \sim D'$ implies that $\|x - x\|_1 \leq 1$. Let us use the notation $B_1^{\mathcal{X}} \triangleq \{x : \|x\|_1 \leq 1\}$ for the unit ball of the ℓ_1 norm in $\mathbb{R}^{\mathcal{X}}$. Then, $Ax - Ax' \in K_{\mathcal{Q}}$, where

$$K_{\mathcal{Q}} \triangleq \{Ax : \|x\|_1 \leq 1\} = A \cdot B_1^{\mathcal{X}}$$

is the *sensitivity polytope* associated with $\mathcal{Q}$. In other words, the sensitivity polytope is the smallest convex body such that $Ax' \in Ax + K_{\mathcal{Q}}$ for any histogram x and any histogram x' of a neighboring database. In this sense, $K_{\mathcal{Q}}$ describes how the answers to the queries $\mathcal{Q}$ can change between neighboring databases, which motivates the terminology. Informally, a differentially private algorithm must "hide" where in $Ax + K_{\mathcal{Q}}$ the true query answers are.

Another very useful property of the sensitivity polytope is that the vector Ax of query answers for any database of size at most n is contained in $n \cdot K_{\mathcal{Q}} = \{Ax : \|x\|_1 \leq n\}$.

Geometrically, $K_{\mathcal{Q}}$ is a convex polytope in $\mathbb{R}^{\mathcal{Q}}$, centrally symmetric around 0, i.e., $K_{\mathcal{Q}} = -K_{\mathcal{Q}}$. It is the convex hull of the points $\{\pm a_\chi : \chi \in \mathcal{X}\}$, where a_χ is the column of A indexed by the universe element χ, i.e., $a_\chi = (q(\chi))_{q \in \mathcal{Q}}$.

The sensitivity polytope was introduced by Hardt and Talwar [12]. The name was suggested by Li Zhang.

The Generalized Gaussian Mechanism

We mentioned informally that a differentially private mechanism must hide where in $Ax + K_{\mathcal{Q}}$ the true query answers lie. A simple formalization of this intuition is the Gaussian Mechanism, which we present here in a generalized geometric variant.

Recall that an *ellipsoid* in $\mathbb{R}^m$ is an affine transformation $F \cdot B_2^m + y$ of the unit Euclidean ball $B_2^m \triangleq \{x \in \mathbb{R}^m : \|x\|_2 \leq 1\}$ ($\|x\|_2$ is the usual Euclidean, i.e., ℓ_2 norm). In this article, we will only consider centrally symmetric ellipsoids, i.e., ellipsoids of the form $E = F \cdot B_2^m$.

Algorithm 1: Generalized Gaussian Mechanism $\mathcal{A}_E$

Input: *(Public)* Query set $\mathcal{Q}$; ellipsoid $E = F \cdot B_2^{\mathcal{Q}}$ such that $K_{\mathcal{Q}} \subseteq E$.
Input: *(Private)* Database D.
Sample a vector $g \sim N(0, c_{\varepsilon,\delta}^2)^{\mathcal{Q}}$, where $c_{\varepsilon,\delta} = \frac{0.5\sqrt{\varepsilon} + \sqrt{2\ln(1/\delta)}}{\varepsilon}$;
Compute the query matrix A and the histogram x for the database D;
Output: Vector of query answers $Ax + Fg$.

The *generalized Gaussian mechanism* $\mathcal{A}_E$ is shown as Algorithm 1. The notation $g \sim N(0, c_{\varepsilon,\delta}^2)^{\mathcal{Q}}$ means that each coordinate of g is an independent Gaussian random variable with mean 0 and variance $c_{\varepsilon,\delta}^2$. In the special case, when the ellipsoid E is just the Euclidean ball $\Delta_2 \cdot B_2^{\mathcal{Q}}$, with radius equal to the diameter $\Delta_2 \triangleq \max_{y \in K_{\mathcal{Q}}} \|y\|_2$ of $K_{\mathcal{Q}}$, $\mathcal{A}_E$ is the well-known Gaussian mechanism, whose privacy was analyzed in [6–8]. The diameter Δ_2 is also known as the ℓ_2-*sensitivity* of $\mathcal{Q}$ and for linear queries is always upper bounded by $\sqrt{|\mathcal{Q}|}$. The privacy of the generalized version is an easy corollary of the privacy of the standard Gaussian mechanism (see [16] for a proof).

Theorem 1 ([6–8, 16]) *For any ellipsoid E containing $K_{\mathcal{Q}}$, $\mathcal{A}_E$ satisfies (ε, δ)-differential privacy.*

It is not hard to analyze the error of the mechanism $\mathcal{A}_E$. Let $E = F \cdot B_2^{\mathcal{Q}}$, and recall the Hilbert-Schmidt norm $\|F\|_{HS} = \sqrt{\mathrm{tr}(FF^\intercal)}$ and the 1-to-2 norm $\|F^\intercal\|_{1\to 2}$ which is equal to the largest ℓ_2 norm of any row of F. Geometrically, $\|F\|_{HS}$ is equal to the square root of the sum of squared major axis lengths of E, and $\|F^\intercal\|_{1\to 2}$ is equal to the largest ℓ_∞ norm of any point in E. We have the error bounds

$$\mathrm{err}_{\mathrm{avg}}(\mathcal{A}, \mathcal{Q}) = O(c_{\varepsilon,\delta}) \cdot \frac{1}{\sqrt{|\mathcal{Q}|}} \|F\|_{HS};$$

$$\mathrm{err}_{\mathrm{wc}}(\mathcal{A}, \mathcal{Q}) = O(c_{\varepsilon,\delta} \sqrt{\log |\mathcal{Q}|}) \cdot \|F^\intercal\|_{1\to 2}.$$

Surprisingly, for any query set $\mathcal{Q}$, there exists an ellipsoid E such that the generalized Gaussian noise mechanism $\mathcal{A}_E$ is nearly optimal *among all differentially private mechanisms* for $\mathcal{Q}$. In order to formulate the result, let us define $\mathrm{opt}_{\mathrm{avg}}^{\varepsilon,\delta}(\mathcal{Q})$ (respectively, $\mathrm{opt}_{\mathrm{wc}}^{\varepsilon,\delta}(\mathcal{Q})$) to be the infimum of $\mathrm{err}_{\mathrm{avg}}(\mathcal{A}, \mathcal{Q})$ (respectively, $\mathrm{err}_{\mathrm{wc}}(\mathcal{A}, \mathcal{Q})$) over all (ε, δ)-differentially private mechanisms $\mathcal{A}$.

Theorem 2 ([16]) *Let $E = F \cdot B_2^{\mathcal{Q}}$ be the ellipsoid that minimizes $\|F\|_{HS}$ over all ellipsoids E containing $K_{\mathcal{Q}}$. Then*

$$\mathrm{err}_{\mathrm{avg}}(\mathcal{A}_E, \mathcal{Q}) = O(\log |\mathcal{Q}| \sqrt{\log 1/\delta}) \cdot \mathrm{opt}_{\mathrm{avg}}^{\varepsilon,\delta}(\mathcal{Q}).$$

If $E = F \cdot B_2^{\mathcal{Q}}$ minimizes $\|F^\intercal\|_{1\to 2}$ subject to $K_{\mathcal{Q}} \subseteq E$, then

$$\mathrm{err}_{\mathrm{wc}}(\mathcal{A}_E, \mathcal{Q}) = O((\log |\mathcal{Q}|)^{3/2} \sqrt{\log 1/\delta}) \cdot \mathrm{opt}_{\mathrm{wc}}^{\varepsilon,\delta}(\mathcal{Q}).$$

Minimizing $\|F\|_{HS}$ or $\|F^\intercal\|_{1\to 2}$ subject to $K_{\mathcal{Q}} \subseteq F \cdot B_2^{\mathcal{Q}}$ is a convex minimization problem.

An optimal solution can be approximated to within any prescribed degree of accuracy in time polynomial in $|\mathcal{Q}|$ and $|\mathcal{X}|$ via the ellipsoid algorithm. In fact, more efficient solutions are available: both problems can be formulated as semidefinite programs and solved via interior point methods, or one can also use the Plotkin-Shmoys-Tardos framework [1, 17]. Algorithm $\mathcal{A}_E$ also runs in time polynomial in $n, |\mathcal{Q}|, |\mathcal{X}|$, since it only needs to compute the true query answers and sample $|\mathcal{Q}|$ many Gaussian random variables. Thus, Theorem 2 gives an efficient approximation to the optimal differentially private mechanism for any set of linear queries.

The near-optimal mechanisms of Theorem 2 are closely related to the matrix mechanism [13]. The matrix mechanism, given a set of queries $\mathcal{Q}$ with query matrix A, solves an optimization problem to find a strategy matrix M, then computes answers $\tilde{y}$ to the queries Mx using the standard Gaussian mechanism, and outputs $AM^{-1}\tilde{y}$. The generalized Gaussian mechanism $\mathcal{A}_E$ instantiated with ellipsoid $E = F \cdot B_2^{\mathcal{Q}}$ is equivalent to the matrix mechanism with strategy matrix $F^{-1}A$.

The proof of optimality for the generalized Gaussian mechanism is related to a fundamental geometric fact: if all ellipsoids containing a convex body are "large," then the body itself must be "large." In particular, if the sum of squared major axis lengths of any ellipsoid containing $K_{\mathcal{Q}}$ is large, then $K_{\mathcal{Q}}$ must contain a simplex of proportionally large volume. Moreover, this simplex is the convex hull of a subset of the contact points of $K_{\mathcal{Q}}$ with the optimal ellipsoid. Since the contact points must be vertices of $K_{\mathcal{Q}}$, and all vertices of $K_{\mathcal{Q}}$ are either columns of the query matrix A or their negations, this guarantees the existence of a submatrix of A with large determinant. Determinants of submatrices in turn bound $\mathrm{opt}_{\mathrm{avg}}^{\varepsilon,\delta}(\mathcal{Q})$ from below (this is a consequence of a connection between combinatorial discrepancy and privacy [15], and the determinant lower bound on discrepancy [14]). This phenomenon is related to the Restricted Invertibility Principle of Bourgain and Tzafriri [4] and was established for the closely related minimum volume ellipsoid by Vershynin [18].

The Gaussian noise mechanism can only provide approximate privacy guarantees: when $\delta = 0$, the noise variance scaling factor $c_{\varepsilon,\delta}$ is unbounded. The case of pure privacy requires different techniques. Nevertheless, $(\varepsilon, 0)$-differentially private algorithms with efficiency and optimality guarantees analogous to these in Theorem 2 are known [2,12]. They use a more complicated noise distribution. In the important special case when $K_{\mathcal{Q}}$ is "well rounded" (technically, when $K_{\mathcal{Q}}$ is isotropic), the noise vector is sampled uniformly from $r \cdot K_{\mathcal{Q}}$, where r is a Γ-distributed random variable. Optimality is established conditional on the Hyperplane Conjecture [12] or unconditionally using Klartag's proof of an isomorphic version of the conjecture [2].

The Projection Mechanism

Despite the near-optimality guarantees, the generalized Gaussian mechanism has some drawbacks that can limit its applicability. One issue is that in some natural scenarios, the universe size $|\mathcal{X}|$ can be huge, and running time linear in $|\mathcal{X}|$ is impractical. Another is that its error is sometimes larger even than the database size, making the query answers unusable. We shall see that a simple modification of the Gaussian mechanism, based on an idea from statistics, goes a long way towards addressing these issues.

It is known that there exist sets of linear queries $\mathcal{Q}$ for which $\mathrm{opt}_{\mathrm{avg}}^{\varepsilon,\delta}(\mathcal{Q}) = \Omega(\sqrt{|\mathcal{Q}|})$ for any small enough constant ε and δ [6, 9]. However, this lower bound only holds for large databases, and algorithms with significantly better error guarantees are known when $n = o(|\mathcal{Q}|)$ [3, 10, 11]. We now know that there are (ε, δ)-differentially private algorithms that answer any set $\mathcal{Q}$ of linear queries on any database $D \in \mathcal{X}^n$ with average error at most

$$O\left(\frac{\sqrt{n}(\log|\mathcal{X}|)^{1/4}(\log 1/\delta)^{1/4}}{\sqrt{\varepsilon}}\right). \quad (1)$$

Moreover, for k-way marginal queries, this much error is necessary, up to factors logarithmic in n and $|\mathcal{Q}|$ [5]. Here, we describe a simple geometric algorithm from [16] that achieves this error bound for any $\mathcal{Q}$.

We know that for any set of queries $\mathcal{Q}$ and a database of size n, the true query answers are between 0 and n. Therefore, it is always safe to take the noisy answers $\tilde{y}$ output by the Gaussian mechanism and truncate them inside the interval $[0, n]$. However, we can do better by using knowledge of the query set $\mathcal{Q}$. For any database D of size n, the true query answers $y = Ax$ lie in $n \cdot K_{\mathcal{Q}}$. This suggests a regression approach: find the vector of answers $\hat{y} \in n \cdot K_{\mathcal{Q}}$ which is closest to the noisy output $\tilde{y}$ from the Gaussian mechanism. This is the main insight used in the projection mechanism (Algorithm 2).

Algorithm 2: Projection Mechanism $\mathcal{A}_{\text{proj}}$

Input: *(Public)* Query set $\mathcal{Q}$;
Input: *(Private)* Database $D \in \mathcal{X}^n$.
Compute a noisy vector of query answers $\tilde{y}$ with $\mathcal{A}_E$ for $E = \sqrt{|\mathcal{Q}|} \cdot B_2^{\mathcal{Q}}$.
Compute a projection $\hat{y}$ of $\tilde{y}$ onto $n \cdot K_{\mathcal{Q}}$:
$\hat{y} \triangleq \arg\min_{\hat{y} \in n \cdot K_{\mathcal{Q}}} \|\tilde{y} - \hat{y}\|_2$.
Output: Vector of answers $\hat{y}$.

The fact that the projection mechanism is (ε, δ)-differentially private is immediate, because its only interaction with the database is via the (ε, δ)-differentially private Gaussian mechanism, and post-processing cannot break differential privacy.

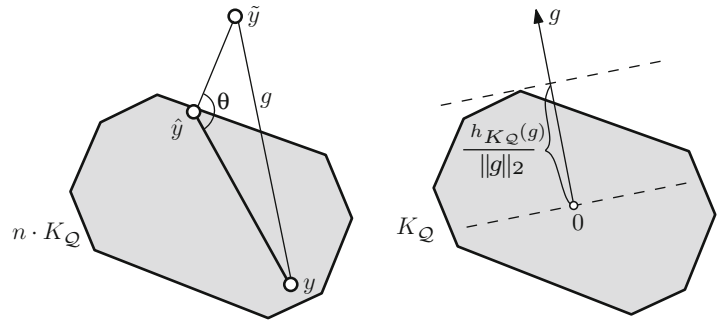
The projection step in $\mathcal{A}_{\text{proj}}$ reduces the noise significantly when $n = o(|\mathcal{Q}|/\varepsilon)$. Intuitively, in this case, $n \cdot K_{\mathcal{Q}}$ is small enough so that projection cancels a significant portion of the noise. Let us sketch the analysis. Let $y = Ax$ be the true query answers and $g = \tilde{y} - y$ the Gaussian noise vector. A simple geometric argument shows that $\|y - \hat{y}\|_2^2 \le 2|\langle y - \hat{y}, g\rangle|$: the main observation is that in the triangle formed by y, $\tilde{y}$, and $\hat{y}$, the angle at $\hat{y}$ is an obtuse or right angle; see Fig. 1. Since $\hat{y} \in n \cdot K_{\mathcal{Q}}$, there exists some histogram vector $\hat{x}$ with $\|\hat{x}\|_1 \le n$ such that $\hat{y} = A\hat{x}$. We can rewrite the inner product $\langle y - \hat{y}, g\rangle$ as $\langle x - \hat{x}, A^{\mathsf{T}} g\rangle$. Now, we apply Hölder's inequality and get

$$\begin{aligned} \mathbb{E}_g \|y - \hat{y}\|_2^2 &\le 2\mathbb{E}_g |\langle x - \hat{x}, A^{\mathsf{T}} g\rangle| \\ &\le 2\mathbb{E}_g \|x - \hat{x}\|_1 \|A^{\mathsf{T}} g\|_\infty \\ &\le 4n\mathbb{E}_g \|A^{\mathsf{T}} g\|_\infty. \end{aligned} \tag{2}$$

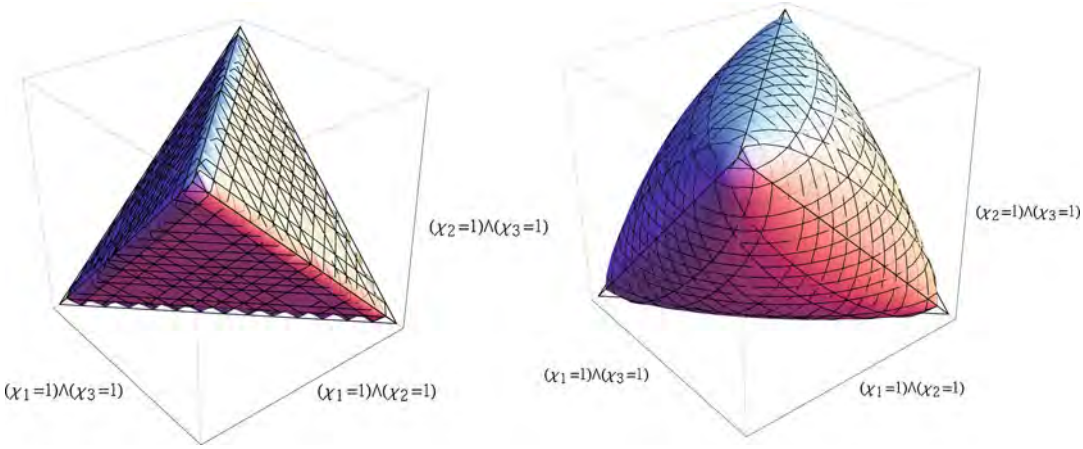
The term $\mathbb{E}_g \|A^{\mathsf{T}} g\|_\infty$ is the expected maximum of $|\mathcal{X}|$ Gaussian random variables, each with mean 0 and variance $c_{\varepsilon,\delta}^2 |\mathcal{Q}|^2$, and standard techniques give the bound $O(c_{\varepsilon,\delta} |\mathcal{Q}| \sqrt{\log |\mathcal{X}|})$. Plugging this into (2) shows that $\text{err}_{\text{avg}}(\mathcal{A}_{\text{proj}}, \mathcal{Q})$ is always bounded by (1). It is useful to note that $\|A^{\mathsf{T}} g\|_\infty$ is equal to $h_{K_{\mathcal{Q}}}(g) = \max_{y \in K_{\mathcal{Q}}} |\langle y, g\rangle|$, where $h_{K_{\mathcal{Q}}}$ is the *support function* of $K_{\mathcal{Q}}$. Geometrically, $h_{K_{\mathcal{Q}}}(g)$ is equal to half the width of $K_{\mathcal{Q}}$ in the direction of g, scaled by the Euclidean length of g (see Fig. 1). Thus, the average error of $\mathcal{A}_{\text{proj}}$ scales with the expected width of $n \cdot K_{\mathcal{Q}}$ in a random direction.

Running in Time Sublinear in $|\mathcal{X}|$

An important example when running time linear in $|\mathcal{X}|$ is impractical is marginal queries: the size of the universe is 2^d, which is prohibitive even for a moderate number of attributes. Notice, however, that in order to compute $\tilde{y}$ in Algorithm 2, we only need to compute the true query answers and add independent Gaussian noise to each. This can be done in time $O(n|\mathcal{Q}|)$. The computa-



Geometric Approaches to Answering Queries, Fig. 1 The projection mechanism $\mathcal{A}_{\text{proj}}$ on the *left*: the angle θ is necessarily obtuse or right. The figure on the *right* shows the value of the support function $h_{K_{\mathcal{Q}}}(g)$ is equal to $\frac{1}{2}\|g\|_2$ times the width of $K_{\mathcal{Q}}$ in the direction of g



Geometric Approaches to Answering Queries, Fig. 2 The sensitivity polytope for 2-way marginals on 3 attributes (*left*) and a spectrahedral relaxation of the polytope (*right*). A projection onto 3 of the 12 queries restricted to the positive orthant is shown

tionally expensive operation then is computing the projection $\tilde{y}$. This is a convex optimization problem, and can be solved using the ellipsoid algorithm, provided we have a separation oracle for $K_{\mathcal{Q}}$. (A more practical approach is to use the Frank-Wolfe algorithm which can be implemented efficiently as long as we can solve arbitrary linear programs with feasible region $K_{\mathcal{Q}}$.) For k-way marginals, after a linear transformation that doesn't significantly affect error, $K_{\mathcal{Q}}$ can be assumed to be the convex hull of $\{\pm\chi^{\otimes k} : \chi \in \{-1,1\}^d\}$, where $\chi^{\otimes k}$ is the k-fold tensor power of χ. Unfortunately, even for $k = 2$, separation for this convex body is NP-hard. Nevertheless, a small modification of the analysis of $\mathcal{A}_{\text{proj}}$ shows that the algorithm achieves asymptotically the same error bound if we project onto a convex body L such that $K_{\mathcal{Q}} \subseteq L$ and $\mathbb{E}_g h_L(g) \leq O(1) \cdot \mathbb{E}_g h_{K_{\mathcal{Q}}}(g)$. In other words, we need a convex L that relaxes $K_{\mathcal{Q}}$ but is not too much wider than $K_{\mathcal{Q}}$ in a random direction. If we can find such an L with an efficient separation oracle, we can implement $\mathcal{A}_{\text{proj}}$ in time polynomial in $\mathcal{Q}$ and n while only increasing the error by a constant factor. For 2-way marginals, an appropriate relaxation can be derived from Grothendieck's inequality and is formulated using semidefinite programming. The sensitivity polytope $K_{\mathcal{Q}}$ and the relaxation L are shown for 2-way marginals on $\{0,1\}^3$ in Fig. 2. Finding a relaxation L for k-way marginals with efficient separation and mean width bound $\mathbb{E}_g h_L(g) \leq O(1) \cdot \mathbb{E}_g h_{K_{\mathcal{Q}}}(g)$ is an open problem for $k \geq 3$.

Optimal Error for Small Databases

We can refine the optimal error $\text{opt}^{\varepsilon,\delta}_{\text{avg}}(\mathcal{Q})$ to a curve $\text{opt}^{\varepsilon,\delta}_{\text{avg}}(\mathcal{Q}, n)$, where $\text{opt}^{\varepsilon,\delta}_{\text{avg}}(\mathcal{Q}, n)$ is the infimum of $\text{err}_{\text{avg}}(\mathcal{A}, \mathcal{Q}, n)$ over all (ε, δ)-differentially private algorithms $\mathcal{A}$. There exists an algorithm that, for any database of size at most n and any query set $\mathcal{Q}$, has an average error only a polylogarithmic (in $|\mathcal{Q}|$, $|\mathcal{X}|$, and $1/\delta$) factor larger than $\text{opt}^{\varepsilon,\delta}_{\text{avg}}(\mathcal{Q}, n)$ [16]. The algorithm is similar to $\mathcal{A}_{\text{proj}}$. However, the noise distribution used is the optimal one from Theorem 2. The post-processing step is also slightly more complicated, but the key step is again noise reduction via projection onto a convex body. The running time is polynomial in $n, |\mathcal{Q}|, |\mathcal{X}|$. Giving analogous guarantees for worst-case error remains open.

Recommended Reading

1. Arora S, Hazan E, Kale S (2012) The multiplicative weights update method: a meta-algorithm and applications. Theory Comput 8(1):121–164
2. Bhaskara A, Dadush D, Krishnaswamy R, Talwar K (2012) Unconditional differentially private mechanisms for linear queries. In: Proceedings of the 44th

symposium on theory of computing (STOC'12), New York. ACM, New York, pp 1269–1284. DOI 10.1145/2213977.2214089, http://doi.acm.org/10.1145/2213977.2214089
3. Blum A, Ligett K, Roth A (2008) A learning theory approach to non-interactive database privacy. In: Proceedings of the 40th annual ACM symposium on theory of computing (STOC'08), Victoria. ACM, New York, pp 609–618. http://doi.acm.org/10.1145/1374376.1374464
4. Bourgain J, Tzafriri L (1987) Invertibility of large submatrices with applications to the geometry of banach spaces and harmonic analysis. Isr J Math 57(2):137–224
5. Bun M, Ullman J, Vadhan S (2013) Fingerprinting codes and the price of approximate differential privacy. arXiv preprint arXiv:13113158
6. Dinur I, Nissim K (2003) Revealing information while preserving privacy. In: Proceedings of the 22nd ACM symposium on principles of database systems, San Diego, pp 202–210
7. Dwork C, Nissim K (2004) Privacy-preserving datamining on vertically partitioned databases. In: Advances in cryptology – CRYPTO'04, Santa Barbara, pp 528–544
8. Dwork C, Mcsherry F, Nissim K, Smith A (2006) Calibrating noise to sensitivity in private data analysis. In: TCC, New York. http://www.cs.bgu.ac.il/~kobbi/papers/sensitivity-tcc-final.pdf
9. Dwork C, McSherry F, Talwar K (2007) The price of privacy and the limits of LP decoding. In: Proceedings of the thirty-ninth annual ACM symposium on theory of computing (STOC'07), San Diego. ACM, New York, pp 85–94. DOI 10.1145/1250790.1250804, http://doi.acm.org/10.1145/1250790.1250804
10. Gupta A, Roth A, Ullman J (2012) Iterative constructions and private data release. In: TCC, Taormina, pp 339–356. http://dx.doi.org/10.1007/978-3-642-28914-9_19
11. Hardt M, Rothblum G (2010) A multiplicative weights mechanism for privacy-preserving data analysis. In: Proceedings of the 51st foundations of computer science (FOCS), Las Vegas. IEEE
12. Hardt M, Talwar K (2010) On the geometry of differential privacy. In: Proceedings of the 42nd ACM symposium on theory of computing (STOC'10), Cambridge. ACM, New York, pp 705–714. DOI 10.1145/1806689.1806786, http://doi.acm.org/10.1145/1806689.1806786
13. Li C, Hay M, Rastogi V, Miklau G, McGregor A (2010) Optimizing linear counting queries under differential privacy. In: Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS'10), Indianapolis. ACM, New York, pp 123–134. http://doi.acm.org/10.1145/1807085.1807104
14. Lovász L, Spencer J, Vesztergombi K (1986) Discrepancy of set-systems and matrices. Eur J Comb 7(2):151–160
15. Muthukrishnan S, Nikolov A (2012) Optimal private halfspace counting via discrepancy. In: Proceedings of the 44th symposium on theory of computing (STOC'12), New York. ACM, New York, pp 1285–1292. DOI 10.1145/2213977.2214090, http://doi.acm.org/10.1145/2213977.2214090
16. Nikolov A, Talwar K, Zhang L (2013) The geometry of differential privacy: the sparse and approximate cases. In: Proceedings of the 45th annual ACM symposium on symposium on theory of computing (STOC'13), Palo Alto. ACM, New York, pp 351–360. DOI 10.1145/2488608.2488652, http://doi.acm.org/10.1145/2488608.2488652
17. Plotkin SA, Shmoys DB, Tardos E (1995) Fast approximation algorithms for fractional packing and covering problems. Math Oper Res 20(2):257–301. DOI 10.1287/moor.20.2.257, http://dx.doi.org/10.1287/moor.20.2.257
18. Vershynin R (2001) John's decompositions: selecting a large part. Isr J Math 122(1):253–277

Geometric Dilation of Geometric Networks

Rolf Klein
Institute for Computer Science, University of Bonn, Bonn, Germany

Keywords

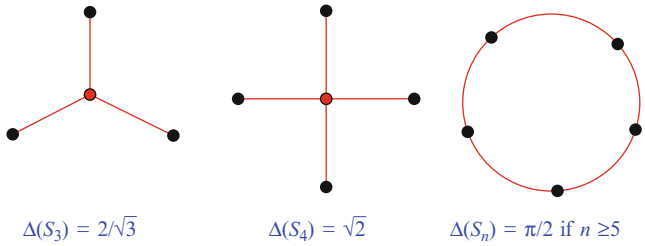
Detour; Spanning ratio; Stretch factor

Years and Authors of Summarized Original Work

2006; Dumitrescu, Ebbers-Baumann, Grüne, Klein, Knauer, Rote

Problem Definition

Urban street systems can be modeled by *plane geometric networks* $G = (V, E)$ whose edges $e \in E$ are piecewise smooth curves that connect the vertices $v \in V \subset \mathbb{R}^2$. Edges do not intersect, except at common endpoints in V. Since streets are lined with houses, the quality of such a network can be measured by the length of the connections



Geometric Dilation of Geometric Networks, Fig. 1 Minimum dilation embeddings of regular point sets

it provides between two arbitrary points p and q on G.

Let $\xi_G(p,q)$ denote a shortest path from p to q in G. Then

$$\delta(p,q) := \frac{|\xi_G(p,q)|}{|pq|} \tag{1}$$

is the detour one encounters when using network G, in order to get from p to q, instead of walking straight. Here, $|\ .\ |$ denotes the Euclidean length. The *geometric dilation of network* G is defined by

$$\delta(G) := \sup_{p \neq q \in G} \delta(p,q). \tag{2}$$

This definition differs from the notion of stretch factor (or spanning ratio) used in the context of spanners; see the monographs by Eppstein [6] or Narasimhan and Smid [11]. In the latter, only the paths between the vertices $p, q \in V$ are considered, whereas the geometric dilation involves all points on the edges as well. As a consequence, the stretch factor of a triangle T equals 1, but its geometric dilation is given by $\delta(T) = \sqrt{2/(1-\cos\alpha)} \geq 2$, where $\alpha \leq 60°$ is the most acute angle of T.

Presented with a finite set S of points in the plane, one would like to find a finite geometric network containing S whose geometric dilation is as small as possible. The value of

$$\Delta(S) := \inf\{\delta(G); G \text{ finite plane geometric network containing } S\}$$

is called the *geometric dilation of point set S*. The problem is in computing, or bounding, $\Delta(S)$ for a given set S.

Key Results

Theorem 1 ([4]) *Let S_n denote the set of corners of a regular n-gon. Then, $\Delta(S_3) = 2/\sqrt{3}, \Delta(S_4) = \sqrt{2}$, and $\Delta(S_n) = \pi/2$ for all $n \geq 5$.*

The networks realizing these minimum values are shown in Fig. 1. The proof of minimality uses the following two lemmata that may be interesting in their own right. Lemma 1 was independently obtained by Aronov et al. [1].

Lemma 1 *Let T be a tree containing S_n. Then $\delta(T) \geq n/\pi$.*

Lemma 2 follows from a result of Gromov's [7]. It can more easily be proven by applying Cauchy's surface area formula; see [4].

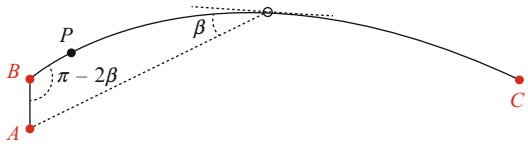
Lemma 2 *Let C denote a simple closed curve in the plane. Then $\delta(C) \geq \pi/2$.*

Clearly, Lemma 2 is tight for the circle. The next lemma implies that the circle is the only closed curve attaining the minimum geometric dilation of $\pi/2$.

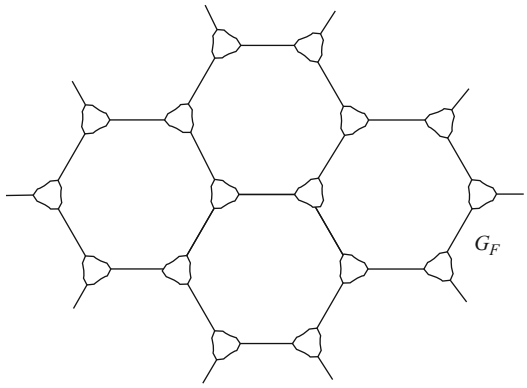
Lemma 3 ([3]) *Let C be a simple closed curve of geometric dilation $< \pi/2 + \epsilon(\delta)$. Then C is contained in an annulus of width δ.*

For points in general position, computing their geometric dilation seems quite complicated. Only for sets $S = \{A, B, C\}$ of size three is the solution completely known.

Theorem 2 ([5]) *The plane geometric network of minimum geometric dilation containing three given points $\{A, B, C\}$ is either a line segment, or a Steiner tree as depicted in Fig. 1, or a simple path consisting of two line segments and one segment of an exponential spiral; see Fig. 2.*



Geometric Dilation of Geometric Networks, Fig. 2 The minimum dilation embedding of points A, B, and C



Geometric Dilation of Geometric Networks, Fig. 3 A network of geometric dilation ≈1,6778

The optimum path shown in Fig. 2 contains a degree two Steiner vertex, P, situated at distance $|AB|$ from B. The path runs straight between A, B and B, P. From P to C, it follows an exponential spiral centered at A.

The next results provide upper and lower bounds to $\Delta(S)$.

Theorem 3 ([4]) *For each finite point set S, the estimate $\Delta(S) < 1.678$ holds.*

To prove this general upper bound, one can replace each vertex of the hexagonal tiling of $\mathbb{R}^2$ with a certain closed Zindler curve (by definition, all point pairs bisecting the perimeter of a Zindler curve have identical distance). This results in a network G_F of geometric dilation ≈1.6778; see Fig. 3. Given a finite point set S, one applies a slight deformation to a scaled version of G_F, such that all points of S lie on a finite part, G, of the deformed net. By Dirichlet's result on simultaneous approximation of real numbers by rationals, a deformation small as compared to the cell size is sufficient, so that the dilation is not affected. See [8] for the history and properties of Zindler curves.

Theorem 4 ([3]) *There exists a finite point set S such that $\Delta(S) > (1 + 10^{-11})\pi/2$.*

Theorem 4 holds for the set S of 19×19 vertices of the integer grid. Roughly, if S were contained in a geometric network G of dilation close to $\pi/2$, the boundaries of the faces of G must be contained in small annuli, by Lemma 3. To the inner and outer circles of these annuli, one can now apply a result by Kuperberg et al. [9] stating that an enlargement, by a certain factor, of a packing of disks of radius ≤ 1 cannot cover a square of size 4.

Applications

The geometric dilation has applications in the theory of knots; see, e.g., Kusner and Sullivan [10] and Denne and Sullivan [2]. With respect to urban planning, the above results highlight principal dilation bounds for connecting given sites with plane geometric networks.

Open Problems

For practical applications, one would welcome upper bounds to the weight (= total edge length) of a geometric network, in addition to upper bounds on its geometric dilation. Some theoretical questions require further investigation, too. Is $\Delta(S)$ always attained by a finite network? How to compute, or approximate, $\Delta(S)$ for a given finite set S? What is the precise value of $\sup\{\Delta(S); S \text{finite}\}$?

Cross-References

▸ Dilation of Geometric Networks

Recommended Reading

1. Aronov B, de Berg M, Cheong O, Gudmundsson J, Haverkort H, Vigneron A (2008) Sparse geometric graphs with small dilation. Comput Geom Theory Appl 40(3):207–219
2. Denne E, Sullivan JM (2004) The distortion of a knotted curve. http://www.arxiv.org/abs/math.GT/0409438
3. Dumitrescu A, Ebbers-Baumann A, Grüne A, Klein R, Rote G (2006) On the geometric dilation of closed curves, graphs, and point sets. Comput Geom Theory Appl 36(1):16–38
4. Ebbers-Baumann A, Grüne A, Klein R (2006) On the geometric dilation of finite point sets. Algorithmica 44(2):137–149
5. Ebbers-Baumann A, Klein R, Knauer C, Rote G (2006) The geometric dilation of three points. Manuscript
6. Eppstein D (1999) Spanning trees and spanners. In: Sack J-R, Urrutia J (eds) Handbook of computational geometry, pp 425–461. Elsevier, Amsterdam
7. Gromov M (1981) Structures Métriques des Variétés Riemanniennes. Textes Math. CEDIX, vol 1. F. Nathan, Paris
8. Grüne A (2006) Geometric dilation and halving distance. Ph.D. thesis, Institut für Informatik I, Universität Bonn
9. Kuperberg K, Kuperberg W, Matousek J, Valtr P (1999) Almost tiling the plane with ellipses. Discrete Comput Geom 22(3):367–375
10. Kusner RB, Sullivan JM (1998) On distortion and thickness of knots. In: Whittington SG et al (eds) Topology and geometry in polymer science. IMA volumes in mathematics and its applications, vol 103. Springer, New York, pp 67–78
11. Narasimhan G, Smid M (2007) Geometric spanner networks. Cambridge University Press, Cambridge/New York

Geometric Object Enumeration

Shin-ichi Nakano
Department of Computer Science, Gunma University, Kiryu, Japan

Keywords

Enumeration; Floor plan; Generation; Listing; Plane triangulation; Reverse search

Years and Authors of Summarized Original Work

2001; Li, Nakano
2001; Nakano
2004; Nakano

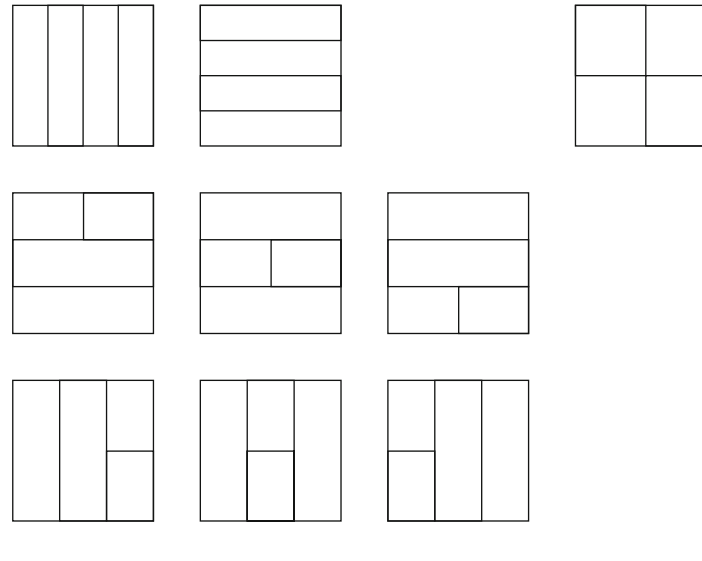
Problem Definition

Enumerating objects with the given property is one of basic problems in mathematics. We review some geometric objects enumeration problems and algorithms to solve them.

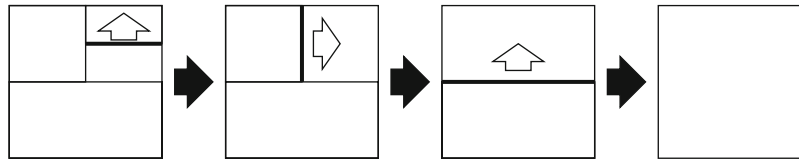
A graph is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. A *plane* graph is a planar graph with a fixed planar embedding. A plane graph divides the plane into connected regions called *faces*. The unbounded face is called the outer face, and other faces are called inner faces.

A plane graph is a *floor plan* if each face (including the outer face) is a rectangle. A *based* floor plan is a floor plan with one designated line on the contour of the outer face. The designated line is called the *base* line and we always draw the base line as the lowermost horizontal line of the drawing. The 25 based floor plans having 4 inner faces are shown in Fig. 1. Given an integer f the problem of *floor plan enumeration* asks for generating all floor plans with exactly f inner faces.

A plane graph is a *plane triangulation* if each inner face has exactly three edges on its contour. A *based* plane triangulation is a plane triangulation with one designated edge on the contour of the outer face. The designated edge is called the *base* edge. Triangulations are important model for 3D modeling. A graph is biconnected if removing any vertex always results in a connected graph. A graph is triconnected if removing any two vertices always results in a connected graph. Given two integers n and r, the problem of *biconnected plane triangulation enumeration* asks for generating all biconnected



Geometric Object Enumeration, Fig. 1 The 25 based floor plans having 4 inner faces



Geometric Object Enumeration, Fig. 2 The sequence

Geometric Object Enumeration, Fig. 3 The tree F_4

plane triangulations with exactly n vertices including exactly r vertices on the outer face. Given two integers n and r, the problem of *triconnected plane triangulation enumeration* asks for generating all triconnected plane triangulations with exactly n vertices including exactly r vertices on the outer face.

Key Results

Enumeration of All Floor Plans

Using *reverse search method* [1], one can enumerate all based floor plans with f inner faces in $O(1)$ time for each [3]. We sketch the method in [3].

Let S_f be the set of all based floor plans with $f > 1$ inner faces. Let R be a based floor plan in S_f and F a face of R having the upper right corner of R. We have two cases. If R has a vertical line segment with upper end at the lower left corner of F, then by continually shrinking R to the uppermost horizontal line of R with preserving the width of F and enlarging the faces below R, we can have a based floor plan with one less inner face. If R has no vertical line segment with the upper end at the lower left corner of F, then R has a horizontal line segment with the right end at the lower left corner of F, and then by continually shrinking R to the rightmost vertical line of R with preserving the height of F and enlarging the faces locating the left of R, we can have a base floor plan with one less inner face. Repeating this results in the sequence of based floor plans which always ends with the based floor plan with one inner face. See an example in Fig. 2. If we merge the sequence of all R in S_f, then we have the tree T_f in which every R in S_f appears as a leaf in T_f. See Fig. 3.

The reverse search method efficiently traverses the tree (without storing the tree in the memory) and output each based floor plan in S_f at each corresponding leaf. Thus, we can efficiently enumerate all based floor plans in S_f. The algorithm enumerates all based floor plans in S_f in $O(1)$ time for each.

Enumeration of Triangulations

Similarly, using *reverse search method* [1], given two integers n and r, one can enumerate all based biconnected triangulations having exactly n vertices including exactly r vertices on the outer face in $O(1)$ time for each [2], all based triconnected triangulations having exactly n vertices including exactly r vertices on the outer face in $O(1)$ time for each [3], and all triconnected (non-based) plane triangulation having exactly n vertices including exactly r vertices on the outer face in $O(r^2 n)$ time for each [3]. Also one can enumerate all based triangulation having exactly n vertices with exactly three vertices on the outer face in $O(1)$ time for each [3].

Cross-References

- ▸ Enumeration of Paths, Cycles, and Spanning Trees
- ▸ Reverse Search; Enumeration Algorithms
- ▸ Tree Enumeration

Recommended Reading

1. Avis D, Fukuda K (1996) Reverse search for enumeration. Discret Appl Math 65(1–3):21–46
2. Li Z, Nakano S (2001) Efficient generation of plane triangulations without repetitions. In: Proceedings of the 28th international colloquium on automata, languages and programming, (ICALP 2001), Crete. LNCS, vol 2076, pp 433–443
3. Nakano S (2004) Efficient generation of triconnected plane triangulations. Comput Geom 27:109–122
4. Nakano S (2001) Enumerating floorplans with n rooms. In: Proceedings of the 12th international symposium on algorithms and computation (ISAAC 2001), Christchurch. LNCS, vol 2223, pp 107–115

Geometric Shortest Paths in the Plane

John Hershberger
Mentor Graphics Corporation, Wilsonville, OR, USA

Keywords

Computational geometry; Continuous Dijkstra method; Shortest path; Shortest path map; Visibility graph

Years and Authors of Summarized Original Work

1996; Mitchell
1999; Hershberger, Suri

Problem Definition

Finding the shortest path between a source and a destination is a natural optimization problem with many applications. Perhaps the oldest variant of the problem is the geometric shortest path problem, in which the domain is physical space: the problem is relevant to human travelers, migrating animals, and even physical phenomena like wave propagation. The key feature that distinguishes the geometric shortest path problem from the corresponding problem in graphs or other discrete spaces is the unbounded number of paths in a multidimensional space. To solve the problem efficiently, one must use the "shortness" criterion to limit the search.

In computational geometry, physical space is modeled abstractly as the union of some number of constant-complexity primitive elements. The traditional formulation of the shortest path problem considers paths in a domain bounded by linear elements – line segments in two dimensions, triangles in three dimensions, and $(d-1)$-dimensional simplices in $d > 3$. Canny and Reif showed that the three-dimensional shortest path problem is NP-complete [2], so this article will focus on the two-dimensional problem.

We consider paths in a free space P bounded by h polygons – one outer boundary and $(h-1)$ obstacles – with a total of n vertices. The free space is closed, so paths may touch the boundary. The source and destination of the shortest path are points s and t inside or on the boundary of P. The goal of the shortest path problem is to find the shortest path from s to t inside P, denoted by $\pi(s,t)$, as efficiently as possible, where running time and memory use are expressed as functions of n and h. The length of $\pi(s,t)$ is denoted by $dist(s,t)$; in some applications, it may be desirable to compute $dist(s,t)$ without finding $\pi(s,t)$ explicitly.

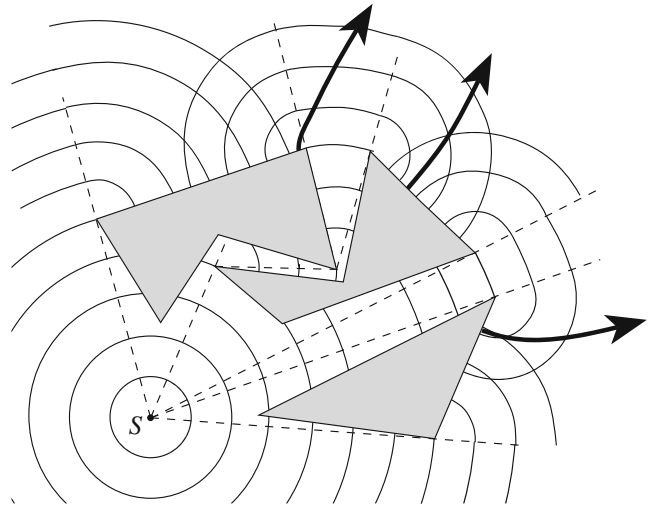
Key Results

Visibility Graph Algorithms

Early approaches to the two-dimensional shortest path problem exploited the *visibility graph* to reduce the continuous shortest path problem to a discrete graph problem [1, 12, 18]. The visibility graph is a graph whose nodes are s, t, and the vertices of P and whose edges (u,v) connect vertex pairs such that the line segment $\overline{uv}$ is contained in P. It is convenient and customary to identify the edges of the abstract visibility graph with the line segments they represent. The visibility graph is important because the edges of the shortest path $\pi(s,t)$ are a subset of the visibility graph edges. This is easy to understand intuitively, because of subpath optimality – for any two points $a, b \in \pi(s,t)$, the subpath of $\pi(s,t)$ between a and b is also the shortest path between a and b. In particular, if $\overline{ab}$ is contained in P, then $\pi(s,t)$ coincides with $\overline{ab}$ between a and b, and the distance $dist(a,b)$ is equal to $|ab|$, the length of the segment $\overline{ab}$. If $\pi(s,t)$ has a bend anywhere except at a vertex of P, an infinitesimal subpath near the bend can be shortened by a straight shortcut, implying that $\pi(s,t)$ is *not* the shortest path. Hence every segment of $\pi(s,t)$ is an edge of the visibility graph.

This observation leads directly to an algorithm for computing shortest paths: compute the visibility graph of P and then run Dijkstra's algorithm to find the shortest path from s to t in the visibility graph. The visibility graph can be constructed in $O(n \log n + m)$ time, where m is the number of edges, using an algorithm of Ghosh and Mount [7]. Dijkstra's algorithm takes $O(n \log n + m)$ time on a graph with n nodes and m edges [4], so this is the running time of the straightforward visibility graph solution to the shortest path problem. This algorithm can be quadratic in the worst case, since m, the number of visibility graph edges, can be as large as $\Theta(n^2)$.

The running time can be improved somewhat by noting that only a subset of the visibility graph edges can belong to a shortest path. In particular, any shortest path must turn toward the boundary of P at any path vertex. This limits the edges to common tangents of the polygons of P. We omit



Geometric Shortest Paths in the Plane, Fig. 1
The spreading wavefront

the details, but note that if s and t are known, the common tangent restriction limits the number of visibility graph edges that may belong to $\pi(s,t)$ to $O(n+h^2)$ [11]. These useful edges can be computed in $O(n\log n+h^2)$ time [17], and so the shortest path can be computed in the same time bound by applying Dijkstra's algorithm to the subgraph [11].

Continuous Dijkstra Algorithms

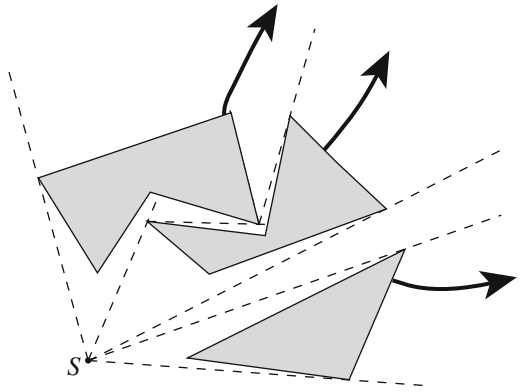
Visibility graph approaches to finding the shortest path run in quadratic time in the worst case, since h may be $\Theta(n)$. This led Mitchell to propose an alternative approach called the *continuous Dijkstra method* [15]. Imagine a wavefront that spreads at unit speed inside P, starting from s. The wavefront at time τ is the set of points in P whose geodesic (shortest path) distance from s is exactly τ. Said another way, the shortest path distance from s to a point $p \in P$ is equal to the time at which the wavefront reaches p.

The wavefront at time τ is a union of paths and cycles bounding the region whose geodesic distance from s is at most τ. Each path or cycle is a sequence of circular arc *wavelets*, each centered on a vertex v that is its *root*. The radius of the wavelet is $\tau - dist(s,v)$, with $dist(s,v) < \tau$. As τ increases, the combinatorial structure of the wavefront changes at discrete event times when the wavefront hits the free space boundary, collides with itself, or eliminates wavelets squeezed between neighboring wavelets. See Fig. 1. The continuous Dijkstra method simulates the spread of this wavefront, computing the shortest path distance to every point of the free space in the process.

Mitchell used the continuous Dijkstra method to compute shortest paths under the L_1 metric in $O(n\log n)$ time [15]. He later extended the approach to compute L_2 (Euclidean) shortest paths in $O(n^{5/3+\epsilon})$ time, for ϵ arbitrarily small [16]. Hershberger and Suri gave an alternative implementation of the continuous Dijkstra scheme, using different data structures, that computes Euclidean shortest paths in $O(n\log n)$ time [9]. The next two subsections discuss these algorithms in more detail.

Continuous Dijkstra with Sector Propagation Queries

If p is a point in P, $\pi(s,p)$ is a shortest path from s to p, and the *predecessor* of p is the vertex of $\pi(s,p)$ adjacent to (immediately preceding) p in the path. If a point is reached by multiple shortest paths, it has multiple predecessors. The *shortest path map* is a linear-complexity partition of P into regions such that every point inside a region has the same predecessor. See Fig. 2. The *root* of each region is the predecessor of all points in the region. The edges of the shortest path map are polygon edges and *bisectors* (curves with two distinct predecessors, namely, the roots of the regions separated by the bisector).



Geometric Shortest Paths in the Plane, Fig. 2 The shortest path map for the wavefront in Fig. 1

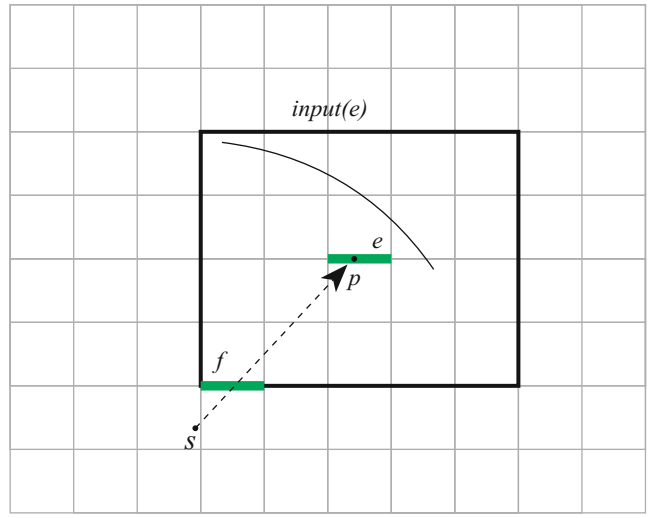
Mitchell's shortest path algorithm simulates the spread of the wavefront inside the shortest path map. This may seem a bit peculiar, since the shortest path map is not known until the shortest paths have been computed. The trick is that the algorithm builds the shortest path map as it runs, and it propagates a *pseudo-wavefront* inside its current model of the shortest path map at each step. The true wavefront is a subset of the pseudo-wavefront. This pseudo-wavefront is locally correct – each wavelet's motion is determined by its neighbors in the pseudo-wavefront and the shortest path map known so far – but it may overrun itself. When an overrun is detected, the algorithm revises its model of the shortest path map in the neighborhood of the overrun.

To be more specific, each wavelet w is a circular arc centered at a root vertex $r(w)$. The endpoints of w move along left and right *tracks* $\alpha(w)$ and $\beta(w)$. Each track is either a straight line segment (a polygon edge or an extension of a visibility edge) or a bisector determined by w and the left/right neighbor wavefront $L(w)$ or $R(w)$. For example, if $r = r(w)$ and $r' = r(L(w))$, the left bisector is the set of points x such that $dist(s, r) + |rx| = dist(s, r') + |r'x|$; consequently, the bisector is a hyperbolic arc. For every wavelet w, the algorithm computes a next event, which is the next value of τ where w reaches an endpoint of one of its tracks, the left and right tracks collide, or w hits a polygon vertex between its left and right tracks. (Collisions with polygon edges or other wavefront arcs are *not* detected.) The events for all wavelets are placed in a global priority queue and processed in order of increasing τ values.

When the algorithm processes an event, it updates the wavelets involved and their events in the priority queue. Processing wavelet collisions with a polygon vertex v is the most complicated case: To detect possible previous collisions with polygon edges, the algorithm performs a ray shooting query from $r(w)$ toward v [8]. If the ray hits an edge, the algorithm traces the edge through the current shortest path map regions and updates the corresponding wavelets. If v is reached for the first time by w, then the algorithm updates the wavefront with a new wavelet rooted at v. If vertex v was previously reached by another wavelet, then there are previously undiscovered bisectors between w and the other wavelet. The algorithm traces these bisectors through its local shortest path map model and carves off portions that are reached by a shorter path following another route. Processing other events (track vertices and track collisions) is similar.

Mitchell shows that even though vertices may be reached more than once by different wavelets, and portions of the shortest path map are carved off and discarded when they are discovered to be invalid, no vertex is reached more than $O(1)$ times, and the total shortest path map complexity, even including discarded portions, is $O(n)$. The most costly part of the algorithm is finding the first polygon vertex hit by each wavelet. All the rest of the algorithm – ray shooting, priority queue, bisector tracing, and maintaining the shortest path map structure – can be done in $O(n \log n)$ total time.

The complexity of Mitchell's algorithm is dominated by *wavelet dragging queries*, which find the first obstacle vertex hit by a wavelet w between the left and right tracks $\alpha(w)$ and $\beta(w)$. Mitchell phrases this as a ten-dimensional optimization problem dependent on the position and distance from s for the root of w and its neighbors in the wavefront, plus the start time τ. Although Euclidean distances are square roots of quadratics (by Pythagoras), Mitchell is able,



Geometric Shortest Paths in the Plane, Fig. 3 The well-covering region for edge e is bounded by $input(e)$

by squaring, substitution, and simplification, to convert the distance minimization problem into a linear optimization range query over a constant-size polyhedron in $\Re^5$. (The objects in the range query are n 5-dimensional points, images of the polygon vertices.) There are $O(n)$ such queries to be performed. Using known bounds and balancing preprocessing against query time, the $O(n)$ queries can be answered in $O(n^{5/3+\epsilon})$ time and space [3, 13, 14]. All other parts of the algorithm take near-linear time, so the total time for Mitchell's algorithm to find the Euclidean shortest path map is $O(n^{5/3+\epsilon})$ [16].

Continuous Dijkstra in a Conforming Subdivision

The challenge of implementing the continuous Dijkstra paradigm is that detecting and processing wavefront events in strict temporal order is difficult to do efficiently, but processing events out of order may lead to incorrect results or to processing too many invalid events. Mitchell addresses the challenge by detecting only one subclass of events in temporal order (wavelet contacts with polygon vertices) and repairing errors in the shortest path map structure as they are discovered. Hershberger and Suri achieve a better time bound (optimal $O(n \log n)$) by processing events in an even more relaxed order [9]. The key to their approach is a subdivision of the free space in which spatial locality is used to bound the temporal inaccuracy of wavefront event processing.

As a simple example, consider a wavefront propagating across an obstacle-free plane that has been subdivided into a grid of unit squares. Each edge e of the grid lies at the center of a 4×5 rectangle of squares. The distance from e to each of the 18 edges on the rectangle boundary is at least 2. If the wavefront source is outside the rectangle, then the first wavelet that reaches any point $p \in e$ must pass through the rectangle boundary at least two time units before it reaches p. By the triangle inequality, an edge of length δ is completely covered by the wavefront within time δ of the time the wavefront first hits it. It follows that if the shortest path to $p \in e$ passes through an edge f on the rectangle boundary, edge f is completely covered by the wavefront at least one time unit before the wavefront reaches p. See Fig. 3.

The algorithm propagates the wavefront from edge to edge in the grid. For each edge e, let $input(e)$ be the edges on the boundary of the 4×5 rectangle around e. The algorithm computes a *cover time* for e, denoted $cover(e)$, that is an upper bound on the time when the wavefront completely covers e. If $fvc(e)$ is the time at which the wavefront first contacts a vertex of e, and $|e|$ is the length of e, then $cover(e)$ is defined to be $fvc(e) + |e|$. For each edge e, $cover(e)$ is

determined by a wavefront passing through $f \in input(e)$, and $cover(f) < cover(e)$.

The propagation algorithm processes edges in order of cover time, computing the wavefront at e by combining the wavefronts from edges $f \in input(e)$ with $cover(f) < cover(e)$. The combination algorithm is linear in the number of features of the shortest path map that lie inside the rectangle for e. The algorithm computes a one-dimensional representation of the intersection of the shortest path map with each edge of the grid; each bisector that has an event (an arc endpoint) within the input region of an edge e is flagged in the wavefront representation for e. To turn the one-dimensional wavefront representation at edges into a two-dimensional representation of the shortest path map, the algorithm combines the wavefronts of the edges on each cell's boundary to compute the shortest path map inside the cell. (The algorithm computes additively weighted Voronoi diagrams [6] for the wavelet roots whose bisectors have events (endpoints) in the cell, plus compact representations of the groups of bisectors that have no endpoints in the cell.)

The key feature of the grid subdivision is *well-covering property*: each edge e is surrounded by a region that is the union of $O(1)$ cells, and the distance from e to the region boundary is relatively large. In particular, if f is an edge on the boundary, $dist(e, f) \geq 2 \cdot \max(|e|, |f|)$. This property allows the algorithm to perform *spatial* (not temporal) wavefront propagation at discrete cover times. Hershberger and Suri show how to extend the well-covering property to a special *conforming subdivision* of free space made up of $O(n)$ constant-complexity cells. The wavefront propagation algorithm carries over from the grid to the conforming subdivision of free space with only a few changes to handle the obstacle vertices. As on the grid, the number of propagation steps and data structure changes is $O(n)$. Including the overhead of a priority queue and data structure updates (full persistence is needed [5]) increases the time and space by a factor of $O(\log n)$, so the overall algorithm runs in $O(n \log n)$ time and space.

Extensions

Hershberger and Suri's algorithm supports multiple wavefront sources, including line-segment sources. Hence the algorithm can be used to compute geodesic Voronoi diagrams, including Voronoi diagrams whose sites are points, segments, polygons, or combinations of all these.

Since the publication of Hershberger and Suri's optimal-time algorithm for shortest paths among polygonal obstacles, their result has been extended to other two-dimensional domains. Schreiber showed how to find shortest paths on the surface of a convex polyhedron in $O(n \log n)$ time [20]. His algorithm decomposes the surface into cells and then propagates wavefronts between cell edges similarly to Hershberger and Suri's algorithm. Schreiber extended his algorithm for polyhedra to work for polygonal terrains as well, assuming that the maximum gradient of the terrain is bounded by a constant [19]. More recently, Hershberger, Suri, and Yıldız [10] extended the algorithm for polygonal obstacles [9] to find shortest paths in a free space bounded by curved obstacle edges. The conforming subdivision for the free space is very similar to that for polygonal obstacles; the chief difficulty is computing the positions of bisector events (intersections). Bisectors for polygonal obstacles are hyperbolic arcs, but they are much more complicated curves for curved obstacles. The algorithm of [10] approximates the bisector events using primitive tangent-finding operations on individual obstacle curves, with the result that the algorithm's running time is $O(n \log(n/\epsilon))$, where ϵ is the relative error of the computed path length.

Cross-References

- Range Searching
- Single-Source Shortest Paths
- Voronoi Diagrams and Delaunay Triangulations

Recommended Reading

1. Asano T, Asano T, Guibas LJ, Hershberger J, Imai H (1986) Visibility of disjoint polygons. Algorithmica 1:49–63
2. Canny J, Reif JH (1987) New lower bound techniques for robot motion planning problems. In: Proceedings of the 28th annual IEEE symposium on foundations of computer Science, Washington, DC, pp 49–60
3. Chazelle B, Sharir M, Welzl E (1992) Quasi-optimal upper bounds for simplex range searching and new zone theorems. Algorithmica 8:407–429
4. Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms, 2nd edn. MIT Press, Cambridge
5. Driscoll JR, Sarnak N, Sleator DD, Tarjan RE (1989) Making data structures persistent. J Comput Syst Sci 38:86–124
6. Fortune SJ (1987) A sweepline algorithm for Voronoi diagrams. Algorithmica 2:153–174
7. Ghosh SK, Mount DM (1991) An output-sensitive algorithm for computing visibility graphs. SIAM J Comput 20:888–910
8. Hershberger J, Suri S (1995) A pedestrian approach to ray shooting: Shoot a ray, take a walk. J Algorithms 18:403–431
9. Hershberger J, Suri S (1999) An optimal algorithm for Euclidean shortest paths in the plane. SIAM J Comput 28(6):2215–2256
10. Hershberger J, Suri S, Yıldız H (2013) A near-optimal algorithm for shortest paths among curved obstacles in the plane. In: Proceedings of the 29th annual symposium on computational geometry, SoCG '13. ACM, New York, pp 359–368
11. Kapoor S, Maheshwari SN, Mitchell JSB (1997) An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. Discret Comput Geom 18:377–383
12. Lozano-Perez T, Wesley MA (1979) An algorithm for planning collision-free paths among polyhedral obstacles. Commun ACM 22:560–570
13. Matoušek J (1993) Range searching with efficient hierarchical cuttings. Discret Comput Geom 10(2):157–182
14. Megiddo N (1983) Applying parallel computation algorithms in the design of serial algorithms. J ACM 30(4):852–865
15. Mitchell JSB (1992) L_1 shortest paths among polygonal obstacles in the plane. Algorithmica 8:55–88
16. Mitchell JSB (1996) Shortest paths among obstacles in the plane. Int J Comput Geom Appl 6:309–332
17. Pocchiola M, Vegter G (1996) Topologically sweeping visibility complexes via pseudotriangulations. Discret Comput Geom 16(4):419–453
18. Rohnert H (1988) Time and space efficient algorithms for shortest paths between convex polygons. Inf Process Lett 27:175–179
19. Schreiber Y (2010) An optimal-time algorithm for shortest paths on realistic polyhedra. Discret Comput Geom 43(1):21–53
20. Schreiber Y, Sharir M (2008) An optimal-time algorithm for shortest paths on a convex polytope in three dimensions. Discret Comput Geom 39(1–3):500–579

Geometric Spanners

Joachim Gudmundsson[1,2], Giri Narasimhan[3,4], and Michiel Smid[5]

[1]DMiST, National ICT Australia Ltd, Alexandria, Australia

[2]School of Information Technologies, University of Sydney, Sydney, NSW, Australia

[3]Department of Computer Science, Florida International University, Miami, FL, USA

[4]School of Computing and Information Sciences, Florida International University, Miami, FL, USA

[5]School of Computer Science, Carleton University, Ottawa, ON, Canada

Keywords

Computational geometry; Dilation; Geometric networks; Spanners

Years and Authors of Summarized Original Work

2002; Gudmundsson, Levcopoulos, Narasimhan

Problem Definition

Consider a set S of n points in d-dimensional Euclidean space. A network on S can be modeled as an undirected graph G with vertex set S of size n and an edge set E where every edge (u, v) has a weight. A geometric (Euclidean) network is a network where the weight of the edge (u, v) is the Euclidean distance $|uv|$ between its end points. Given a real number $t > 1$, we say that G is a *t-spanner* for S, if for each pair of points $u, v \in S$, there exists a path in G of weight at most t times the Euclidean distance between u and v. The minimum t such that G is a t-spanner for S is called the stretch factor, or dilation, of G. For a detailed description of many constructions

of t-spanners, see the book by Narasimhan and Smid [30]. The problem considered is the construction of t-spanners given a set S of n points in $\mathcal{R}^d$ and a positive real value $t > 1$, where d is a constant. The aim is to compute a good t-spanner for S with respect to the following quality measures:

size: the number of edges in the graph
degree: the maximum number of edges incident on a vertex
weight: the sum of the edge weights
spanner diameter: the smallest integer k such that for any pair of vertices u and v in S, there is a path in the graph of length at most $t \cdot |uv|$ between u and v containing at most k edges
fault tolerance: the resilience of the graph to edge, vertex, or region failures

Thus, good t-spanners require large fault tolerance and small size, degree, weight, and spanner diameter. Additionally, the time required to compute such spanners must be as small as possible.

Key Results

This section contains descriptions of several known approaches for constructing a t-spanner of a set of points in Euclidean space. We also present descriptions of the construction of fault-tolerant spanners, spanners among polygonal obstacles, and, finally, a short note on dynamic and kinetic spanners.

Spanners of Points in Euclidean Space

The most well-known classes of t-spanner networks for points in Euclidean space include Θ-graphs, WSPD graphs, and greedy spanners. In the following sections, the main idea of each of these classes is given, together with the known bounds on the quality measures.

The Θ-Graph

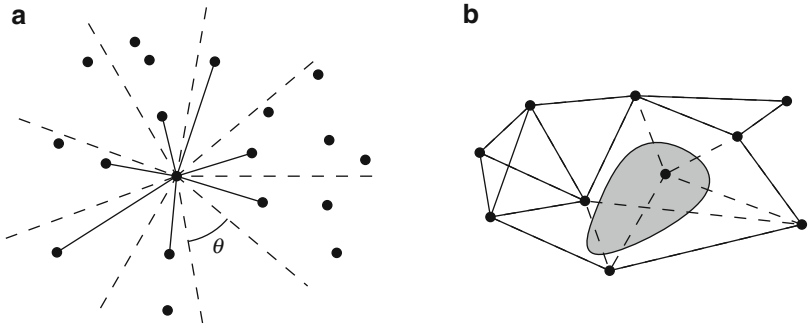
The Θ-graph was discovered independently by Clarkson and Keil in the late 1980s. The general idea is to process each point $p \in S$ independently as follows: partition $\mathcal{R}^d$ into k simplicial cones of angular diameter at most θ and apex at p, where $k = O(1/\theta^{d-1})$. For each nonempty cone C, an edge is added between p and the point in C whose orthogonal projection onto some fixed ray in C emanating from p is closest to p; see Fig. 1a. The resulting graph is called the Θ-graph on S. The following result is due to Arya et al. [9].

Theorem 1 *The Θ-graph is a t-spanner of S for $t = \frac{1}{\cos\theta - \sin\theta}$ with $O\left(\frac{n}{\theta^{d-1}}\right)$ edges and can be computed in $O\left(\frac{n}{\theta^{d-1}} \log^{d-1} n\right)$ time using $O\left(\frac{n}{\theta^{d-1}} + n \log^{d-2} n\right)$ space.*

The following variants of the Θ-graph also give bounds on the degree, spanner diameter, and weight.

Skip-List Spanners

The idea is to generalize skip lists and apply them to the construction of spanners. Construct a



Geometric Spanners, Fig. 1 (**a**) Illustrating the Θ-graph and (**b**) a graph with a region fault

sequence of h subsets, $S_1, \ldots, S_h$, where $S_1 = S$ and S_i is constructed from S_{i-1} as follows (reminiscent of the levels in a skip list). For each point in S_{i-1}, flip a fair coin. The set S_i is the set of all points of S_{i-1} whose coin flip produced heads. The construction stops if $S_i = \emptyset$. For each subset, a Θ-graph is constructed. The union of the graphs is the skip-list spanner of S with dilation t, having $O\left(\frac{n}{\theta^{d-1}}\right)$ edges and $O(\log n)$ spanner diameter with high probability [9].

Gap Greedy

A set of directed edges is said to satisfy the *gap* property if the sources of any two distinct edges in the set are separated by a distance that is at least proportional to the length of the shorter of the two edges. Arya and Smid [6] proposed an algorithm that uses the gap property to decide whether or not an edge should be added to the t-spanner graph. Using the gap property, the constructed spanner can be shown to have degree $O(1/\theta^{d-1})$ and weight $O(\log n \cdot \mathrm{wt}(\mathrm{MST}(S)))$, where $\mathrm{wt}(\mathrm{MST}(S))$ is the weight of the minimum spanning tree of S.

The WSPD Graph

The well-separated pair decomposition (WSPD) was developed by Callahan and Kosaraju [12]. The construction of a t-spanner using the well-separated pair decomposition is done by first constructing a WSPD of S with respect to a separation constant $s = \frac{4(t+1)}{(t-1)}$. Initially set the spanner graph $G = (S, \emptyset)$ and add edges iteratively as follows. For each well-separated pair $\{A, B\}$ in the decomposition, an edge (a, b) is added to the graph, where a and b are arbitrary points in A and B, respectively. The resulting graph is called the WSPD graph on S.

Theorem 2 *The WSPD graph is a t-spanner for S with $O(s^d \cdot n)$ edges and can be constructed in time $O(s^d n + n \log n)$, where $s = 4(t+1)/(t-1)$.*

There are modifications that can be made to obtain bounded spanner diameter or bounded degree.

Bounded spanner diameter: Arya, Mount, and Smid [7] showed how to modify the construction algorithm such that the spanner diameter of the graph is bounded by $2 \log n$. Instead of selecting an arbitrary point in each well-separated set, their algorithm carefully chooses a representative point for each set.

Bounded degree: A single point v can be part of many well-separated pairs, and each of these pairs may generate an edge with an end point at v. Arya et al. [8] suggested an algorithm that retains only the shortest edge for each cone direction, thus combining the Θ-graph approach with the WSPD graph. By adding a postprocessing step that handles all high-degree vertices, a t-spanner of degree $O\left(\frac{1}{(t-1)^{2d-1}}\right)$ is obtained.

The Greedy Spanner

The greedy algorithm was first presented in 1989 by Bern, and since then, the greedy algorithm has been subject to considerable research. The graph constructed using the greedy algorithm is called a Greedy spanner, and the general idea is that the algorithm iteratively builds a graph G. The edges in the complete graph are processed in order of increasing edge length. Testing an edge (u, v) entails a shortest path query in the partial spanner graph G. If the shortest path in G between u and v is at most $t \cdot |uv|$, then the edge (u, v) is discarded; otherwise, it is added to the partial spanner graph G.

Das, Narasimhan, and Salowe [22] proved that the Greedy spanner fulfills the so-called *leapfrog property*. A set of undirected edges E is said to satisfy the t-leapfrog property, if for every $k \geq 2$, and for every possible sequence $\{(p_1, q_1), \ldots, (p_k, q_k)\}$ of pairwise distinct edges of E,

$$t \cdot |p_1 q_1| < \sum_{i=2}^{k} |p_i q_i| + t \cdot \left(\sum_{i=1}^{k-1} |q_i p_{i+1}| + |p_k q_1| \right).$$

Using the leapfrog property, it has been shown that the total edge weight of the graph is within a constant factor of the weight of a minimum spanning tree of S.

Using Dijkstra's shortest-path algorithm, the greedy spanner can be constructed in $O(n^3 \log n)$ time. Bose et al. [10] improved the time to $O(n^2 \log n)$, while using $O(n^2)$ space. Alewijnse et al. [4] improved the space bound to $O(n)$, while slightly increasing the time bound to $O(n^2 \log^2 n)$.

Das and Narasimhan [21] observed that an approximation of the greedy spanner can be constructed while maintaining the leapfrog property. This observation allowed for faster construction algorithms.

Theorem 3 ([27]) *The greedy spanner is a t-spanner of S with $O\left(\frac{n}{(t-1)^d} \log(\frac{1}{t-1})\right)$ edges, maximum degree $O\left(\frac{1}{(t-1)^d} \log(\frac{1}{t-1})\right)$, and weight $O\left(\frac{1}{(t-1)^{2d}} \cdot \mathrm{wt}(\mathrm{MST}(S))\right)$ and can be computed in time $O\left(\frac{n}{(t-1)^{2d}} \log n\right)$.*

The Transformation Technique

Chandra et al. [16, 17] introduced a transformation technique for general metrics that transforms an algorithm for constructing spanners with small stretch factor and size into an algorithm for constructing spanners with the same asymptotic stretch factor and size, but with the additional feature of small weight. Elkin and Solomon [24] refined their approach to develop a transformation technique that achieved the following: It takes an algorithm for constructing spanners with small stretch factor, small size, small degree, and small spanner diameter and transforms it into an algorithm for constructing spanners with a small increase in stretch factor, size, degree, and spanner diameter, but that also has small weight and running time.

Using the transformation technique allowed Elkin and Solomon to prove the following theorem.

Theorem 4 ([24]) *For any set of n points in Euclidean space of any constant dimension d, any $\epsilon > 0$, and any parameter $\rho \geq 2$, there exists a $(1+\epsilon)$-spanner with $O(n)$ edges, degree $O(\rho)$, spanner diameter $O(\log_\rho n + \alpha(\rho))$, andweight $O(\rho \cdot \log_\rho n \cdot \mathrm{wt}(\mathrm{MST}))$, which can be constructed in time $O(n \log n)$.*

Given the lower bounds proved by Chan and Gupta [13] and Dinitz et al. [23], these results represent optimal tradeoffs in the entire range of the parameter ρ.

Fault-Tolerant Spanners

The concept of fault-tolerant spanners was first introduced by Levcopoulos et al. [28] in 1998: After one or more vertices or edges fail, the spanner should retain its good properties. In particular, there should still be a short path between any two vertices in what remains of the spanner after the fault. Czumaj and Zhao [19] showed that a greedy approach produces a k-vertex (or k-edge) fault-tolerant geometric t-spanner with degree $O(k)$ and total weight $O(k^2 \cdot \mathrm{wt}(\mathrm{MST}(S)))$; these bounds are asymptotically optimal. Chan et al. [15] used a "standard net-tree with cross-edge framework" developed by [14, 26] to design an algorithm that produces a k-vertex (or k-edge) fault-tolerant geometric $(1+\epsilon)$-spanner with degree $O(k^2)$, diameter $O(\log n)$, and total weight $O(k^2 \log n \cdot \mathrm{wt}(\mathrm{MST}(S)))$. Such a spanner can be constructed in $O(n \log n + k^2 n)$ time.

For geometric spanners, it is natural to consider *region faults*, i.e., faults that destroy all vertices and edges intersecting some geometric fault region. For a fault region F, let $G \ominus F$ be the part of G that remains after the points from S inside F and all edges that intersect F have been removed from the graph; see Fig. 1b. Abam et al. [2] showed how to construct region-fault tolerant t-spanners of size $O(n \log n)$ that are fault tolerant to any convex region fault. If one is allowed to use Steiner points, then a linear size t-spanner can be achieved.

Spanners Among Obstacles

The visibility graph of a set of pairwise non-intersecting polygons is a graph of intervisible locations. Each polygonal vertex is a vertex in the graph and each edge represents a visible

connection between them, that is, if two vertices can see each other, an edge is drawn between them. This graph is useful since it contains the shortest obstacle avoiding path between any pair of vertices.

Das [20] showed that a t-spanner of the visibility graph of a point set in the Euclidean plane can be constructed by using the Θ-graph approach followed by a pruning step. The obtained graph has linear size and constant degree.

Dynamic and Kinetic Spanners

Arya et al. [9] designed a data structure of size $O(n \log^d n)$ that maintains the skip-list spanner, described in section "The Θ-Graph," in $O(\log^d n \log\log n)$ expected amortized time per insertion and deletion in the model of random updates.

Gao et al. [26] showed how to maintain a t-spanner of size $O\left(\frac{n}{(t-1)^d}\right)$ and maximum degree $O\left(\frac{1}{(t-2)^d}\log\alpha\right)$, in time $O\left(\frac{\log\alpha}{(t-1)^d}\right)$ per insertion and deletion, where α denotes the aspect ratio of S, i.e., the ratio of the maximum pairwise distance to the minimum pairwise distance. The idea is to use an hierarchical structure T with $O(\log\alpha)$ levels, where each level contains a set of centers (subset of S). Each vertex v on level i in T is connected by an edge to all other vertices on level i within distance $O\left(\frac{2^i}{t-1}\right)$ of v. The resulting graph is a t-spanner of S and it can be maintained as stated above. The approach can be generalized to the kinetic case so that the total number of events in maintaining the spanner is $O(n^2 \log n)$ under pseudo-algebraic motion. Each event can be updated in $O\left(\frac{\log\alpha}{(t-1)^d}\right)$ time.

The problem of maintaining a spanner under insertions and deletions of points was settled by Gottlieb and Roditty [5]: For every set of n points in a metric space of bounded doubling dimension, there exists a $(1+\epsilon)$-spanner whose maximum degree is $O(1)$ and that can be maintained under insertions and deletions of points, in $O(\log n)$ time per operation.

Recently several papers have considered the kinetic version of the spanner construction problem. Abam et al. [1, 3] gave the first data structures for maintaining the Θ-graph, which was later improved by Rahmati et al. [32]. Assuming the trajectories of the points can be described by polynomials whose degrees are at most a constant s, the data structure uses $O(n \log^d n)$ space and handles $O(n^2)$ events with a total cost of $O\left(n\lambda_{2s+2}(n)\log^{d+1} n\right)$, where $\lambda_{2s+2}(n)$ is the maximum length of Davenport-Schinzel sequences of order $2s+2$ on n symbols. The kinetic data structure is compact, efficient, responsive (in an amortized sense), and local.

Applications

The construction of sparse spanners has been shown to have numerous application areas such as metric space searching [31], which includes query by content in multimedia objects, text retrieval, pattern recognition, and function approximation. Another example is broadcasting in communication networks [29]. Several well-known theoretical results also use the construction of t-spanners as a building block, for example, Rao and Smith [33] made a breakthrough by showing an optimal $O(n \log n)$-time approximation scheme for the well-known Euclidean *traveling salesperson problem*, using t-spanners (or banyans). Similarly, Czumaj and Lingas [18] showed approximation schemes for minimum-cost multi-connectivity problems in geometric networks.

Open Problems

A few open problems are mentioned below:

1. Determine if there exists a fault-tolerant t-spanner of linear size for convex region faults.
2. Can the k-vertex fault-tolerant spanner be computed in $O(n \log n + kn)$ time?

Experimental Results

The problem of constructing spanners has received considerable attention from a theoretical perspective but not much attention from a practical or experimental perspective. Navarro and Paredes [31] presented four heuristics for point sets in high-dimensional space ($d = 20$) and showed by empirical methods that the running time was $O(n^{2.24})$ and the number of edges in the produced graphs was $O(n^{1.13})$. Farshi and Gudmundsson [25] performed a thorough comparison of the construction algorithms discussed in section "Spanners of Points in Euclidean Space." The results showed that the spanner produced by the original greedy algorithm is superior compared to the graphs produced by the other approaches discussed in section "Spanners of Points in Euclidean Space" when it comes to number of edges, maximum degree, and weight. However, the greedy algorithm requires $O(n^2 \log n)$ time [10] and uses quadratic space, which restricted experiments in [25] to instances containing at most 13,000 points. Alewijnse et al. [4] showed how to reduce the space usage to linear only paying an additional $O(\log n)$ factor in the running time. In their experiments, they could handle more than a million points. In a follow-up paper, Bouts et al. [11] gave further experimental improvements.

Cross-References

- Applications of Geometric Spanner Networks
- Dilation of Geometric Networks
- Simple Algorithms for Spanners in Weighted Graphs
- Single-Source Shortest Paths
- Sparse Graph Spanners
- Well Separated Pair Decomposition

Recommended Reading

1. Abam MA, de Berg M (2011) Kinetic spanners in $\mathbb{R}^d$. Discret Comput Geom 45(4):723–736
2. Abam MA, de Berg M, Farshi M, Gudmundsson J (2009) Region-fault tolerant geometric spanners. Discret Comput Geom 41:556–582
3. Abam MA, de Berg M, Gudmundsson J (2010) A simple and efficient kinetic spanner. Comput Geom 43(3):251–256
4. Alewijnse SPA, Bouts QW, ten Brink AP, Buchin K (2013) Computing the greedy spanner in linear space. In: Bodlaender HL, Italiano GF (eds) 21st annual European symposium on algorithms. Lecture notes in computer science, vol 8125. Springer, Heidelberg, pp 37–48
5. Arikati SR, Chen DZ, Chew LP, Das G, Smid M, Zaroliagis CD (1996) Planar spanners and approximate shortest path queries among obstacles in the plane. In: Proceedings of 4th European symposium on algorithms. Lecture notes in computer science, vol 1136. Springer Berlin/Heidelberg, pp 514–528
6. Arya S, Smid M (1997) Efficient construction of a bounded-degree spanner with low weight. Algorithmica 17:33–54
7. Arya S, Mount DM, Smid M (1994) Randomized and deterministic algorithms for geometric spanners of small diameter. In: Proceedings of 35th IEEE symposium on foundations of computer science, Milwaukee, pp 703–712
8. Arya S, Das G, Mount DM, Salowe JS, Smid M (1995) Euclidean spanners: short, thin, and lanky. In: Proceedings of 27th ACM symposium on theory of computing, Las Vegas, pp 489–498
9. Arya S, Mount DM, Smid M (1999) Dynamic algorithms for geometric spanners of small diameter: randomized solutions. Comput Geom – Theory Appl 13(2):91–107
10. Bose P, Carmi P, Farshi M, Maheshwari A, Smid MHM (2010) Computing the greedy spanner in near-quadratic time. Algorithmica 58(3):711–729
11. Bouts QW, ten Brink AP, Buchin K (2014) A framework for computing the greedy spanner. In: Cheng SW, Devillers O (eds) Symposium on computational geometry, Kyoto. ACM, pp 11–20
12. Callahan PB, Kosaraju SR (1995) A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. J ACM 42:67–90
13. Chan HTH, Gupta A (2009) Small hop-diameter sparse spanners for doubling metrics. Discret Comput Geom 41(1):28–44
14. Chan HTH, Gupta A, Maggs B, Zhou S (2005) On hierarchical routing in doubling metrics. In: Symposium on discrete algorithms, Vancouver. ACM, pp 762–771
15. Chan HTH, Li M, Ning L, Solomon S (2013) New doubling spanners: better and simpler. In: Automata, languages, and programming. Springer, Berlin/Heidelberg, pp 315–327
16. Chandra B, Das G, Narasimhan G, Soares J (1992) New sparseness results on graph spanners. In: Proceedings of 8th annual symposium on computational geometry, Berlin, pp 192–201
17. Chandra B, Das G, Narasimhan G, Soares J (1995) New sparseness results on graph spanners. Int J Comput Geom Appl 5:124–144

18. Czumaj A, Lingas A (2000) Fast approximation schemes for Euclidean multi-connectivity problems. In: Proceedings of 27th international colloquium on automata, languages and programming. Lecture notes in computer science, vol 1853. Springer, Berlin/Heidelberg, pp 856–868
19. Czumaj A, Zhao H (2004) Fault-tolerant geometric spanners. Discret Comput Geom 32(2): 207–230
20. Das G (1997) The visibility graph contains a bounded-degree spanner. In: Proceedings of 9th Canadian conference on computational geometry, Kingston
21. Das G, Narasimhan G (1997) A fast algorithm for constructing sparse Euclidean spanners. Int J Comput Geom Appl 7:297–315
22. Das G, Narasimhan G, Salowe J (1995) A new way to weigh malnourished Euclidean graphs. In: Proceedings of 6th ACM-SIAM symposium on discrete algorithms, San Francisco, pp 215–222
23. Dinitz Y, Elkin M, Solomon S (2010) Low-light trees, and tight lower bounds for Euclidean spanners. Discret Comput Geom 43(4):736–783
24. Elkin M, Solomon S (2013) Optimal Euclidean spanners: really short, thin and lanky. In: Proceedings of the forty-fifth annual ACM symposium on theory of computing, Palo Alto, pp 645–654
25. Farshi M, Gudmundsson J (2009) Experimental study of geometric t-spanners. ACM J Exp Algorithmics 14(1):3–39
26. Gao J, Guibas LJ, Nguyen A (2004) Deformable spanners and applications. In: Proceedings of 20th ACM symposium on computational geometry, Brooklyn, pp 190–199
27. Gudmundsson J, Levcopoulos C, Narasimhan G (2002) Improved greedy algorithms for constructing sparse geometric spanners. SIAM J Comput 31(5):1479–1500
28. Levcopoulos C, Narasimhan G, Smid M (2002) Improved algorithms for constructing fault-tolerant spanners. Algorithmica 32(1):144–156
29. Li XY (2003) Applications of computational geomety in wireless ad hoc networks. In: Cheng XZ, Huang X, Du DZ (eds) Ad Hoc wireless networking. Kluwer, Dordrecht, pp 197–264
30. Narasimhan G, Smid M (2007) Geometric spanner networks. Cambridge University Press, Cambridge/New York
31. Navarro G, Paredes R (2003) Practical construction of metric t-spanners. In: Proceedings of 5th workshop on algorithm engineering and experiments, Baltimore, Maryland. SIAM Press, pp 69–81
32. Rahmati Z, Abam MA, King V, Whitesides S (2014) Kinetic data structures for the semi-Yao graph and all nearest neighbors in $\mathbb{R}^d$. In: He M, Zeh N (eds) Canadian conference on computational geometry
33. Rao S, Smith WD (1998) Approximating geometrical graphs via spanners and banyans. In: Proceedings of 30th ACM symposium on theory of computing, Dallas, pp 540–550

Global Minimum Cuts in Surface-Embedded Graphs

Erin W. Chambers[1], Jeff Erickson[2], Kyle Fox[3], and Amir Nayyeri[4]
[1]Department of Computer Science and Mathematics, Saint Louis University, St. Louis, MO, USA
[2]Department of Computer Science, University of Illinois, Urbana, IL, USA
[3]Institute for Computational and Experimental Research in Mathematics, Brown University, Providence, RI, USA
[4]Department of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA

Keywords

Covering spaces; Fixed-parameter tractability; Graph embedding; Homology; Minimum cuts; Topological graph theory

Years and Authors of Summarized Original Work

2009; Chambers, Erickson, Nayyeri
2011; Erickson, Nayyeri
2012; Erickson, Fox, Nayyeri

Problem Definition

Given a graph G in which every edge has a nonnegative capacity, the goal of the minimum-cut problem is to find a subset of edges of G with minimum total capacity whose deletion disconnects G. The closely related minimum (s,t)-cut problem further requires two specific vertices s and t to be separated by the deleted edges. Minimum cuts and their generalizations play a central role in divide-and-conquer and network optimization algorithms.

The fastest algorithms known for computing minimum cuts in arbitrary graphs run in roughly $O(mn)$ time for graphs with n vertices

and m edges. However, even faster algorithms are known for graphs with additional topological structure. This entry sketches algorithms to compute minimum cuts in near-linear time when the input graph can be drawn on a surface with bounded genus – informally, a sphere with a bounded number of handles.

Problem 1 (Minimum (s,t)-Cut)
INPUT: *An undirected graph $G = (V, E)$ embedded on an orientable surface of genus g, a nonnegative capacity function $c: E \to \mathbb{R}$, and two vertices s and t.* OUTPUT: *A minimum-capacity (s, t)-cut in G.*

Problem 2 (Global Minimum Cut)
INPUT: *An undirected graph $G = (V, E)$ embedded on an orientable surface of genus g and a nonnegative capacity function $c: E \to \mathbb{R}$.* OUTPUT: *A minimum-capacity cut in G.*

Key Results

Topological Background

A surface is a compact space in which each point has a neighborhood homeomorphic to either the plane or a closed half plane. Points with half-plane neighborhoods comprise the boundary of the surface, which is the union of disjoint simple cycles. The genus is the maximum number of disjoint simple cycles whose deletion leaves the surface connected. A surface is orientable if it does not contain a MÃbius band. An embedding is a drawing of a graph on a surface, with vertices drawn as distinct points and edges as simple interior-disjoint paths, whose complement is a collection of disjoint open disks called the faces of the embedding.

An even subgraph of G is a subgraph in which every vertex has even degree; each component of an even subgraph is Eulerian. Two even subgraphs of an embedded graph G are $\mathbb{Z}_2$-homologous, or in the same $\mathbb{Z}_2$-homology class, if their symmetric difference is the boundary of a subset of the surface. If G is embedded on a surface of genus g with $b > 0$ boundary cycles, the even subgraphs of G fall into 2^{2g+b-1} $\mathbb{Z}_2$-homology classes. An even subgraph of G is $\mathbb{Z}_2$-minimal if it has minimum total cost within its $\mathbb{Z}_2$-homology class. Each component of a $\mathbb{Z}_2$-minimal even subgraph is itself $\mathbb{Z}_2$-minimal.

Every embedded graph G has a dual graph G^*, embedded on the same surface, whose vertices correspond to faces of G and whose edges correspond to pairs of faces that share an edge in G. The cost of a dual edge in G^* is the capacity of the corresponding primal edge in G.

Duality maps cut to certain sets of cycles and vice versa. For example, the minimum-capacity (s, t)-cut in any planar graph G is dual to the minimum-cost cycle in G^* that separates the dual faces s^* and t^*. If we remove s^* and t^* from the sphere, the dual of the minimum cut is the shortest generating cycle of the resulting annulus [11, 15]. More generally, let X denote the set of edges that cross some minimum (s, t)-cut in an embedded graph G, and let X^* denote the corresponding subgraph of the dual graph G^*. Then X^* is a minimum-cost even subgraph of G^* that separates s^* and t^*. If we remove s^* and t^* from the surface, X^* becomes a $\mathbb{Z}_2$-minimal subgraph homologous with the boundary of s^*.

Crossing Sequences

Our first algorithm [6] reduces computing a minimum (s, t)-cut in a graph embedded on a genus-g surface to $g^{O(g)}$ instances of the planar minimum-cut problem.

The algorithm begins by constructing a collection A of $2g + 1$ paths in G^*, called a greedy system of arcs, with three important properties. First, the endpoints of each path are incident to the boundary faces s^* and t^*. Second, each path is composed of two shortest paths plus at most one additional edge. Finally, the complement $\Sigma \setminus A$ of the paths is a topological open disk. A greedy system of arcs can be computed in $O(gn)$ time [5, 9].

We regard each component of X^* as a closed walk, and we enumerate all possible sequences of crossings between the components of X^* with the arcs in A. The components of any $\mathbb{Z}_2$-minimal even subgraph cross any shortest path, and there-

fore any arc in A, at most $O(g)$ times. It follows that we need to consider at most $g^{O(g)}$ crossing sequences, each of length at most $O(g^2)$. Following Kutz [13], the shortest closed walk with a given crossing sequence is the shortest generating cycle in an annulus obtained by gluing together $O(g^2)$ copies of the disk $\Sigma \setminus A$, which can be computed in $O(g^2 n \log\log n)$ time using the planar minimum-cut algorithm of Italiano et al. [12]. The overall running time of this algorithm is $g^{O(g)} n \log\log n$.

Surprisingly, a reduction from MAXCUT implies that finding the minimum-cost even subgraph in an arbitrary $\mathbb{Z}_2$-homology class is NP-hard. Different reductions imply that it is NP-hard to find the minimum-cost *closed walk* [5] or *simple cycle* [2] in a given $\mathbb{Z}_2$-homology class.

$\mathbb{Z}_2$-Homology Cover

Our second algorithm [9] finds the minimum-cost closed walks in G^* in every $\mathbb{Z}_2$-homology class by searching a certain covering space and then assembles X^* from these closed walks via dynamic programming.

As in our first algorithm, we first compute a greedy system of arcs A. The homology class of any cycle γ is determined by the parity of the number of crossings of γ with each arc in A. Each arc $\alpha_i \in A$ appears as two paths α_i^+ and α_i^- on the boundary of the disk $D = \Sigma \setminus A$.

We then construct a new surface $\overline{\Sigma}$, called the $\mathbb{Z}_2$-*homology cover* of Σ, by gluing together several copies of D as follows. We associate each homology class $h \in \mathbb{Z}_2^{2g+1}$ with a vector of $2g+1$ bits. Let $h \wedge i$ denote the bit vector obtained from h by flipping its ith bit. For each bit vector h, we construct a copy D_h of D; let $\alpha_{i,h}^+$ and $\alpha_{i,h}^-$ denote the copies of α_i^+ and α_i^- on the boundary of D_h. Finally, we construct $\overline{\Sigma}$ by identifying the paths $\alpha_{i,h}^+$ and $\alpha_{i,h\wedge i}^-$ for each homology class h and index i.

This construction also yields a graph $\overline{G}$ embedded in $\overline{\Sigma}$, with 2^{2g+1} vertices v_h and edges e_h for each vertex v and edge e of G^*. Each edge e_h of $\overline{G}$ inherits the cost of the corresponding edge e in G^*. Any walk in $\overline{G}$ projects to a walk in G^* by dropping subscripts; in particular, any walk in $\overline{G}$ from v_0 to v_h projects to a closed walk in G^* with homology class h that starts and ends at vertex v. Conversely, the *shortest* closed walk in G^* in any homology class h is the projection of the *shortest* path from v_0 to v_h, for some vertex v.

Any cycle in any nontrivial homology class crosses some path α_i, an odd number of times, and therefore at least once. To find all such cycles for each index i, we slice $\overline{G}$ along the lifted path $\alpha_{i,0}$ to obtain a new boundary cycle, and then compute the shortest path from each vertex v_0 on this cycle to every other vertex v_h in $2^{O(g)} n \log n$ time, using an algorithm of Chambers et al. [3]. Altogether, we find the shortest closed walk in every $\mathbb{Z}_2$-homology class in $2^{O(g)} n \log n$ time. The dual minimum cut X^* can then be built from these $\mathbb{Z}_2$-minimal cycles in $2^{O(g)}$ additional time via dynamic programming.

Global Minimum Cuts

Our final result generalizes the recent $O(n \log\log n)$-time algorithm for planar graphs by Lacki and Sankowski [14], which in turn relies on the $O(n \log\log n)$-time algorithm for planar minimum (s,t)-cuts of Italiano et al. [12].

The global minimum cut X in a surface graph G is dual to the minimum-cost nonempty separating subgraph of the dual graph G^*. In particular, if G is planar, X is dual to the shortest nonempty cycle in G^*. There are two cases to consider: either X^* is a simple contractible cycle, or it isn't. We describe two algorithms, one of which is guaranteed to return the minimum-cost separating subgraph.

To handle the contractible cycle case, we first slice the surface Σ to make it planar, first along the shortest non-separating cycle α in G^*, which we compute in $g^{O(g)} n \log\log n$ time using a variant of our crossing sequence algorithm, and then along a greedy system of arcs A connecting the resulting boundary cycles. Call the resulting planar graph D; each edge of $\alpha \cup A$ appears as two edges on the boundary of D. Let e^+ and e^- be edges on the boundary of D corresponding to some edge e of α. Using the planar algorithm of Lacki and Sankowski [14], we find the shortest cycle γ^+ in $D \setminus e^+$ and the shortest cycle γ^- in $D \setminus e^-$. The shorter of these two cycles projects to a closed walk γ in the original dual graph G^*.

Results of Cabello [2] imply that if γ is a simple cycle, it is the shortest contractible simple cycle in G^*; otherwise, X^* is not a simple cycle.

Our second algorithm begins by enumerating all $2^{O(g)}$ $\mathbb{Z}_2$-minimal even subgraphs in G^* in $g^{O(g)}n \log\log n$ time, using our crossing sequence algorithm. Our algorithm marks the faces on either side of an arbitrary edge of each $\mathbb{Z}_2$-minimal even subgraphs in G^*. If X^* is not a simple contractible cycle, then some pair of marked faces must be separated by X^*. In other words, in $g^{O(g)}n \log\log n$ time, we identify a set T of $2^{O(g)}$ vertices of G, at least two of which are separated by the global minimum cut. Thus, if we fix an arbitrary source vertex s and compute the minimum (s,t)-cut for each vertex $t \in T$ in $g^{O(g)}n \log\log n$ time, the smallest such cut is the global minimum cut X.

Open Problems

Extending these algorithms to *directed* surface graphs remains an interesting open problem; currently the only effective approach known is to compute a maximum (s,t)-flow and apply the maxflow-mincut theorem. The recent algorithm of Borradaile and Klein [1] computes maximum flows in directed *planar* graphs in $O(n \log n)$ time. For higher-genus graphs, Chambers et al. [7] describes maximum-flow algorithms that run in $g^{O(g)}n^{3/2}$ time for arbitrary capacities and in $O(g^8 n \log^2 n \log^2 C)$ for integer capacities that sum to C.

Another open problem is reducing the dependencies on the genus from exponential to polynomial. Even though there are near-quadratic algorithms to compute minimum cuts, the only known approach to achieving near-linear time for bounded-genus graphs with weighted edges is to solve an NP-hard problem.

Finally, it is natural to ask whether minimum cuts can be computed quickly in other minor-closed families of graphs, for which embeddings on to bounded-genus surfaces may not exist. Such results already exist for one-crossing-minor-free families [4] and in particular, graphs of bounded treewidth [10].

Cross-References

- Max Cut
- Separators in Graphs
- Shortest Paths in Planar Graphs with Negative Weight Edges
- Sparsest Cut

Recommended Reading

1. Borradaile G, Klein P (2009) An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. J ACM 56(2):9:1–30
2. Cabello S (2010) Finding shortest contractible and shortest separating cycles in embedded graphs. ACM Trans Algorithms 6(2):24:1–24:18
3. Cabello S, Chambers EW, Erickson J (2013) Multiple-source shortest paths in embedded graphs. SIAM J Comput 42(4):1542–1571
4. Chambers E, Eppstein D (2013) Flows in one-crossing-minor-free graphs. J Graph Algorithms Appl 17(3):201–220. doi:10.7155/jgaa.00291
5. Chambers EW, Colin de Verdière É, Erickson J, Lazarus F, Whittlesey K (2008) Splitting (complicated) surfaces is hard. Comput Geom Theory Appl 41(1–2):94–110
6. Chambers EW, Erickson J, Nayyeri A (2009) Minimum cuts and shortest homologous cycles. In: Proceedings of the 25th annual symposium computational geometry, Aarhus, pp 377–385
7. Chambers EW, Erickson J, Nayyeri A (2012) Homology flows, cohomology cuts. SIAM J Comput 41(6):1605–1634
8. Erickson J (2012) Combinatorial optimization of cycles and bases. In: Zomorodian A (ed) Advances in applied and computational topology. Invited survey for an AMS short course on computational topology at the 2011 joint mathematics meetings, New Orleans. Proceedings of symposia in applied mathematics, vol 70. American Mathematical Society, pp 195–228
9. Erickson J, Nayyeri A (2011) Minimum cuts and shortest non-separating cycles via homology covers. In: Proceedings of the 22nd annual ACM-SIAM symposium on discrete algorithms, San Francisco, pp 1166–1176
10. Hagerup T, Katajainen J, Nishimura N, Ragde P (1998) Characterizing multiterminal flow networks and computing flows in networks of small treewidth. J Comput Syst Sci 57(3):366–375
11. Itai A, Shiloach Y (1979) Maximum flow in planar networks. SIAM J Comput 8:135–150
12. Italiano GF, Nussbaum Y, Sankowski P, Wulff-Nilsen C (2011) Improved algorithms for min cut and max flow in undirected planar graphs. In: Proceedings of the 43rd annual ACM symposium theory of computing, San Jose, pp 313–322

13. Kutz M (2006) Computing shortest non-trivial cycles on orientable surfaces of bounded genus in almost linear time. In: Proceedings of the 22nd annual symposium on computational geometry, Sedona, pp 430–438
14. Łącki J, Sankowski P (2011) Min-cuts and shortest cycles in planar graphs in $O(n \log\log n)$ time. In: Proceedings of the 19th annual European symposium on algorithms, Saarbrücken. Lecture notes in computer science, vol 6942. Springer, pp 155–166
15. Reif J (1983) Minimum s-t cut of a planar undirected network in $O(n \log^2 n)$ time. SIAM J Comput 12:71–81

Global Routing

Minsik Cho[1] and David Z. Pan[2]
[1]IBM T. J. Watson Research Center, Yorktown Heights, NY, USA
[2]Department of Electrical and Computer Engineering, University of Texas, Austin, TX, USA

Keywords

EDA; Global routing; Graph search; Layout; Mathematical programming; Netlist; Shortest path

Years and Authors of Summarized Original Work

2006; Cho, Pan

Problem Definition

Global routing is a key step in VLSI physical design after floor planning and placement. Its main goal is to reduce the overall routing complexity and guide the detailed router by planning the approximate routing path of each net. The commonly used objectives during global routing include minimizing total wirelength, mitigating routing congestion, or meeting routing resource constraints. If timing critical paths are known, they can also be put in the design objectives during global routing, along with other metrics such as manufacturability and noise.

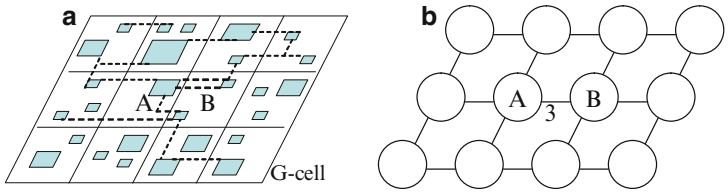
The global routing problem can be formulated using graph models. For a given netlist graph $G(C, N)$, vertices C represent pins on placed objects such as standard cells or IP blocks, and edges N represent nets connecting the pins. The routing resources on a chip can be modeled in another graph $G(V, E)$ by dividing the entire global routing region into a set of smaller regions, so-called global routing cells (G-cells), where $v \in V$ represents a G-cell and $e \in E$ represents the boundary between two adjacent G-cells with a given routing capacity (c_e). Figure 1 shows how the chip can be abstracted into a 2-dimensional global routing graph. Such abstraction can be easily extended to 3-dimensional global routing graph to perform layer assignment (e.g., [15]). Since all standard cells and IP blocks are placed before the global routing stage (e.g., C can be mapped into V), the goal of global routing is to find G-cell to G-cell paths for N while trying to meet certain objectives such as routability optimization and wirelength minimization.

A straightforward mathematical optimization for global routing can be formulated as a 0-1 integer linear programming (ILP). Let R_i be a set of Steiner trees on G for net $n_i \in N$, and $x_{i,j}$ be the binary variable to indicate whether $r_{i,j} \in R_i$ is selected as the routing solution. Then an example ILP formulation can be written as follows:

The above formulation minimizes the total routing capacity utilization under the maximum routing capacity constraint for each edge $e \in E$. In fact, minimizing the total routing capacity is equivalent to minimizing total wirelength, because a unit wirelength in the global routing utilizes one routing resource (i.e., crossing the boundary between two adjacent G-cells). Other objectives/constraints can include timing optimization, noise reduction [10, 14], or manufacturability (e.g., CMP) [8].

Key Results

The straightforward formulation using ILP, e.g., as in Fig. 2, is NP-complete which cannot be solved efficiently for modern VLSI designs. One common technique to solve ILP is to use linear



Global Routing, Fig. 1 Graph model for global routing. (**a**) Real circuit with G-cells. (**b**) Global routing graph

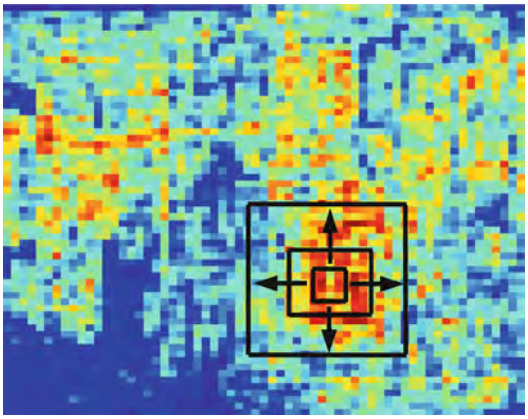
$$\mathbf{min}: \sum_{e \in E} u_e$$

$$\mathbf{s.t}: \sum_{r_{i,j} \in R_i} x_{i,j} = 1, \forall n_i \in N$$

$$u_e = \sum_{(i,j):e \in r_{i,j}} x_{i,j}, \forall e \in E$$

$$u_e \leq c_e, \forall e \in E$$

Global Routing, Fig. 2 An example of global routing formulation using ILP



Global Routing, Fig. 3 Box expansion for Progressive ILP during BoxRouter

programming relaxation where the binary variables are made continuous, $x_{i,j} \in [0, 1]$, and that can be solved in polynomial time. Once a linear programming solution is obtained, rounding technique is used to find the binary solution. Another technique is a hierarchical divide-and-conquer scheme to limit the complexity, which solves many independent subproblems of similar sizes. These approaches may suffer from large amount of rounding errors or lack of interactions between subproblems, resulting in poor quality.

BoxRouter [1, 6] proposed a new approach to divide the entire routing region into a set of synergistic subregions. The key idea in BoxRouter is the progressive ILP based on the routing box expansion, which pushes congestion outward progressively from the highly congested region. Unlike conventional hierarchical divide-and-conquer approach, BoxRouter solves a sequence of ILP problems where an early problem is a subset of a later problem. BoxRouter progressively applies box expansion to build a sequence of ILP problems starting from the most congested region which is obtained through a very fast pre-routing stage. Figure 3 illustrates the concept of box expansion.

The advantage of BoxRouter over conventional approach is that each problem synergically reflects the decisions made so far by taking the previous solutions as constraints, in order to enhance congestion distribution and shorten the wirelength. In that sense, the first ILP problem has the largest flexibility which motivates the box expansion originating from the most congested region. Even though the last box can cover the whole design, the effective ILP size remains tractable in BoxRouter, as ILP is only performed on the wires between two subsequent boxes.

Compared with the formulation in Fig. 2 which directly minimizes the total wirelength, progressive ILP in BoxRouter maximizes the completion rate (e.g., minimizing unrouted nets) which can be more important and practical than minimizing total wirelength as shown in Fig. 4. Wirelength minimization in BoxRouter is indirectly achieved by allowing only the minimum rectilinear Steiner trees for each binary variable (i.e., $x_{i,j}$) and being augmented with the post-maze routing step. Such change in the objective function provides higher computation efficiency: it is found that the BoxRouter formulation can be solved significantly faster

$$\max : \sum x_{i,j}$$

$$\text{s.t} : \sum_{r_{i,j} \in R_i} x_{i,j} \leq 1, \forall n_i \in N$$

$$\sum_{(i,j):e \in r_{i,h}} x_{i,j} \leq c_e, \forall e \in E$$

Global Routing, Fig. 4 Global routing formulation in BoxRouter for minimal unrouted nets

than the traditional formulations due to its simple and well-exploited knapsack structure [7].

In case some nets remain unrouted after each ILP problem either due to insufficient routing resources inside a box or a limited number of Steiner graphs for each net, BoxRouter applies adaptive maze routing which penalizes using routing resources outside the current box, in order to reserve them for subsequence problems. Based on the new ILP techniques, BoxRouter has obtained much better results than previous state-of-the-art global routers [4, 9] and motivated many further studies in global routing (e.g., [5, 11–13, 15]) and global routing contests at ISPD 2007 and ISPD 2008 [2, 3].

Cross-References

- ▸ Circuit Placement
- ▸ Rectilinear Steiner Tree
- ▸ Routing

Recommended Reading

1. (2007) Minsik Cho, Kun Yuan, Katrina Lu and David Z. Pan http://www.cerc.utexas.edu/utda/download/BoxRouter.htm
2. (2007) http://archive.sigda.org/ispd2007/contest.html
3. (2008) http://archive.sigda.org/ispd2008/contests/ispd08rc.html
4. Albrecht C (2001) Global routing by new approximation algorithms for multicommodity flow. IEEE Trans Comput-Aided Des Integr Circuits Syst 20(5):622–632
5. Chang YJ, Lee YT, Wang TC (2008) NTHU-Route 2.0: a fast and stable global router. In: Proceedings of the international conference on computer aided design, San Jose, pp 338–343
6. Cho M, Pan DZ (2006) BoxRouter: a new global router based on box expansion and progressive ILP. In: Proceedings of the design automation conference, San Francisco, pp 373–378
7. Cho M, Pan DZ (2007) BoxRouter: a new global router based on box expansion and progressive ILP. IEEE Trans Comput-Aided Des Integr Circuits Syst 26(12):2130–2143
8. Cho M, Xiang H, Puri R, Pan DZ (2006) Wire density driven global routing for CMP variation and timing. In: Proceedings of the international conference on computer aided design, San Jose
9. Kastner R, Bozorgzadeh E, Sarrafzadeh M (2002) Pattern routing: use and theory for increasing predictability and avoiding coupling. IEEE Trans Comput-Aided Des Integr Circuits Syst 21(7):777–790
10. Kay R, Rutenbar RA (2000) Wire packing: a strong formulation of crosstalk-aware chip-level track/layer assignment with an efficient integer programming solution. In: Proceedings of the international symposium on physical design, San Diego
11. Moffitt MD (2008) Maizerouter: engineering an effective global router. In: Proceedings of the Asia and South Pacific design automation conference, Seoul
12. Ozdal M, Wong M (2007) Archer: a history-driven global routing algorithm. In: Proceedings of the international conference on computer aided design, San Jose, pp 488–495
13. Pan M, Chu C (2007) Fastroute 2.0: a high-quality and efficient global router. In: Proceedings of the Asia and South Pacific design automation conference, Yokohama
14. Wu D, Hu J, Mahapatra R, Zhao M (2004) Layer assignment for crosstalk risk minimization. In: Proceedings of the Asia and South Pacific design automation conference, Yokohama
15. Wu TH, Davoodi A, Linderoth JT (2009) GRIP: scalable 3D global routing using integer programming. In: Proceedings of the design automation conference, San Francisco, pp 320–325

Gomory-Hu Trees

Debmalya Panigrahi
Department of Computer Science, Duke University, Durham, NC, USA

Keywords

Cut trees; Minimum *s*-*t* cut; Undirected graph connectivity

Years and Authors of Summarized Original Work

2007; Bhalgat, Hariharan, Kavitha, Panigrahi

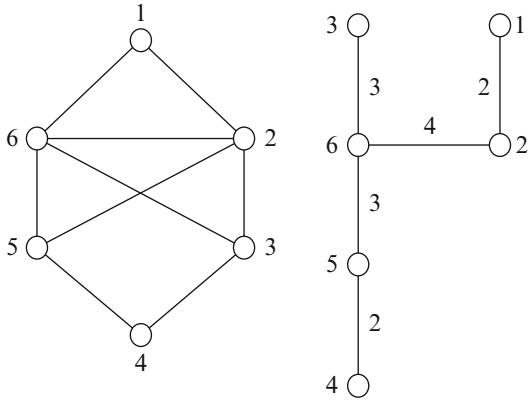
Problem Definition

Let $G = (V, E)$ be an undirected graph with $|V| = n$ and $|E| = m$. The edge connectivity of two vertices $s, t \in V$, denoted by $\lambda(s, t)$, is defined as the size of the smallest cut that separates s and t; such a cut is called a minimum $s - t$ cut. Clearly, one can represent the $\lambda(s, t)$ values for all pairs of vertices s and t in a table of size $O(n^2)$. However, for reasons of efficiency, one would like to represent all the $\lambda(s, t)$ values in a more succinct manner. *Gomory-Hu trees* (also known as *cut trees*) offer one such succinct representation of linear (i.e., $O(n)$) space and constant (i.e., $O(1)$) lookup time. It has the additional advantage that apart from representing all the $\lambda(s, t)$ values, it also contains structural information from which a minimum $s - t$ cut can be retrieved easily for any pair of vertices s and t.

Formally, a Gomory-Hu tree $T = (V, F)$ of an undirected graph $G = (V, E)$ is a weighted undirected tree defined on the vertices of the graph such that the following properties are satisfied:

- For any pair of vertices $s, t \in V, \lambda(s, t)$ is equal to the minimum weight on an edge in the unique path connecting s to t in T. Call this edge $e(s, t)$. If there are multiple edges with the minimum weight on the s to t path in T, any one of these edges is designated as $e(s, t)$.
- For any pair of vertices s and t, the bipartition of vertices into components produced by removing $e(s, t)$ (if there are multiple candidates for $e(s, t)$, this property holds for each candidate edge) from T corresponds to a minimum $s - t$ cut in the original graph G.

To understand this definition better, consider the following example. Figure 1 shows an undirected graph and a corresponding Gomory-Hu tree. Focus on a pair of vertices, for instance, 3 and 5.



Gomory-Hu Trees, Fig. 1 An undirected graph (*left*) and a corresponding Gomory-Hu tree (*right*)

Clearly, the edge (6,5) of weight 3 is a minimum-weight edge on the 3 to 5 path in the Gomory-Hu tree. It is easy to see that $\lambda(3, 5) = 3$ in the original graph. Moreover, removing edge (6,5) in the Gomory-Hu tree produces the vertex bipartition ({1,2,3,6},{4,5}), which is a cut of size 3 in the original graph.

It is not immediate that such Gomory-Hu trees exist for all undirected graphs. In a classical result in 1961, Gomory and Hu [8] showed that not only do such trees exist for all undirected graphs but that they can also be computed using $n - 1$ minimum s-t cut (or equivalently maximum s-t flow) computations. In fact, a graph can have multiple Gomory-Hu trees.

All previous algorithms for constructing Gomory-Hu trees for undirected graphs used maximum-flow subroutines. Gomory and Hu gave an algorithm to compute a cut tree T using $n - 1$ maximum-flow computations and graph contractions. Gusfield [9] proposed an algorithm that does not use graph contractions; all $n - 1$ maximum-flow computations are performed on the input graph. Goldberg and Tsioutsiouliklis [7] did an experimental study of the algorithms due to Gomory and Hu and due to Gusfield for the cut tree problem and described efficient implementations of these algorithms. Examples were shown by Benczúr [1] that cut trees do not exist for directed graphs.

Any maximum-flow-based approach for constructing a Gomory-Hu tree would have a running

time of $(n-1)$ times the time for computing a single maximum flow. Till now, faster algorithms for Gomory-Hu trees were by-products of faster algorithms for computing a maximum flow. The current fastest $\tilde{O}(m+n\lambda(s,t))$ (polylog n factors ignored in $\tilde{O}$ notation) maximum-flow algorithm, due to Karger and Levine [11], yields the current best expected running time of $\tilde{O}(n^3)$ for Gomory-Hu tree construction on simple unweighted graphs with n vertices. Bhalgat et al. [2] improved this time complexity to $\tilde{O}(mn)$. Note that both Karger and Levine's algorithm and Bhalgat et al.'s algorithm are randomized Las Vegas algorithms. The fastest deterministic algorithm for the Gomory-Hu tree construction problem is a by-product of Goldberg and Rao's maximum-flow algorithm [6] and has a running time of $\tilde{O}(nm^{1/2}\min(m, n^{3/2}))$.

Since the publication of the results of Bhalgat et al. [2], it has been observed that the maximum-flow subroutine of Karger and Levine [11] can also be used to obtain an $\tilde{O}(mn)$ time Las Vegas algorithm for constructing the Gomory-Hu tree of an unweighted graph. However, this algorithm does not yield partial Gomory-Hu trees which are defined below. For planar undirected graphs, Borradaile et al. [3] gave an $\tilde{O}(mn)$ time algorithm for constructing a Gomory-Hu tree.

It is important to note that in spite of the tremendous recent progress in approximate maximum s-t flow (or approximate minimum s-t cut) computation, this does not immediately translate to an improved algorithm for approximate Gomory-Hu tree construction. This is because of two reasons: first, the property of uncrossability of minimum s-t cuts used by Gomory and Hu in their minimum s-t cut based cut tree construction algorithm does not hold for approximate minimum s-t cuts, and second, the errors introduced in individual minimum s-t cut computation can add up to create large errors in the Gomory-Hu tree.

Key Results

Bhalgat et al. [2] considered the problem of designing an efficient algorithm for constructing a Gomory-Hu tree on unweighted undirected graphs. The main theorem shown in this entry is the following.

Theorem 1 *Let $G = (V, E)$ be a simple unweighted graph with m edges and n vertices. Then a Gomory-Hu tree for G can be built in expected time $\tilde{O}(mn)$.*

Their algorithm is always faster by a factor of $\tilde{\Omega}(n^{2/9})$ (polylog n factors ignored in $\tilde{\Omega}$ notation) compared to the previous best algorithm.

Instead of using maximum-flow subroutines, they use a Steiner connectivity algorithm. The *Steiner connectivity* of a set of vertices S (called the *Steiner set*) in an undirected graph is the minimum size of a cut which splits S into two parts; such a cut is called a *minimum Steiner cut*. Generalizing a tree-packing algorithm given by Gabow [5] for finding the edge connectivity of a graph, Cole and Hariharan [4] gave an algorithm for finding the Steiner connectivity k of a set of vertices in either undirected or directed Eulerian unweighted graphs in $\tilde{O}(mk^2)$ time. (For undirected graphs, their algorithm runs a little faster in time $\tilde{O}(m + nk^3)$.) Bhalgat et al. improved this result and gave the following theorem.

Theorem 2 *In an undirected or directed Eulerian unweighted graph, the Steiner connectivity k of a set of vertices can be determined in time $\tilde{O}(mk)$.*

The algorithm in [4] was used by Hariharan et al. [10] to design an algorithm with expected running time $\tilde{O}(m + nk^3)$ to compute a *partial* Gomory-Hu tree for representing the $\lambda(s,t)$ values for all pairs of vertices s, t that satisfied $\lambda(s,t) \leq k$. Replacing the algorithm in [4] by the new algorithm for computing Steiner connectivity yields an algorithm to compute a partial Gomory-Hu tree in expected running time $\tilde{O}(m+nk^2)$. Bhalgat et al. showed that using a more detailed analysis, this result can be improved to give the following theorem.

Theorem 3 *The partial Gomory-Hu tree of an undirected unweighted graph to represent all $\lambda(s,t)$ values not exceeding k can be constructed in expected time $\tilde{O}(mk)$.*

Since $\lambda(s,t) < n$ for all s,t vertex pairs in an unweighted (and simple) graph, setting k to n in Theorem 3 implies Theorem 1.

Applications

Gomory-Hu trees have many applications in multiterminal network flows and are an important data structure in graph connectivity literature.

Open Problems

The problem of derandomizing the algorithm due to Bhalgat et al. [2] to produce an $\tilde{O}(mn)$ time deterministic algorithm for constructing Gomory-Hu trees for unweighted undirected graphs remains open. The other main challenge is to extend the results in [2] to weighted graphs.

Experimental Results

Goldberg and Tsioutsiouliklis [7] did an extensive experimental study of the cut tree algorithms due to Gomory and Hu [8] and that due to Gusfield [9]. They showed how to efficiently implement these algorithms and also introduced and evaluated heuristics for speeding up the algorithms. Their general observation was that while Gusfield's algorithm is faster in many situations, Gomory and Hu's algorithm is more robust. For more detailed results of their experiments, refer to [7].

No experimental results are reported for the algorithm due to Bhalgat et al. [2].

Recommended Reading

1. Benczúr AA (1995) Counterexamples for directed and node capacitated cut-trees. SIAM J Comput 24(3):505–510
2. Bhalgat A, Hariharan R, Kavitha T, Panigrahi D (2007) An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs. In: Proceedings of the 39th annual ACM symposium on theory of computing, San Diego
3. Borradaile G, Sankowski P, Wulff-Nilsen C (2010) Min st-cut Oracle for planar graphs with near-linear preprocessing time. In: Proceedings of the 51th annual IEEE symposium on foundations of computer science, Las Vegas, pp 601–610
4. Cole R, Hariharan R (2003) A fast algorithm for computing steiner edge connectivity. In: Proceedings of the 35th annual ACM symposium on theory of computing, San Diego, pp 167–176
5. Gabow HN (1995) A matroid approach to finding edge connectivity and packing arborescences. J Comput Syst Sci 50:259–273
6. Goldberg AV, Rao S (1998) Beyond the flow decomposition barrier. J ACM 45(5):783–797
7. Goldberg AV, Tsioutsiouliklis K (2001) Cut tree algorithms: an experimental study. J Algorithms 38(1):51–83
8. Gomory RE, Hu TC (1961) Multi-terminal network flows. J Soc Ind Appl Math 9(4):551–570
9. Gusfield D (1990) Very simple methods for all pairs network flow analysis. SIAM J Comput 19(1):143–155
10. Hariharan R, Kavitha T, Panigrahi D (2007) Efficient algorithms for computing all low s-t edge connectivities and related problems. In: Proceedings of the 18th annual ACM-SIAM symposium on discrete algorithms, New Orleans, pp 127–136
11. Karger D, Levine M (2002) Random sampling in residual graphs. In: Proceedings of the 34th annual ACM symposium on theory of computing, Montreal, pp 63–66

Grammar Compression

Hideo Bannai
Department of Informatics, Kyushu University, Fukuoka, Japan

Keywords

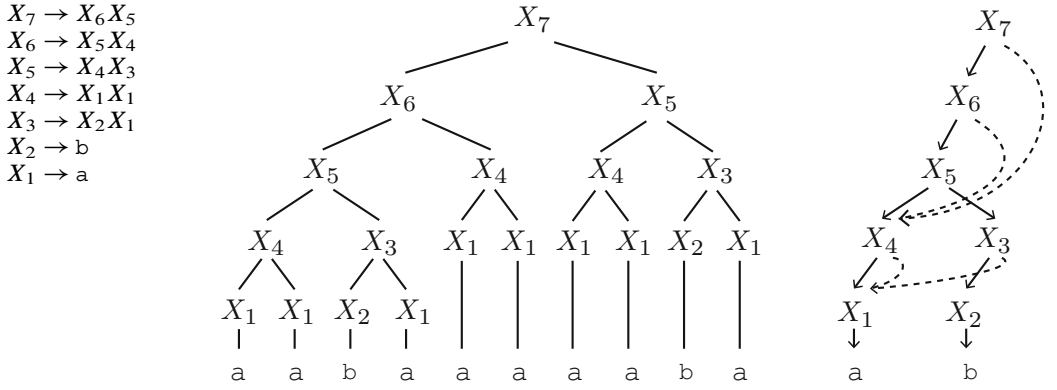
Balanced binary grammar; Context-free grammar; LZ77 factorization; Smallest grammar problem; Straight line program

Years and Authors of Summarized Original Work

2000; Kieffer, Yang
2003; Rytter
2004; Sakamoto et al.
2005; Charikar et al.

Problem Definition

Given an input string S, the grammar-based compression problem is to find a small



Grammar Compression, Fig. 1 (*Left*) an SLP G that represents string aabaaaaaba, where the variable X_7 is the start symbol. (*Center*) derivation tree of G. (*Right*) an ordered DAG corresponding to G, where the *solid* and *dashed edges* respectively correspond to the first and second child of each node

description of S that is based on a deterministic context-free grammar that generates a language consisting only of S. We will call such a context-free grammar, a grammar that represents S.

Generally, grammar-based compression can be divided into two phases [8], the *grammar transform* phase, where a context-free grammar G that represents string S is computed, and the *grammar encoding* phase, where an encoding for G is computed. Kieffer and Yang [8] showed that if a grammar transform is *irreducible*, namely, if the resulting grammar that represents S satisfies the following three conditions: (1) distinct variables derive different strings, (2) every variable other than the start symbol is used more than once (rule utility), and (3) all pairs of symbols have at most one nonoverlapping occurrence in the right-hand side of production rules (di-gram uniqueness); then, the grammar-based code using a zero order arithmetic code for encoding the grammar is universal.

Grammar-based compression algorithms differ mostly by how they perform the grammar transform, which can be stated as the following problem.

Problem 1 (Smallest Grammar Problem) Given an input string S of length N, output the smallest context-free grammar that represents S.

Here, the *size* of the grammar is defined as the total length of the right-hand side of the production rules in the grammar. Often, grammars are considered to be in the Chomsky normal form, in which case the grammar is called a *straight line program* (SLP) [7], i.e., the right-hand side of each production rule is either a terminal character or a pair of variables. Note that any grammar of size n can be converted into an SLP of size $O(n)$. Figure 1 shows an example of an SLP that represents string aabaaaaaba. A grammar representing a string can be considered as an ordered directed acyclic graph. Another important feature is that grammars allow for exponential compression, that is, the size of a grammar that represents a string of length N can be as small as $O(\log N)$.

Grammar-based compression is known to be especially suitable for compressing *highly repetitive strings*, for example, multiple whole genomes, where, although each individual string may not be easily compressible, the ensemble of strings is very compressible since each string is very similar to each other. Also, due to its ease of manipulation, grammar-based representation of strings is a frequently used model for *compressed string processing*, where the aim is to efficiently process compressed strings without explicit decompression. Such an approach allows for theoretical and practical speedups compared to a naive decompress-then-process approach.

Key Results

Hardness

The smallest grammar problem is known to be NP-hard [21]. The approximation ratio of a grammar-based algorithm A is defined as $\max_{S \in \Sigma^*} \frac{|G_S^{\mathrm{A}}|}{|G_S^{\mathrm{opt}}|}$, where $|G_S^{\mathrm{A}}|$ is the size of the grammar that represents string S produced by A and $|G_S^{\mathrm{opt}}|$ is the size of the smallest grammar that represents string S. Charikar et al. [3] showed that there is no polynomial-time algorithm for the smallest grammar problem with approximation ratio less than $\frac{8,569}{8,568}$, unless $P = NP$. Furthermore, they show that for a given set $\{k_1, \ldots, k_m\}$ of positive integers, the smallest grammar for string $\mathtt{a}^{k_1}\mathtt{b}\mathtt{a}^{k_2}\mathtt{b}\cdots\mathtt{b}\mathtt{a}^{k_m}$ is within a constant factor of the smallest number of multiplications required to compute $x^{k_1}, \ldots, x^{k_m}$, given a real number x. This is a well-studied problem known as the addition chain problem, whose best-known approximation algorithm has an approximation ratio of $O(\frac{\log N}{\log \log N})$ [23]. Thus, achieving $o(\frac{\log N}{\log \log N})$ approximation for the smallest grammar problem may be difficult.

Algorithms for Finding Small Grammars

Heuristics

Below, we give brief descriptions of several grammar-based compression algorithms based on simple greedy heuristics for which approximation ratios have been analyzed [3] (see Table 1).

- LZ78 [24] can be considered as constructing a grammar. Recall that each LZ78 factor of length at least two consists of a previous factor and a letter and can be expressed as a production rule of a grammar.
- SEQUITUR [15] processes the string in an online manner and adds a new character of the string to the right-hand side of the production rule of the start symbol, which is initially empty. For each new character, the algorithm updates the grammar, adding or removing production rules and replacing corresponding symbols in the grammar so that the di-gram uniqueness and rule utility properties are satisfied. The algorithm can be implemented to run in expected linear time. The grammar produced by SEQUITUR is not necessarily irreducible, and thus a revised version called SEQUENTIAL was proposed in [8].
- RE-PAIR [11] greedily and recursively replaces the most frequent di-gram in the string with a new symbol until no di-gram occurs more than once. Each such replacement corresponds to a new production rule in the final grammar. The algorithm can be implemented to run in linear time.
- LONGEST MATCH [8] greedily and recursively replaces the longest substring that has more than one nonoverlapping occurrence. The algorithm can be implemented to run in linear time by carefully maintaining a structure based on the suffix tree, through the course of the algorithm [10, 14].
- GREEDY [1] (originally called OFF-LINE, but coined in [3]) greedily and recursively replaces substrings that give the highest compression (with several variations in its definition). The algorithm can be implemented to run in $O(N \log N)$ time for each production rule, utilizing a data structure called minimal augmented suffix trees, which augments the suffix tree in order to consider the total number of nonoverlapping occurrences of a given substring.
- BISECTION [9] recursively partitions the string S into strings L and R of lengths 2^i and $N - 2^i$, where $i = \lceil \log N \rceil - 1$, each time forming a production rule $X_S \rightarrow X_L X_R$. A new production rule is created only for each distinct substring, and the rule is shared for identical substrings. The algorithm can be viewed as fixing the shape of the derivation tree and then computing the smallest grammar whose derivation tree is of the given shape.

Approximation Algorithms

Rytter [16] and Charikar et al. [3] independently and almost simultaneously developed linear time

Grammar Compression, Table 1 Known upper and lower bounds on approximation ratios for the simple heuristic algorithms (Taken from [3] with corrections)

Algorithm	Upper bound	Lower bound
LZ78 [24]	$O((N/\log N)^{2/3})$	$\Omega(N^{2/3}/\log N)$
RE-PAIR [11]		$\Omega(\sqrt{\log N})$
LONGEST MATCH [8]		$\Omega(\log\log N)$
GREEDY [1]		$(5\log 3)/(3\log 5) > 1.137\ldots$
SEQUENTIAL [8]	$O((N/\log N)^{3/4})$	$\Omega(N^{1/3})$
BISECTION [9]	$O((N/\log N)^{1/2})$	$\Omega(N^{1/2}/\log N)$

Grammar Compression, Table 2 Approximation algorithms for the smallest grammar problem. N is the size of the input string, and n is the size of the output grammar

Algorithm	Approximation ratio	Working space	Running time
Charikar et al. [3]	$O(\log(N/G_S^{\text{opt}}))$	$O(N)$	$O(N)$
Rytter [16]			
LEVELWISE-REPAIR [18]			
Jeż [5]			
Jeż [6]			
LCA [19]	$O((\log N)\log G_S^{\text{opt}})$	$O(n)$	$O(N)$ expected
LCA* [20]	$O((\log^* N)\log N)$	$O(n)$	$O(N\log^* N)$ expected
OLCA [12]	$O(\log^2 N)$	$O(n)$	$O(N)$ expected
FOLCA [13]	$O(\log^2 N)$	$2n\log n(1+o(1))+2n$ bits	$O(N\log N)$

algorithms which achieve the currently best approximation ratio of $O(\log(N/G_S^{\text{opt}}))$, essentially relying on the same two key ideas: the LZ77 factorization and balanced binary grammars. Below, we briefly describe the approach by Rytter to obtain an $O(\log N)$ approximation algorithm.

The string is processed from left to right, and the LZ77 factorization of the string helps to reuse, as much as possible, the grammar of previously occurring substrings. For string S, let $S = f_1 \ldots f_z$ be the LZ77 factorization of S. The algorithm sequentially processes each LZ factor f_i, maintaining a grammar G_i for $f_1 \ldots f_i$. Recall that by definition, each factor f_i of length at least 2 occurs in $f_1 \ldots f_{i-1}$. Therefore, there exists a sequence of $O(h_{i-1})$ variables of grammar G_{i-1} whose concatenation represents f_i, where h_{i-1} is the height of the derivation tree of G_{i-1}. Using this sequence of variables, a grammar for f_i is constructed, which is then subsequently appended to G_{i-1} to finally construct G_i.

A balanced binary grammar is a grammar in which the shape of the derivation tree resembles a balanced binary tree. Rytter proposed AVL (height balanced) grammars, where the height of sibling sub-trees differ by at most one. By restricting the grammar to AVL grammars, the height of the grammar is bounded by $O(\log N)$, and the above operations can be performed in $O(\log N)$ time for each LZ77 factor, by adding $O(\log N)$ new variables and using techniques resembling those of binary balanced search trees for re-balancing the tree. The resulting time complexity as well as the size of the grammar is $O(z\log N)$.

Finally, an important observation is that the size of the LZ77 factorization of a string S is a lower bound on the size of any grammar G that represents S.

Theorem 1 ([3, 16]) *For string S, let $S = f_1 \ldots f_z$ be the LZ77 factorization of S. Then, for any grammar G that represents S, $z \le |G|$.*

Thus, the total size of the grammar is $O(G_S^{\text{opt}} \log N)$, achieving an $O(\log N)$ approximation ratio. Instead of AVL grammars, Charikar et al. use α balanced (length balanced) grammars, where the ratio between the lengths of sibling sub-trees is between $\frac{\alpha}{1-\alpha}$ and $\frac{1-\alpha}{\alpha}$ for some constant $0 < \alpha \leq \frac{1}{2}$, but the remaining arguments are similar.

Several other linear time algorithms that achieve $O(\log(N/G_S^{\text{opt}}))$ approximation have been proposed [5, 6, 18]. These algorithms resemble RE-PAIR in that they basically replace di-grams in the string with a new symbol in a bottom-up fashion but with specific mechanisms to choose the di-grams so that a good approximation ratio is achieved.

LCA and its variants [12, 13, 19, 20] are approximation algorithms shown to be among the most scalable and practical. The approximation ratios are slightly weaker, but the algorithm can be made to run in an online manner and to use small space (see Table 2). Although seemingly proposed independently, the core idea of LCA is essentially the same as LCP [17] which constructs a grammar based on a technique called *locally consistent parsing*. The parsing is a partitioning of the string that can be computed using only local characteristics and guarantees that for any two occurrences of a given substring, the partitioning in the substring will be almost identical with exceptions in a sufficiently short prefix and suffix of the substring. This allows the production rules of the grammar to be more or less the same for repeated substrings, thus bounding the approximation ratio.

Decompression

The string that a grammar represents can be recovered in linear time by a simple depth-first left-to-right traversal on the grammar. Given an SLP G of size n that represents a string S of length N, G can be preprocessed in $O(n)$ time and space so that each variable holds the length of the string it derives. Using this information, it is possible to access $S[i]$ for any $1 \leq i \leq N$ in $O(h)$ time, where h is the height of the SLP, by simply traversing down the production rules starting from the start symbol until reaching a terminal character corresponding to $S[i]$. Balanced SLPs have height $O(\log N)$ and, therefore, allow access to any position of S in $O(\log N)$ time. For any grammar G, G can be preprocessed in $O(n)$ time and space, so that an arbitrary substring of length l of S can be obtained in $O(l + \log N)$ time [2]. Also, G can be preprocessed in $O(n)$ time and space so that the prefix or suffix of any length l for any variable in G can be obtained in $O(l)$ time [4]. On the other hand, it has been shown that using any data structure of size polynomial in n, the time for retrieving a character at an arbitrary position is at least $(\log N)^{1-\epsilon}$ for any constant $\epsilon > 0$ [22].

URLs to Code and Data Sets

Publicly available implementations of SEQUITUR:

- http://www.sequitur.info

Publicly available implementations of RE-PAIR:

- http://www.dcc.uchile.cl/~gnavarro/software/repair.tgz
- http://www.cbrc.jp/~rwan/en/restore.html, and
- https://code.google.com/p/re-pair/

Publicly available implementations of GREEDY (OFF-LINE):

- http://www.cs.ucr.edu/~stelo/Offline/.

Publicly available implementations of LCA variants:

- https://code.google.com/p/lcacomp/
- https://github.com/tb-yasu/olca-plus-plus

Cross-References

- ▸ Arithmetic Coding for Data Compression
- ▸ Lempel-Ziv Compression
- ▸ Pattern Matching on Compressed Text

Recommended Reading

1. Apostolico A, Lonardi S (2000) Off-line compression by greedy textual substitution. Proc IEEE 88(11):1733–1744
2. Bille P, Landau GM, Raman R, Sadakane K, Satti SR, Weimann O (2011) Random access to grammar-compressed strings. In: Proceedings of the SODA'11, San Francisco, pp 373–389
3. Charikar M, Lehman E, Liu D, Panigrahy R, Prabhakaran M, Sahai A, Shelat A (2005) The smallest grammar problem. IEEE Trans. Inf. Theory 51(7):2554–2576
4. Gąsieniec L, Kolpakov R, Potapov I, Sant P (2005) Real-time traversal in grammar-based compressed files. In: Proceedings of the DCC'05, Snowbird, p 458
5. Jeż A (2013) Approximation of grammar-based compression via recompression. In: Proceedings of the CPM'13, Bad Herrenalb, pp 165–176
6. Jeż A (2014) A *really* simple approximation of smallest grammar. In: Proceedings of the CPM'14, Moscow, pp 182–191
7. Karpinski M, Rytter W, Shinohara A (1997) An efficient pattern-matching algorithm for strings with short descriptions. Nord J Comput 4:172–186
8. Kieffer JC, Yang EH (2000) Grammar-based codes: a new class of universal lossless source codes. IEEE Trans Inf Theory 46(3):737–754
9. Kieffer J, Yang E, Nelson G, Cosman P (2000) Universal lossless compression via multilevel pattern matching. IEEE Trans Inf Theory 46(4):1227–1245
10. Lanctôt JK, Li M, Yang E (2000) Estimating DNA sequence entropy. In: Proceedings of the SODA'00, San Francisco, pp 409–418
11. Larsson NJ, Moffat A (2000) Off-line dictionary-based compression. Proc IEEE 88(11):1722–1732
12. Maruyama S, Sakamoto H, Takeda M (2012) An online algorithm for lightweight grammar-based compression. Algorithms 5(2):214–235
13. Maruyama S, Tabei Y, Sakamoto H, Sadakane K (2013) Fully-online grammar compression. In: Proceedings of the SPIRE'13, Jerusalem, pp 218–229
14. Nakamura R, Inenaga S, Bannai H, Funamoto T, Takeda M, Shinohara A (2009) Linear-time text compression by longest-first substitution. Algorithms 2(4):1429–1448
15. Nevill-Manning CG, Witten IH (1997) Identifying hierarchical structure in sequences: a linear-time algorithm. J Artif Intell Res 7(1):67–82
16. Rytter W (2003) Application of Lempel-Ziv factorization to the approximation of grammar-based compression. Theor Comput Sci 302(1–3): 211–222
17. Sahinalp SC, Vishkin U (1995) Data compression using locally consistent parsing. Technical report, UMIACS Technical Report
18. Sakamoto H (2005) A fully linear-time approximation algorithm for grammar-based compression. J Discret Algorithms 3(2–4):416–430
19. Sakamoto H, Kida T, Shimozono S (2004) A space-saving linear-time algorithm for grammar-based compression. In: Proceedings of the SPIRE'04, Padova, pp 218–229
20. Sakamoto H, Maruyama S, Kida T, Shimozono S (2009) A space-saving approximation algorithm for grammar-based compression. IEICE Trans 92-D(2):158–165
21. Storer JA (1977) NP-completeness results concerning data compression. Technical report 234, Department of Electrical Engineering and Computer Science, Princeton University
22. Verbin E, Yu W (2013) Data structure lower bounds on random access to grammar-compressed strings. In: Proceedings of the CPM'13, Bad Herrenalb, pp 247–258
23. Yao ACC (1976) On the evaluation of powers. SIAM J Comput 5(1):100–1–03
24. Ziv J, Lempel A (1978) Compression of individual sequences via variable-length coding. IEEE Trans Inf Theory 24(5):530–536

Graph Bandwidth

James R. Lee
Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA

Keywords

Approximation algorithms; Graph bandwidth; Metric embeddings

Years and Authors of Summarized Original Work

1998; Feige
2000; Feige

Problem Definition

The *graph bandwidth problem* concerns producing a linear ordering of the vertices of a graph $G = (V, E)$ so as to minimize the maximum

"stretch" of any edge in the ordering. Formally, let $n = |V|$, and consider any one-to-one mapping $\pi : V \to \{1, 2, \ldots, n\}$. The *bandwidth* of this ordering is $\mathsf{bw}_\pi(G) = \max_{\{u,v\}\in E} |\pi(u) - \pi(v)|$. The *bandwidth of G* is given by the bandwidth of the best possible ordering: $\mathsf{bw}(G) = \min_\pi \mathsf{bw}_\pi(G)$.

The original motivation for this problem lies in the preprocessing of sparse symmetric square matrices. Let A be such an $n \times n$ matrix, and consider the problem of finding a permutation matrix P such that the non-zero entries of $P^{\mathrm{T}}AP$ all lie in as narrow a band as possible about the diagonal. This problem is equivalent to minimizing the bandwidth of the graph G whose vertex set is $\{1, 2, \ldots, n\}$ and which has an edge $\{u, v\}$ precisely when $A_{u,v} \neq 0$.

In lieu of this fact, one tries to efficiently compute a linear ordering π for which $\mathsf{bw}_\pi(G) \leq A \cdot \mathsf{bw}(G)$, with the *approximation factor* A is as small as possible. There is even evidence that achieving any value $A = O(1)$ is NP-hard [18]. Much of the difficulty of the bandwidth problem is due to the objective function being a maximum over all edges of the graph. This makes divide-and-conquer approaches ineffective for graph bandwidth, whereas they often succeed for related problems like Minimum Linear Arrangement [6] (here the objective is to minimize $\sum_{\{u,v\}\in E} |\pi(u) - \pi(v)|$). Instead, a more global algorithm is required. To this end, a good lower bound on the value of $\mathsf{bw}(G)$ has to be initially discussed.

The Local Density

For any pair of vertices $u, v \in V$, let $d(u, v)$ to be the shortest path distance between u and v in the graph G. Then, define $B(v, r) = \{u \in V : d(u, v) \leq r\}$ as the *ball of radius r* about a vertex $v \in V$. Finally, the *local density of G* is defined by $D(G) = \max_{v\in V, r\geq 1} |B(v, r)|/(2r)$. It is not difficult to see that $\mathsf{bw}(G) \geq D(G)$. Although it was conjectured that an upper bound of the form $\mathsf{bw}(G) \leq \mathrm{poly}(\log n) \cdot D(G)$ holds, it was not proven until the seminal work of Feige [7].

Key Results

Feige proved the following.

Theorem 1 *There is an efficient algorithm that, given a graph $G = (V, E)$ as input, produces a linear ordering $\pi: V \to \{1, 2, \ldots, n\}$ for which $\mathsf{bw}_\pi(G) \leq O\left((\log n)^3 \sqrt{\log n \log\log n}\right) \cdot D(G)$. In particular, this provides a $\mathrm{poly}(\log n)$-approximation algorithm for the bandwidth problem in general graphs.*

Feige's algorithmic framework can be described quite simply as follows.

1. Compute a representation $f: V \to \mathbb{R}^n$ of G in Euclidean space.
2. Let $u_1, u_2, \ldots, u_n$ be independent $N(0, 1)$. ($N(0; 1)$ denotes a standard normal random variable with mean 0 and variance 1.) random variables, and for each vertex $v \in V$, compute $h(v) = \sum_{i=1}^n u_i f_i(v)$, where $f_i(v)$ is the ith coordinate of the vector $f(v)$.
3. Sort the vertices by the value $h(v)$, breaking ties arbitrarily, and output the induced linear ordering.

An equivalent characterization of steps (2) and (3) is to choose a uniformly random vector $\mathbf{a} \in S^{n-1}$ from the $(n-1)$-dimensional sphere $S^{n-1} \subseteq \mathbb{R}^n$ and output the linear ordering induced by the values $h(v) = \langle \mathbf{a}, f(v) \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the usual inner product on $\mathbb{R}^n$. In other words, the algorithm first computes a map $f: V \to \mathbb{R}^n$, projects the images of the vertices onto a randomly oriented line, and then outputs the induced ordering; step (2) is the standard way that such a random projection is implemented.

Volume-Respecting Embeddings

The only step left unspecified is (1); the function f has to somehow preserve the structure of the graph G in order for the algorithm to output a low-bandwidth ordering. The inspiration for the existence of such an f comes from the field of *low-distortion metric embeddings* (see,

e.g., [2, 14]). Feige introduced a generalization of low-distortion embeddings to mappings called *volume respecting embeddings*. Roughly, the map f should be non-expansive, in the sense that $\|f(u) - f(v)\| \leq 1$ for every edge $\{u, v\} \in E$, and should satisfy the following property: For any set of k vertices $v_1, \ldots, v_k$, the $(k-1)$-dimensional volume of the convex hull of the points $f(v_1), \ldots, f(v_k)$ should be as large as possible. The proper value of k is chosen to optimize the performance of the algorithm. Refer to [7, 10, 11] for precise definitions on volume-respecting embeddings, and a detailed discussion of their construction. Feige showed that a modification of Bourgain's embedding [2] yields a mapping $f: V \to \mathbb{R}^n$ which is good enough to obtain the results of Theorem 1.

The requirement $\|f(u) - f(v)\| \leq 1$ for every edge $\{u, v\}$ is natural since $f(u)$ and $f(v)$ need to have similar projections onto the random direction $\mathbf{a}$; intuitively, this suggests that u and v will not be mapped too far apart in the induced linear ordering. But even if $|h(u) - h(v)|$ is small, it may be that many vertices project between $h(u)$ and $h(v)$, causing u and v to incur a large stretch. To prevent this, the images of the vertices should be sufficiently "spread out," which corresponds to the volume requirement on the convex hull of the images.

Applications

As was mentioned previously, the graph bandwidth problem has applications to preprocessing sparse symmetric matrices. Minimizing the bandwidth of matrices helps in improving the efficiency of certain linear algebraic algorithms like Gaussian elimination; see [3, 8, 17]. Follow-up work has shown that Feige's techniques can be applied to VLSI layout problems [19].

Open Problems

First, state the *bandwidth conjecture* (see, e.g., [13]).

Conjecture: For any n-node graph $G = (V, E)$, one has $\mathsf{bw}(G) = O(\log n) \cdot D(G)$.

The conjecture is interesting and unresolved even in the special case when G is a tree (see [9] for the best results for trees). The best-known bound in the general case follows from [7, 10], and is of the form $\mathsf{bw}(G) = O(\log n)^{3.5} \cdot D(G)$. It is known that the conjectured upper bound is best possible, even for trees [4]. One suspects that these combinatorial studies will lead to improved approximation algorithms.

However, the best approximation algorithms, which achieve ratio $O((\log n)^3 (\log \log n)^{1/4})$, are not based on the local density bound. Instead, they are a hybrid of a semi-definite programming approach of [1, 5] with the arguments of Feige, and the volume-respecting embeddings constructed in [12, 16]. Determining the approximability of graph bandwidth is an outstanding open problem, and likely requires improving both the upper and lower bounds.

Recommended Reading

1. Blum A, Konjevod G, Ravi R, Vempala S (2000) Semi-definite relaxations for minimum bandwidth and other vertex-ordering problems. Theor Comput Sci 235(1):25–42, Selected papers in honor of Manuel Blum (Hong Kong, 1998)
2. Bourgain J (1985) On Lipschitz embedding of finite-metric spaces in Hilbert space. Israel J Math 52(1–2):46–52
3. Chinn PZ, Chvátalová J, Dewdney AK, Gibbs NE (1982) The bandwidth problem for graphs and matrices – a survey. J Graph Theory 6(3):223–254
4. Chung FRK, Seymour PD (1989) Graphs withsmall bandwidth and cutwidth. Discret Math 75(1–3):113–119, Graph theory and combinatorics, Cambridge (1988)
5. Dunagan J, Vempala S (2001) On Euclidean embeddings and bandwidth minimization. In: Randomization, approximation, and combinatorial optimization. Springer, pp 229–240
6. Even G, Naor J, Rao S, Schieber B (2000) Divide-and-conquer approximation algorithms via spreading metrics. J ACM 47(4):585–616
7. Feige U (2000) Approximating the bandwidth via volume respecting embeddings. J Comput Syst Sci 60(3):510–539
8. George A, Liu JWH (1981) Computer solution of large sparse positive definite systems. Prentice-hall series in computational mathematics. Prentice-Hall, Englewood Cliffs
9. Gupta A (2001) Improved bandwidth approximation for trees and chordal graphs. J Algorithms 40(1):24–36

10. Krauthgamer R, Lee JR, Mendel M, Naor A (2005) Measured descent: a new embedding method for finite metrics. Geom Funct Anal 15(4):839–858
11. Krauthgamer R, Linial N, Magen A (2004) Metric embeddings – beyond one-dimensional distortion. Discret Comput Geom 31(3):339–356
12. Lee JR (2006) Volume distortion for subsets of Euclidean spaces. In: Proceedings of the 22nd annual symposium on computational geometry. ACM, Sedona, pp 207–216
13. Linial N (2002) Finite metric-spaces – combinatorics, geometry and algorithms. In: Proceedings of the international congress of mathematicians, vol. III, Beijing, 2002. Higher Ed. Press, Beijing, pp 573–586
14. Linial N, London E, Rabinovich Y (1995) The geometry of graphs and some of its algorithmic applications. Combinatorica 15(2):215–245
15. Papadimitriou CH (1976) The NP-completeness of the bandwidth minimization problem. Computing 16(3):263–270
16. Rao S (1999) Small distortion and volume preserving embeddings for planar and Euclidean metrics. In: Proceedings of the 15th annual symposium on computational geometry. ACM, New York, pp 300–306
17. Strang G (1980) Linear algebra and its applications, 2nd edn. Academic [Harcourt Brace Jovanovich Publishers], New York
18. Unger W (1998) The complexity of the approximation of the bandwidth problem. In: 39th annual symposium on foundations of computer science. IEEE, 8–11 Oct 1998, pp 82–91
19. Vempala S (1998) Random projection: a new approach to VLSI layout. In: 39th annual symposium on foundations of computer science. IEEE, 8–11 Oct 1998, pp 389–398

Graph Coloring

Michael Langberg [1,3] and Chandra Chekuri[2,3]
[1]Department of Electrical Engineering, The State University of New York, Buffalo, NY, USA
[2]Department of Computer Science, University of Illinois, Urbana-Champaign, Urbana, IL, USA
[3]Department of Mathematics and Computer Science, The Open University of Israel, Raanana, Israel

Keywords

Approximation algorithms; Clique cover; Graph coloring; Semidefinite

Years and Authors of Summarized Original Work

1994, 1998; Karger, Motwani, Sudan

Problem Definition

An independent set in an undirected graph $G = (V, E)$ is a set of vertices that induce a subgraph which does not contain any edges. The size of the maximum independent set in G is denoted by $\alpha(G)$. For an integer k, a k-coloring of G is a function $\sigma : V \to [1 \dots k]$ which assigns colors to the vertices of G. A valid k-coloring of G is a coloring in which each color class is an independent set. The chromatic number $\chi(G)$ of G is the smallest k for which there exists a valid k-coloring of G. Finding $\chi(G)$ is a fundamental NP-hard problem. Hence, when limited to polynomial time algorithms, one turns to the question of estimating the value of $\chi(G)$ or to the closely related problem of *approximate coloring*.

Problem 1 (Approximate coloring)
INPUT: Undirected graph $G = (V, E)$.
OUTPUT: A valid coloring of G with $r \cdot \chi(G)$ colors, for some approximation ratio $r \geq 1$.
OBJECTIVE: Minimize r.

Let G be a graph of size n. The approximate coloring of G can be solved efficiently within an approximation ratio of $r = O\left(\frac{n(\log\log n)^2}{\log^3 n}\right)$ [12]. This holds also for the approximation of $\alpha(G)$ [8]. These results may seem rather weak; however, it is NP-hard to approximate $\alpha(G)$ and $\chi(G)$ within a ratio of $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$ [9, 14, 23]. Under stronger complexity assumptions, there is some constant $0 < \delta < 1$ such that neither problem can be approximated within a ratio of $n/2^{\log^\delta n}$ [19,23]. This entry will concentrate on the problem of coloring graphs G for which $\chi(G)$ is *small*. As will be seen, in this case the approximation ratio achievable significantly improves.

Vector Coloring of Graphs

The algorithms achieving the best ratios for approximate coloring when $\chi(G)$ is small are all based on the idea of *vector coloring*, introduced by Karger, Motwani, and Sudan [17]. (Vector coloring as presented in [17] is closely related to the Lovász θ function [21]. This connection will be discussed shortly.)

Definition 1 A vector k-coloring of a graph is an assignment of unit vectors to its vertices, such that for every edge, the inner product of the vectors assigned to its endpoints is at most (in the sense that it can only be more negative) $-1/(k-1)$.

The *vector chromatic number* $\overrightarrow{\chi}(G)$ of G is the smallest k for which there exists a vector k-coloring of G. The vector chromatic number can be formulated as follows:

$$\begin{aligned} \overrightarrow{\chi}(G)\ \text{Minimize}\quad & k \\ \text{subject to:}\quad & \langle v_i, v_j\rangle \le -\tfrac{1}{k-1} \quad \forall (i,j)\in E \\ & \langle v_i, v_i\rangle = 1 \qquad\quad \forall i \in V \end{aligned}$$

Here, assume that $V = [1,\ldots,n]$ and that the vectors $\{v_i\}_{i=1}^n$ are in R^n. Every k-colorable graph is also vector k-colorable. This can be seen by identifying each color class with one vertex of a perfect $(k-1)$-dimensional simplex centered at the origin. Moreover, unlike the chromatic number, a vector k-coloring (when it exists) can be found in polynomial time using semidefinite programming (up to an arbitrarily small error in the inner products).

Claim 1 (Complexity of vector coloring [17]) Let $\varepsilon > 0$. If a graph G has a vector k-coloring, then a vector $(k+\varepsilon)$-coloring of the graph can be constructed in time polynomial in n and $\log(1/\varepsilon)$.

One can strengthen Definition 1 to obtain a different notion of vector coloring and the vector chromatic number:

$$\begin{aligned} \overrightarrow{\chi}_2(G)\ \text{Minimize}\quad & k \\ \text{subject to:}\quad & \langle v_i, v_j\rangle = -\tfrac{1}{k-1} \quad \forall (i,j)\in E \\ & \langle v_i, v_i\rangle = 1 \qquad\quad \forall i \in V \end{aligned}$$

$$\begin{aligned} \overrightarrow{\chi}_3(G)\ \text{Minimize}\quad & k \\ \text{subject to:}\quad & \langle v_i, v_j\rangle = -\tfrac{1}{k-1} \quad \forall (i,j)\in E \\ & \langle v_i, v_j\rangle \ge -\tfrac{1}{k-1} \quad \forall i,j \in V \\ & \langle v_i, v_i\rangle = 1 \qquad\quad \forall i \in V \end{aligned}$$

The function $\overrightarrow{\chi}_2(G)$ is referred to as the *strict* vector chromatic number of G and is equal to the Lovász θ function on $\bar{G}$ [17,21], where $\bar{G}$ is the *complement* graph of G. The function $\overrightarrow{\chi}_3(G)$ is referred to as the *strong* vector chromatic number. An analog to Claim 1 holds for both $\overrightarrow{\chi}_2(G)$ and $\overrightarrow{\chi}_3(G)$. Let $\omega(G)$ denote the size of the maximum clique in G; it holds that $\omega(G) \le \overrightarrow{\chi}(G) \le \overrightarrow{\chi}_2(G) \le \overrightarrow{\chi}_3(G) \le \chi(G)$.

Key Results

In what follows, assume that G has n vertices and maximal degree Δ. The $\tilde{O}(\cdot)$ and $\tilde{\Omega}(\cdot)$ notation are used to suppress polylogarithmic factors. We now state the key result of Karger, Motwani, and Sudan [17]:

Theorem 1 ([17]) *If $\overrightarrow{\chi}(G) = k$, then G can be colored in polynomial time using $\min\{\tilde{O}(\Delta^{1-2/k}), \tilde{O}(n^{1-3/(k+1)})\}$ colors.*

As mentioned above, the use of vector coloring in the context of approximate coloring was initiated in [17]. Roughly speaking, once given a vector coloring of G, the heart of the algorithm in [17] finds a large independent set in G. In a nutshell, this independent set corresponds to a set of vectors in the vector coloring which are *close* to one another (and thus by definition cannot share an edge). Combining this with the ideas of Wigderson [22] mentioned below yields Theorem 1.

We proceed to describe related work. The first two theorems below appeared prior to the work of Karger, Motwani, and Sudan [17].

Theorem 2 ([22]) *If $\chi(G) = k$, then G can be colored in polynomial time using $O(k\,n^{1-1/(k-1)})$ colors.*

Theorem 3 ([1]) *If $\chi(G) = 3$, then G can be colored in polynomial time using $\tilde{O}(n^{3/8})$ colors. If $\chi(G) = k \geq 4$ then G can be colored in polynomial time using at most $\tilde{O}(n^{1-1/(k-3/2)})$ colors.*

Combining the techniques of [17] and [1], the following results were obtained for graphs G with $\chi(G) = 3, 4$ (these results were also extended for higher values of $\chi(G)$).

Theorem 4 ([2]) *If $\chi(G) = 3$, then G can be colored in polynomial time using $\tilde{O}(n^{3/14})$ colors.*

Theorem 5 ([13]) *If $\chi(G) = 4$, then G can be colored in polynomial time using $\tilde{O}(n^{7/19})$ colors.*

The currently best known result for coloring a 3-colorable graph is presented in [16]. The algorithm of [16] combines enhanced notions of vector coloring presented in [5] with the combinatorial coloring techniques of [15].

Theorem 6 ([16]) *If $\chi(G) = 3$, then G can be colored in polynomial time using $O(n^{0.19996})$ colors.*

To put the above theorems in perspective, it is NP-hard to color a 3-colorable graph G with 4 colors [11, 18] and a k-colorable graph (for sufficiently large k) with $k^{\frac{\log k}{25}}$ colors [19]. Under stronger complexity assumptions (related to the unique games conjecture [20]) for any constant k, it is hard to color a k-colorable graph with any constant number of colors [6]. The wide gap between these hardness results and the approximation ratios presented in this section has been a major initiative in the study of approximate coloring.

Finally, the limitations of vector coloring are addressed. Namely, are there graphs for which $\overrightarrow{\chi}(G)$ is a poor estimate of $\chi(G)$? One would expect the answer to be "yes" as estimating $\chi(G)$ beyond a factor of $n^{1-\varepsilon}$ is a hard problem. As will be stated below, this is indeed the case (even when $\overrightarrow{\chi}(G)$ is small). Some of the results that follow are stated in terms of the maximum independent set $\alpha(G)$ in G. As $\chi(G) \geq n/\alpha(G)$, these results imply a lower bound on $\chi(G)$. Theorem 7 (i) states that the original analysis of [17] is essentially tight. Theorem 7 (ii) presents bounds for the case of $\overrightarrow{\chi}(G) = 3$. Theorem 7 (iii) and Theorem 8 present graphs G in which there is an extremely large gap between $\chi(G)$ and the relaxations $\overrightarrow{\chi}(G)$ and $\overrightarrow{\chi}_2(G)$.

Theorem 7 ([10]) *(i) For every constant $\varepsilon > 0$ and constant $k > 2$, there are infinitely many graphs G with $\overrightarrow{\chi}(G) = k$ and $\alpha(G) \leq n/\Delta^{1-\frac{2}{k}-\varepsilon}$ (here $\Delta > n^{\delta}$ for some constant $\delta > 0$). (ii) There are infinitely many graphs G with $\overrightarrow{\chi}(G) = 3$ and $\alpha(G) \leq n^{0.843}$. (iii) For some constant c, there are infinitely many graphs G with $\overrightarrow{\chi}(G) = O(\frac{\log n}{\log\log n})$ and $\alpha(G) \leq \log^c n$.*

Theorem 8 ([7]) *For some constant c, there are infinitely many graphs G with $\overrightarrow{\chi}_2(G) \leq 2^{\sqrt{\log n}}$ and $\chi(G) \geq n/2^{c\sqrt{\log n}}$.*

Vector colorings, including the Lovász θ function and its variants, have been extensively studied in the context of approximation algorithms for problems other than Problem 1. These include approximating $\alpha(G)$, approximating the minimum vertex cover problem, and combinatorial optimization in the context of random graphs.

Applications

Besides its theoretical significance, graph coloring has several concrete applications (see, e.g., [3,4]).

Open Problems

By far the major open problem in the context of approximate coloring addresses the wide gap between what is known to be hard and what can be obtained in polynomial time. The case of constant $\chi(G)$ is especially intriguing, as the best known upper bounds (on the approximation ratio) are polynomial while the lower bounds are of constant nature. Regarding the vector coloring paradigm, a majority of the results stated in section "Key Results" use the weakest from of vector coloring $\overrightarrow{\chi}(G)$ in their proof, while stronger relaxations may also be considered. It

would be very interesting to improve upon the algorithmic results stated above using stronger relaxations, as would a matching analysis of the limitations of these relaxations.

Recommended Reading

1. Blum A (1994) New approximations for graph coloring. J ACM 41(3):470–516
2. Blum A, Karger D (1997) An $\tilde{O}(n^{3/14})$-coloring for 3-colorable graphs. Inf Process Lett 61(6):49–53
3. Chaitin GJ (1982) Register allocation & spilling via graph coloring. In: Proceedings of the 1982 SIGPLAN symposium on compiler construction, pp 98–105
4. Chaitin GJ, Auslander MA, Chandra AK, Cocke J, Hopkins ME, Markstein PW (1981) Register allocation via coloring. Comput Lang 6:47–57
5. Chlamtac E (2007) Approximation algorithms using hierarchies of semidefinite programming relaxations. In: Proceedings of the 48th annual IEEE symposium on foundations of computer science, pp 691–701
6. Dinur I, Mossel E, Regev O (2009) Conditional hardness for approximate coloring. SIAM J Comput 39(3):843–873
7. Feige U (1997) Randomized graph products, chromatic numbers, and the Lovász theta function. Combinatorica 17(1):79–90
8. Feige U (2004) Approximating maximum clique by removing subgraphs. SIAM J Discret Math 18(2):219–225
9. Feige U, Kilian J (1998) Zero knowledge and the chromatic number. J Comput Syst Sci 57:187–199
10. Feige U, Langberg M, Schechtman G (2004) Graphs with tiny vector chromatic numbers and huge chromatic numbers. SIAM J Comput 33(6):1338–1368
11. Guruswami V, Khanna S (2000) On the hardness of 4-coloring a 3-colorable graph. In: Proceedings of the 15th annual IEEE conference on computational complexity, pp 188–197
12. Halldorsson M (1993) A still better performance guarantee for approximate graph coloring. Inf Process Lett 45:19–23
13. Halperin E, Nathaniel R, Zwick U (2002) Coloring k-colorable graphs using smaller palettes. J Algorithms 45:72–90
14. Håstad J (1999) Clique is hard to approximate within $n^{1-\varepsilon}$. Acta Math 182(1):105–142
15. Kawarabayashi K, Thorup M (2012) Combinatorial coloring of 3-Colorable graphs. In: Proceedings of the 53rd annual IEEE symposium on foundations of computer science, pp 68–75
16. Kawarabayashi K, Thorup M (2014) Coloring 3-colorable graphs with $o(n^{1/5})$ colors. In: Proceedings of the 31st international symposium on theoretical aspects of computer science, pp 458–469
17. Karger D, Motwani R, Sudan M (1998) Approximate graph coloring by semidefinite programming. J ACM 45(2):246–265
18. Khanna S, Linial N, Safra S (2000) On the hardness of approximating the chromatic number. Combinatorica 20:393–415
19. Khot S (2001) Improved inapproximability results for max clique, chromatic number and approximate graph coloring. In: Proceedings of the 42nd annual IEEE symposium on foundations of computer science, pp 600–609
20. Khot S (2002) On the power of unique 2-prover 1-round games. In: Proceedings of the 34th annual ACM symposium on theory of computing, pp 767–775
21. Lovász L (1979) On the Shannon capacity of a graph. IEEE Trans Inf Theory 25:2–13
22. Wigderson A (1983) Improving the performance guarantee for approximate graph coloring. J ACM 30(4):729–735
23. Zuckerman D (2006) Linear degree extractors and the inapproximability of max clique and chromatic number. In: Proceedings of the 38th annual ACM symposium on theory of computing, pp 681–690

Graph Connectivity

Samir Khuller[1] and Balaji Raghavachari[2]
[1]Computer Science Department, University of Maryland, College Park, MD, USA
[2]Computer Science Department, The University of Texas at Dallas, Richardson, TX, USA

Keywords

Highly connected subgraphs; Sparse certificates

Years and Authors of Summarized Original Work

1994; Khuller, Vishkin

Problem Definition

An undirected graph is said to be k-connected (specifically, k-vertex-connected) if the removal of any set of $k-1$ or fewer vertices (with their incident edges) does not disconnect G. Analogously, it is k-edge-connected if the removal of any set of $k-1$ edges does not disconnect

G. Menger's theorem states that a k-vertex-connected graph has at least k openly vertex-disjoint paths connecting every pair of vertices. For k-edge-connected graphs there are k edge-disjoint paths connecting every pair of vertices. The connectivity of a graph is the largest value of k for which it is k-connected. Finding the connectivity of a graph, and finding k disjoint paths between a given pair of vertices can be found using algorithms for maximum flow. An edge is said to be *critical* in a k-connected graph if upon its removal the graph is no longer k-connected.

The problem of finding a minimum-cardinality k-vertex-connected (k-edge-connected) subgraph that spans all vertices of a given graph is called k-VCSS (k-ECSS) and is known to be nondeterministic polynomial-time hard for $k \geq 2$. We review some results in finding approximately minimum solutions to k-VCSS and k-ECSS. We focus primarily on simple graphs. A simple approximation algorithm is one that considers the edges in some order and removes edges that are not critical. It thus outputs a k-connected subgraph in which all edges are critical and it can be shown that it is a 2-approximation algorithm (that outputs a solution with at most kn edges in an n-vertex graph, and since each vertex has to have degree at least k, we can claim that $kn/2$ edges are necessary).

Approximation algorithms that do better than the simple algorithm mentioned above can be classified into two categories: depth first search (DFS) based, and matching based.

Key Results

Lower Bounds for k-Connected Spanning Subgraphs

Each node of a k-connected graph has at least k edges incident to it. Therefore, the sum of the degrees of all its nodes is at least kn, where n is the number of its nodes. Since each edge is counted twice in this degree-sum, the cardinality of its edges is at least $kn/2$. This is called the *degree lower bound*. Expanding on this idea yields a stronger lower bound on the cardinality of a k-connected spanning subgraph of a given graph. Let D_k be a subgraph in which the degree of each node is at least k. Unlike a k-connected subgraph, D_k has no connectivity constraints. The counting argument above shows that any D_k has at least $kn/2$ edges. A minimum cardinality D_k can be computed in polynomial time by reducing the problem to matching, and it is called the *matching lower bound*.

DFS-Based Approaches

The following natural algorithm finds a 3/2 approximation for 2-ECSS. Root the tree at some node r and run DFS. All edges of the graph are now either tree edges or back edges. Process the DFS tree in postorder. For each subtree, if the removal of the edge from its root to its parent separates the graph into two components, then add a farthest-back edge from this subtree, whose other end is closest to r. It can be shown that the number of back edges added by the algorithm is at most half the size of *Opt*.

This algorithm has been generalized to solve the 2-VCSS problem with the same approximation ratio, by adding carefully chosen back edges that allow the deletion of tree edges. Wherever it is unable to delete a tree edge, it adds a vertex to an independent set I. In the final analysis, the number of edges used is less than $n + |I|$. Since *Opt* is at least $\max(n, 2|I|)$, it obtains a 3/2-approximation ratio.

The algorithm can also be extended to the k-ECSS problem by repeating these ideas $k/2$ times, augmenting the connectivity by 2 in each round. It has been shown that this algorithm achieves a performance of about 1.61.

Matching-Based Approaches

Several approximation algorithms for k-ECSS and k-VCSS problems have used a minimum cardinality D_k as a starting solution, which is then augmented with additional edges to satisfy the connectivity constraints. This approach yields better ratios than the DFS-based approaches.

$1 + \frac{1}{k}$ Algorithm for k-VCSS
Find a minimum cardinality D_{k-1}. Add just enough additional edges to it to make the subgraph k-connected. In this step, it is ensured that the edges added are critical. It is known by a theorem of Mader that in a k-connected graph, a cycle of critical edges contains at least one node of degree k. Since the edges added by the algorithm in the second step are all critical, there can be no cycle induced by these edges because the degree of all the nodes on such a cycle would be at least $k + 1$. Therefore, at most $n - 1$ edges are added in this step. The number of edges added in the first step, in the minimum D_{k-1} is at most $Opt - n/2$. The total number of edges in the solution thus computed is at most $(1 + 1/k)$ times the number of edges in an optimal k-VCSS.

$1 + \frac{2}{k+1}$ Algorithm for k-ECSS
Mader's theorem about cycles induced by critical edges is valid only for vertex connectivity and not edge connectivity, Therefore, a different algorithm is proposed for k-ECSS in graphs that are k-edge-connected, but not k-connected. This algorithm finds a minimum cardinality D_k and augments it with a minimal set of edges to make the subgraph k-edge-connected. The number of edges added in the last step is at most $\frac{k}{k+1}(n-1)$. Since the number of edges added in the first step is at most Opt, the total number of edges is at most $(1 + \frac{2}{k+1})Opt$.

Better Algorithms for Small k
For $k \in \{2, 3\}$, better algorithms have been obtained by implementing the abovementioned algorithms carefully, deleting unnecessary edges, and by getting better lower bounds. For $k = 2$, a 4/3 approximation can be obtained by generating a path/cycle cover from a minimum cardinality D_2 and 2-connecting them one at a time to a "core" component. Small cycles/paths allow an edge to be deleted when they are 2-connected to the core, which allows a simple amortized analysis. This method also generalizes to the 3-ECSS problem, yielding a 4/3 ratio.

Hybrid approaches have been proposed which use the path/cycle cover to generate a specific DFS tree of the original graph and then 2-connect the tree, trying to delete edges wherever possible. The best ratios achieved using this approach are 5/4 for 2-ECSS, 9/7 for 2-VCSS, and 5/4 for 2-VCSS in 3-connected graphs.

Applications

Network design is one of the main application areas for this work. This involves the construction of low-cost highly connected networks.

Recommended Reading

For additional information on DFS, matchings and path/cycle covers, see [3]. Fast 2-approximation algorithms for k-ECSS and k-VCSS were studied by Nagamochi and Ibaraki [13]. DFS-based algorithms for 2-connectivity were introduced by Khuller and Vishkin [11]. They obtained 3/2 for 2-ECSS, 5/3 for 2-VCSS, and 2 for weighted k-ECSS. The ratio for 2-VCSS was improved to 3/2 by Garg et al. [6], 4/3 by Vempala and Vetta [14], and 9/7 by Gubbala and Raghavachari [7]. Khuller and Raghavachari [10] gave an algorithm for k-ECSS, which was later improved by Gabow [4], who showed that the algorithm obtains a ratio of about 1.61. Cheriyan et al. [2] studied the k-VCSS problem with edge weights and designed an $O(\log k)$ approximation algorithm in graphs with at least $6k^2$ vertices.

The matching-based algorithms were introduced by Cheriyan and Thurimella [1]. They proposed algorithms with ratios of $1 + \frac{1}{k}$ for k-VCSS, $1 + \frac{2}{k+1}$ for k-ECSS, $1 + \frac{1}{k}$ for k-VCSS in directed graphs, and $1 + \frac{4}{\sqrt{k}}$ for k-ECSS in directed graphs. Vempala and Vetta [14] obtained a ratio of 4/3 for 2-VCSS. The ratios were further improved by Krysta and Kumar [12], who introduced the hybrid approach, which was used to derive a 5/4 algorithm by Jothi et al. [9]. A 3/2-approximation algorithm for 3-ECSS has been proposed by Gabow [5] that works on

multigraphs, whereas the earlier algorithm of Cheriyan and Thurimella gets the same ratio in simple graphs only. This ratio has been improved to 4/3 by Gubbala and Raghavachari [8].

1. Cheriyan J, Thurimella R (2000) Approximating minimum-size k-connected spanning subgraphs via matching. SIAM J Comput 30(2):528–560
2. Cheriyan J, Vempala S, Vetta A (2003) An approximation algorithm for the minimum-cost k-vertex connected subgraph. SIAM J Comput 32(4):1050–1055
3. Cook WJ, Cunningham WH, Pulleyblank WR, Schrijver A (1998) Combinatorial optimization. Wiley, New York
4. Gabow HN (2003) Better performance bounds for finding the smallest k-edge connected spanning subgraph of a multigraph. In: SODA, pp 460–469
5. Gabow HN (2004) An ear decomposition approach to approximating the smallest 3-edge connected spanning subgraph of a multigraph. SIAM J Discret Math 18(1):41–70
6. Garg N, Vempala S, Singla A (1993) Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques. In: SODA, pp 103–111
7. Gubbala P, Raghavachari B (2005) Approximation algorithms for the minimum cardinality two-connected spanning subgraph problem. In: Jünger M, Kaibel V (eds) IPCO, vol 3509, Lecture notes in computer science. Springer, Berlin, pp 422–436
8. Gubbala P, Raghavachari B (2007) A 4/3-approximation algorithm for minimum 3-edge-connectivity. In: Proceedings of the workshop on algorithms and data structures (WADS) August 2007, Halifax, pp 39–51
9. Jothi R, Raghavachari B, Varadarajan S (2003) A 5/4-approximation algorithm for minimum 2-edge-connectivity. In: SODA, pp 725–734
10. Khuller S, Raghavachari B (1996) Improved approximation algorithms for uniform connectivity problems. J Algorithms 21(2):434–450
11. Khuller S, Vishkin U (1994) Biconnectivity approximations and graph carvings. J ACM 41(2):214–235
12. Krysta P, Kumar VSA (2001) Approximation algorithms for minimum size 2-connectivity problems. In: Ferreira A, Reichel H (eds) STACS. Lecture notes in computer science, vol 2010. Springer, Berlin, pp 431–442
13. Nagamochi H, Ibaraki T (1992) A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. Algorithmica 7(5–6):583–596
14. Vempala S, Vetta A (2000) Factor 4/3 approximations for minimum 2-connected subgraphs. In: Jansen K, Khuller S (eds) APPROX. Lecture notes in computer science, vol 1913. Springer, Berlin, pp 262–273

Graph Isomorphism

Brendan D. McKay
Department of Computer Science, Australian National University, Canberra, ACT, Australia

Keywords

Graph matching; Symmetry group

Years and Authors of Summarized Original Work

1980; McKay

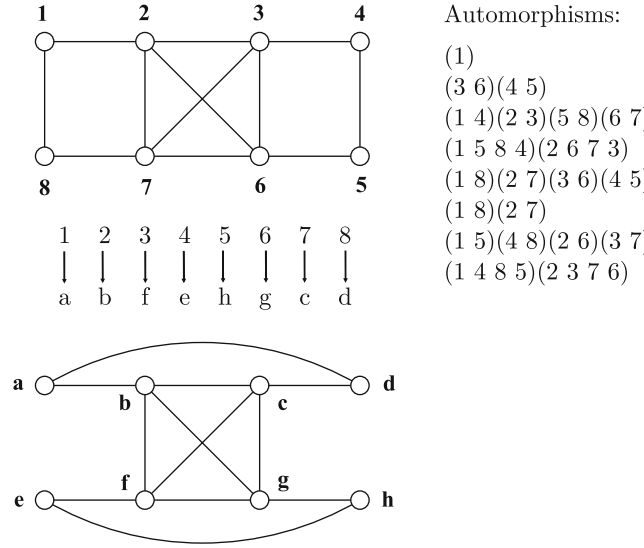
Problem Definition

The problem of determining isomorphism of two combinatorial structures is a ubiquitous one, with applications in many areas. The paradigm case of concern in this chapter is isomorphism of two graphs. In this case, an isomorphism consists of a bijection between the vertex sets of the graphs which induces a bijection between the edge sets of the graphs. One can also take the second graph to be a copy of the first, so that isomorphisms map a graph onto themselves. Such isomorphisms are called *automorphisms* or, less formally, *symmetries*. The set of all automorphisms forms a group under function composition called the *automorphism group*. Computing the automorphism group is a problem rather similar to that of determining isomorphisms.

Graph isomorphism is closely related to many other types of isomorphism of combinatorial structures. In the section entitled "Applications", several examples are given.

Formal Description

A *graph* is a pair $G = (V, E)$ of finite sets, with E being a set of 2-tuples (v, w) of elements of V. The elements of V are called *vertices* (also *points*, *nodes*), while the elements of E are called



Graph Isomorphism, Fig. 1 Example of an isomorphism and an automorphism group

directed edges (also *arcs*). A complementary pair $(v, w), (w, v)$ of directed edges $(v \neq w)$ will be called an *undirected edge* and denoted $\{v, w\}$. A directed edge of the form (v, v) will also be considered an undirected edge, called a *loop* (also *self-loop*). The word "edges" without qualification will indicate undirected edges, directed edges, or both.

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, an *isomorphism* from G_1 to G_2 is a bijection from V_1 to V_2 such that the induced action on E_1 is a bijection onto E_2. If $G_1 = G_2$, then the isomorphism is an *automorphism* of G_1. The set of all *automorphisms* of G_1 is a group under function composition, called the *automorphism group* of G_1, and denoted Aut (G_1).

In Fig. 1 two isomorphic graphs are shown, together with an isomorphism between them and the automorphism group of the first.

Canonical Labeling

Practical applications of graph isomorphism testing do not usually involve individual pairs of graphs. More commonly, one must decide whether a certain graph is isomorphic to any of a collection of graphs (the database lookup problem) or one has a collection of graphs and needs to identify the isomorphism classes in it (the graph sorting problem). Such applications are not well served by an algorithm that can only test graphs in pairs.

An alternative is a *canonical labeling* algorithm. The essential idea is that in each isomorphism class there is a unique, *canonical* graph which the algorithm can find, given as input any graph in the isomorphism class. The canonical graph might be, for example, the least graph in the isomorphism class according to some ordering (such as lexicographic) of the graphs in the class. Practical algorithms usually compute a canonical form designed for efficiency rather than ease of description.

Key Results

The graphisomorphism problem plays a key role in modern complexity theory. It is not known to be solvable in polynomial time, nor to be NP-complete, nor is it known to be in the class co-NP. See [3, 8] for details. Polynomial-time algorithms are known for many special classes, notably graphs with bounded genus, bounded degree, bounded tree-width, and bounded eigenvalue multiplicity. The fastest theoretical algorithm for general graphs requires $\exp(n^{1/2+o(1)})$ time [1], but it is not known to be practical.

In this entry, the focus is on the program `nauty`, which is generally regarded as the most successful for practical use. McKay wrote the first version of `nauty` in 1976 and described its method of operation in [5]. It is known [7] to have exponential worst-case time, but in practice the worst case is rarely encountered.

The input to `nauty` is a graph with colored vertices. Two outputs are produced. The first is a set of generators for the color-preserving automorphism group. Though it is rarely necessary, the full group can also be developed element by element. The second, optional, output is a canonical graph. The canonical graph has the following property: two input graphs with the same number of vertices of each color have the same canonical graph if and only if they are isomorphic by a color-preserving isomorphism.

Two graph data structures are supported: a packed adjacency matrix suitable for small dense graphs and a linked list suitable for large sparse graphs.

Applications

As mentioned, `nauty` can handle graphs with colored vertices. In this section, it is described how several other types of isomorphism problems can be solved by mapping them onto a problem for vertex-colored graphs.

Isomorphism of Edge-Colored Graphs

An isomorphism of two graphs, each with both vertices and edges colored, is defined in the obvious way. An example of such a graph appears at the left of Fig. 2.

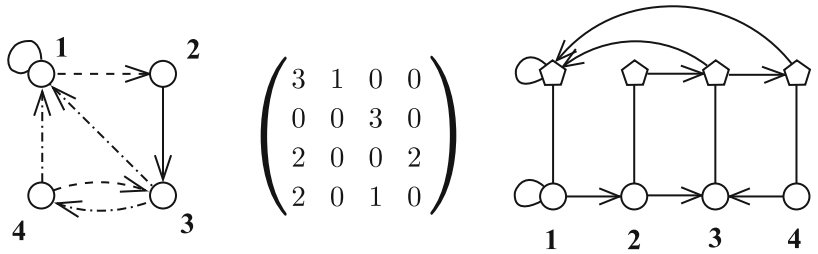
In the center of the figure the colors are identified with the integers $1, 2, 3$. At the right of the figure an equivalent vertex-colored graph is shown. In this case there are two layers, each with its own color. Edges of color 1 are represented as an edge in the first (lowest) layer, edges of color 2 are represented as an edge in the second layer, and edges of color 3 are represented as edges in both layers. It is now easy to see that the automorphism group of the new graph (specifically, its action on the first layer) is the automorphism group of the original graph. Moreover, the order in which a canonical labeling of the new graph labels the vertices of the first layer can be taken to be a canonical labeling of the original graph.

More generally, if the edge colors are integers in $\{1, 2, \ldots, 2^d - 1\}$, there are d layers, and the binary expansion of each color number dictates which layers contain edges. The vertical threads (each corresponding to one vertex of the original graph) can be connected using either paths or cliques. If the original graph has n vertices and k colors, the new graph has $O(n \log k)$ vertices. This can be improved to $O(n\sqrt{\log k})$ vertices by also using edges that are not horizontal.

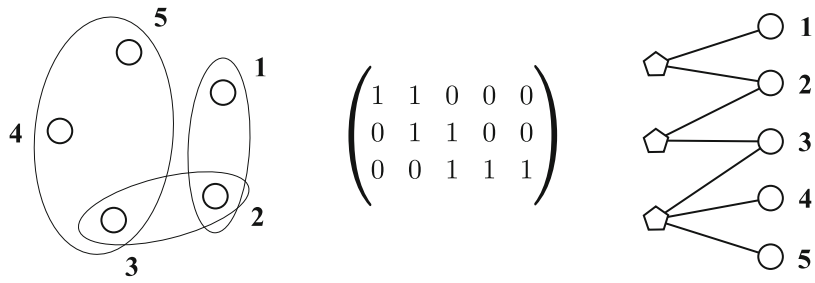
Isomorphism of Hypergraphs and Designs

A *hypergraph* is similar to an undirected graph except that the edges can be vertex sets of any size, not just of size 2. Such a structure is also called a *design*.

On the left of Fig. 3 there is a hypergraph with five vertices, two edges of size 2, and one edge of size 3. On the right is an equivalent vertex-colored graph. The vertices on the left, colored with one color, represent the hypergraph edges, while the edges on the right, colored with a



Graph Isomorphism, Fig. 2 Graph isomorphism with colored edges



Graph Isomorphism, Fig. 3 Hypergraph/design isomorphism as graph isomorphism

different color, represent the hypergraph vertices. The edges of the graph indicate the hypergraph incidence (containment) relationship.

The edge-vertex incidence matrix appears in the center of the figure. This can be any binary matrix at all, which correctly suggests that the problem under consideration is just that of determining the 0–1 matrix equivalence under independent permutation of the rows and columns. By combining this idea with the previous construction, such an equivalence relation on the set of matrices with arbitrary entries can be handled.

Other Examples

For several applications to equivalence operations such as isotopy, important for Latin squares and quasigroups, see [6].

Another important type of equivalence relates matrices over $\{-1, +1\}$. As well as permuting rows and columns, it allows multiplication of rows and columns by -1. A method of converting this *Hadamard equivalence* problem to a graph isomorphism problem is given in [4].

Experimental Results

`Nauty` gives a choice of sparse and dense data structures, and some special code for difficult graph classes. For the following timing examples, the best of the various options are used for a single CPU of a 2.4 GHz Intel Core-duo processor.

1. Random graph with 10,000 vertices, $p = \frac{1}{2}$: 0.014 s for group only, 0.4 s for canonical labeling as well.
2. Random cubic graph with 100,000 vertices: 8 s.
3. 1-skeleton of 20-dimensional cube (1,048,576 vertices, group size 2.5×10^{24}): 92 s.
4. 3-dimensional mesh of size 50 (125,000 vertices): 0.7 s.
5. 1027-vertex strongly regular graph from random Steiner triple system: 0.6 s.

Examples of more difficult graphs can be found in the `nauty` documentation.

URL to Code

The source code of `nauty` is available at http://cs.anu.edu.au/~bdm/nauty/. Another implementation of the automorphism group portion of nauty, highly optimized for large sparse graphs, is available as `saucy` [2]. `Nauty` is also incorporated into a number of general-purpose packages, including GAP, Magma, and MuPad.

Cross-References

▸ Abelian Hidden Subgroup Problem
▸ Parameterized Algorithms for Drawing Graphs

Recommended Reading

1. Babai L, Luks E (1983) Canonical labelling of graphs. In: Proceedings of the 15th annual ACM symposium on theory of computing. ACM, New York, pp 171–183
2. Darga PT, Liffiton MH, Sakallah KA, Markov IL (2004) Exploiting structure in symmetry generation for CNF. In: Proceedings of the 41st design automation

conference, pp 530–534. Source code at http://vlsicad.eecs.umich.edu/BK/SAUCY/
3. Köbler J, Schöning U, Torán J (1993) The graph isomorphism problem: its structural complexity. Birkhäuser, Boston
4. McKay BD (1979) Hadamard equivalence via graph isomorphism. Discret Math 27:213–214
5. McKay BD (1981) Practical graph isomorphism. Congr Numer 30:45–87
6. McKay BD, Meynert A, Myrvold W (2007) Small Latin squares, quasigroups and loops. J Comb Des 15:98–119
7. Miyazaki T (1997) The complexity of McKay's canonical labelling algorithm. In: Groups and computation, II. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 28. American Mathematical Society, Providence, pp 239–256
8. Toran J (2004) On the hardness of graph isomorphism. SIAM J Comput 33:1093–1108

Graph Sketching

Andrew McGregor
School of Computer Science, University of Massachusetts, Amherst, MA, USA

Keywords

Connectivity; Data streams; Linear projections

Years and Authors of Summarized Original Work

2012a; Ahn, Guha, McGregor

Problem Definition

The basic problem we consider is testing whether an undirected graph G on n nodes $\{v_1, \ldots, v_n\}$ is connected. We consider this problem in the following two related models:

1. **Dynamic Graph Stream Model:** The graph G is defined by a sequence of edge insertions and deletions; the edges of G are the set of edges that have been inserted but not subsequently deleted. An algorithm for analyzing G may only read the input sequence from left to right and has limited working memory. If the available memory was $O(n^2)$ bits, then the algorithm could maintain the exact set of edges that have been inserted but not deleted. The primary objective in designing an algorithm in the stream model is to reduce the amount of memory required. Ideally, the time to process each element of the stream and the post-processing time should be small but ensuring this is typically a secondary objective.
2. **Simultaneous Communication Model:** We consider the n rows of the adjacency matrix of G to be partitioned between n players $P_1, \ldots, P_n$ where P_i receives the ith row of the matrix. This means that the existence of any edge is known by exactly two players. An additional player Q wants to evaluate a property of G, and to facilitate this, each player P_i simultaneously sends a message m_i to Q such that Q may evaluate the property given the messages $m_1, m_2, \ldots, m_n$. With n-bit messages from each player, Q could learn the entire graph and the problem would be uninteresting. The objective is to minimize the number of bits sent by each player. Note that the P_i players may not communicate to each other and that each message m_i must be constructed given only the ith row of the adjacency matrix and possibly a set of random bits that is known to all the players.

If there were no edge deletions in the data stream setting, it would be simple to determine whether G was connected using $O(n \log n)$ memory since it is possible to maintain the connected components of the graph; whenever an edge is added, we merge the connected components containing the endpoints of this edge. This algorithm is optimal in terms of space [19]. Such an approach does not extend if edges may also be deleted since it is unclear how the connected components should be updated when an edge is deleted within a connected component.

To illustrate the challenge in the simultaneous communication model, suppose G is connected but $G \setminus \{e\}$ is disconnected for some edge e. The player Q can only learn about the existence of the edge $e = \{v_i, v_j\}$ from either player P_i or player

P_j, but since both of these players have limited knowledge of the graph, neither will realize the important role this edge plays in determining the connectivity of the graph.

Linear Sketches

For both models the best known algorithms are based on random linear projections, aka *linear sketches*. If we denote the n rows of the adjacency matrix by $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \{0,1\}^n$, then the linear sketches of the graph are $\mathcal{A}_1(\mathbf{x}_1), \ldots, \mathcal{A}_n(\mathbf{x}_n)$ where each $\mathcal{A}_i \in \mathbb{R}^{d \times n}$ is a random matrix chosen according to a specific distribution. Note that the matrices $\mathcal{A}_1, \ldots, \mathcal{A}_n$ need not be chosen independently.

In the simultaneous communication model, the message from player P_i is $m_i = \mathcal{A}_i(\mathbf{x}_i)$, and, assuming that the entries of $\mathcal{A}_i$ have polynomial precision, each of these messages requires $O(d \operatorname{polylog} n)$ bits. In the dynamic graph stream model, the algorithm constructs each $\mathcal{A}_i(\mathbf{x}_i)$ using $O(nd \operatorname{polylog} n)$ bits of space. Note that each $\mathcal{A}_i(\mathbf{x}_i)$ can be constructed incrementally using the following update rules:

on the insertion of $\{v_i, v_j\}$:	$\mathcal{A}_i(\mathbf{x}_i) \leftarrow \mathcal{A}_i(\mathbf{x}_i) + \mathcal{A}_i(\mathbf{e}_j)$
on the deletion of $\{v_i, v_j\}$:	$\mathcal{A}_i(\mathbf{x}_i) \leftarrow \mathcal{A}_i(\mathbf{x}_i) - \mathcal{A}_i(\mathbf{e}_j)$
on the insertion/deletion of $\{v_j, v_k\}$ for $i \notin \{j,k\}$:	$\mathcal{A}_i(\mathbf{x}_i) \leftarrow \mathcal{A}_i(\mathbf{x}_i)$

where $\mathbf{e}_j$ is the characteristic vector of the set $\{j\}$. Hence, we have transformed the problem of designing an efficient algorithm into finding the minimum d such that there exists a distribution of matrices $\mathcal{A}_1, \ldots, \mathcal{A}_n \in \mathbb{R}^{d \times n}$ such that for any graph G, we can determine (with high probability) whether G is connected given $\mathcal{A}_1(\mathbf{x}_1), \ldots, \mathcal{A}_n(\mathbf{x}_n)$.

Key Results

The algorithm for connectivity that we present in this entry, and much of the subsequent work on graph sketching, fits the following template. First, we consider a basic "non-sketch" algorithm for the graph problem in question. Second, we design sketches $\mathcal{A}_i$ such that it is possible to emulate the steps of the basic algorithm given only the projections $\mathcal{A}_i(\mathbf{x}_i) \in \mathbb{R}^d$ where $d = O(\operatorname{polylog} n)$.

Connectivity

Basic Non-sketch Algorithm

We pick an incident edge for each node arbitrarily and collapse the resulting connected components into a set of "supernodes." In each subsequent round of the algorithm, we pick an edge from each supernode to another supernode (if one exists) and collapse the connected components into new supernodes. It can be shown that this process terminates after $O(\log n)$ rounds and that the set of edges picked during the different rounds include a spanning forest of the graph. From this we can deduce whether the graph is connected.

Designing the Sketches

There are two main steps required in constructing the sketches for the connectivity algorithm:

An Alternative Graph Representation. Rather than consider the rows of the adjacency matrix $\mathbf{x}_i$, it will be convenient to consider an alternative representation $\mathbf{a}_i \in \{-1,0,1\}^{\binom{n}{2}}$ with entries indexed by pairs

$$\mathbf{a}_i[\{j,k\}] = \begin{cases} 1 & \text{if } i = j < k \text{ and } \{v_j, v_k\} \in E \\ -1 & \text{if } j < k = i \text{ and } \{v_j, v_k\} \in E \\ 0 & \text{otherwise} \end{cases}$$

These vectors have the useful property that for any subset of nodes $\{v_i\}_{i \in S}$, the non-zero en-

tries of $\sum_{i \in S} \mathbf{a}_i$ correspond exactly to the edges across the cut $(S, V \setminus S)$.

For example, consider the graph on nodes $\{v_1, v_2, v_3, v_4\}$ with edges $\{v_1, v_2\}$, $\{v_2, v_3\}$, $\{v_3, v_4\}$, and $\{v_1, v_4\}$. Then

$$\begin{aligned} \mathbf{a}_1 &= (\ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0) \\ \mathbf{a}_2 &= (-1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0) \\ \mathbf{a}_3 &= (\ 0\ \ 0\ \ 0\ -1\ \ 0\ \ 1) \\ \mathbf{a}_4 &= (\ 0\ \ 0\ -1\ \ 0\ \ 0\ -1) \end{aligned}$$

where the entries correspond to the pairs $\{1, 2\}$, $\{1, 3\}$, $\{1, 4\}$, $\{2, 3\}$, $\{2, 4\}$, $\{3, 4\}$ in that order. Note that the nonzero entries of

$$\mathbf{a}_1 + \mathbf{a}_2 = (\ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0)$$

correspond to $\{1, 4\}$ and $\{2, 3\}$ which are exactly the edges across the cut $(S, V \setminus S)$ for $S = \{v_1, v_2\}$.

ℓ_0-Sampling via Linear Sketches. ℓ_0-sampling is a technique that has found numerous applications in data stream processing. We appeal to a result by Jowhari et al. [12] that shows the existence of a distribution over matrices $\mathcal{M} \in \mathbb{R}^{\text{polylog}(n) \times \text{poly}(n)}$ such that for any nonzero vector $\mathbf{z} \in \mathbb{R}^{\text{poly}(n)}$, the index of some nonzero entry of $\mathbf{z}$ can be reconstructed with high probability given $\mathcal{M}(\mathbf{z}) \in \mathbb{R}^{\text{polylog}(n)}$. Note that we do not get to choose which entry is reconstructed.

Emulation Basic Algorithm via Sketches

Let $\mathcal{M}_1, \ldots, \mathcal{M}_r$ be $r = O(\log n)$ independent sketch matrices for ℓ_0-sampling. Given $\mathcal{M}_j(\mathbf{a}_i)$ for all $j \in [r]$ and $i \in [n]$, we can emulate the basic algorithm as follows:

1. Given $\mathcal{M}_1(\mathbf{a}_1), \mathcal{M}_1(\mathbf{a}_2), \ldots, \mathcal{M}_1(\mathbf{a}_n)$, we may emulate the first round of the algorithm since from each $\mathcal{M}_1(\mathbf{a}_i)$ we may reconstruct a nonzero entry of $\mathbf{a}_i$, and these nonzero entries correspond to edges incident to v_i.
2. To emulate round $j > 1$ of the algorithm, suppose S is one of the connected components already constructed. Then, given

$$\sum_{i \in S} \mathcal{M}_j(\mathbf{a}_i) = \mathcal{M}_j\left(\sum_{i \in S} \mathbf{a}_i\right)$$

we may reconstruct a nonzero entry of $\sum_{i \in S} \mathbf{a}_i$ which corresponds to an edge across the cut $(S, V \setminus S)$.

Extensions and Further Work

Subsequent work has extended the above results significantly. If d is increased to $O(k \operatorname{polylog} n)$ then, it is possible to test whether every cut has at least k edges [1]. With $d = O(\epsilon^{-2} \operatorname{polylog} n)$, it is possible to construct graph sparsifiers that can be used to estimate the size of *every* cut up to a $(1 + \epsilon)$ factor [2] along with spectral properties such as the eigenvalues of the graph [14]. With $d = O(\epsilon^{-1} k \operatorname{polylog} n)$, it is possible to distinguish graphs which are not k-vertex connected from those that are at least $(1 + \epsilon)k$-vertex connected [11]. Some of the above results have also been extended to hypergraphs [11]. The algorithm presented in this entry can be implemented with $O(\operatorname{polylog} n)$ update time in the dynamic graph stream model, but a connectivity query may take $\Omega(n)$ time. This was addressed in subsequent work by Kapron et al. [15].

More generally, solving graph problems via linear sketches has become a very active area of research [1–8, 10, 11, 13, 14, 16, 17]. Other problems that have been considered include approximating the densest subgraph [6, 9, 18], maximum matching [5, 7, 8, 16], vertex cover and hitting set [8], correlation clustering [4], and estimating the number of triangles [17].

Cross-References

▸ Counting Triangles in Graph Streams

Recommended Reading

1. Ahn KJ, Guha S, McGregor A (2012) Analyzing graph structure via linear measurements. In: Twenty-third annual ACM-SIAM symposium on discrete algorithms, SODA 2012, pp 459–467. http://

portal.acm.org/citation.cfm?id=2095156&CFID=63838676&CFTOKEN=79617016
2. Ahn KJ, Guha S, McGregor A (2012) Graph sketches: sparsification, spanners, and subgraphs. In: 31st ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, pp 5–14. doi:10.1145/2213556.2213560, http://doi.acm.org/10.1145/2213556.2213560
3. Ahn KJ, Guha S, McGregor A (2013) Spectral sparsification in dynamic graph streams. In: APPROX, pp 1–10. doi:10.1007/978-3-642-40328-6_1, http://dx.doi.org/10.1007/978-3-642-40328-6_1
4. Ahn KJ, Cormode G, Guha S, McGregor A, Wirth A (2015) Correlation clustering in data streams. In: ICML, Lille
5. Assadi S, Khanna S, Li Y, Yaroslavtsev G (2015) Tight bounds for linear sketches of approximate matchings. CoRR abs/1505.01467. http://arxiv.org/abs/1505.01467
6. Bhattacharya S, Henzinger M, Nanongkai D, Tsourakakis CE (2015) Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In: STOC, Portland
7. Bury M, Schwiegelshohn C (2015) Sublinear estimation of weighted matchings in dynamic data streams. CoRR abs/1505.02019. http://arxiv.org/abs/1505.02019
8. Chitnis RH, Cormode G, Esfandiari H, Hajiaghayi M, McGregor A, Monemizadeh M, Vorotnikova S (2015) Kernelization via sampling with applications to dynamic graph streams. CoRR abs/1505.01731. http://arxiv.org/abs/1505.01731
9. Esfandiari H, Hajiaghayi M, Woodruff DP (2015) Applications of uniform sampling: densest subgraph and beyond. CoRR abs/1506.04505. http://arxiv.org/abs/1506.04505
10. Goel A, Kapralov M, Post I (2012) Single pass sparsification in the streaming model with edge deletions. CoRR abs/1203.4900. http://arxiv.org/abs/1203.4900
11. Guha S, McGregor A, Tench D (2015) Vertex and hypergraph connectivity in dynamic graph streams. In: PODS, Melbourne
12. Jowhari H, Saglam M, Tardos G (2011) Tight bounds for lp samplers, finding duplicates in streams, and related problems. In: PODS, Athens, pp 49–58
13. Kapralov M, Woodruff DP (2014) Spanners and sparsifiers in dynamic streams. In: ACM symposium on principles of distributed computing, PODC '14, Paris, 15–18 July 2014, pp 272–281. doi:10.1145/2611462.2611497, http://doi.acm.org/10.1145/2611462.2611497
14. Kapralov M, Lee YT, Musco C, Musco C, Sidford A (2014) Single pass spectral sparsification in dynamic streams. In: FOCS, Philadelphia
15. Kapron B, King V, Mountjoy (2013) Dynamic graph connectivity in polylogarithmic worst case time. In: SODA, New Orleans, pp 1131–1142
16. Konrad C (2015) Maximum matching in turnstile streams. CoRR abs/1505.01460. http://arxiv.org/abs/1505.01460
17. Kutzkov K, Pagh R (2014) Triangle counting in dynamic graph streams. In: Algorithm theory – SWAT 2014 – 14th Scandinavian symposium and workshops, proceedings, Copenhagen, 2–4 July 2014, pp 306–318. doi:10.1007/978-3-319-08404-6_27, http://dx.doi.org/10.1007/978-3-319-08404-6_27
18. McGregor A, Tench D, Vorotnikova S, Vu H (2015) Densest subgraph in dynamic graph streams. In: Mathematical foundations of computer science 2015 – 40th international symposium, MFCS 2014, Proceedings, Part I, Milano, 24–28 Aug 2014
19. Sun X, Woodruff D (2015) Tight bounds for graph problems in insertion streams. In: Approximation algorithms for combinatorial optimization, eighteenth international workshop, APPROX 2015, Proceedings, Princeton, 24–26 Aug 2015

Greedy Approximation Algorithms

Weili Wu[1,2,3] and Feng Wang[4]
[1]College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi Province, China
[2]Department of Computer Science, California State University, Los Angeles, CA, USA
[3]Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA
[4]Mathematical Science and Applied Computing, Arizona State University at the West Campus, Phoenix, AZ, USA

Keywords

Technique for analysis of greedy approximation

Problem Definition

Consider a graph $G = (V, E)$. A subset C of V is called a *dominating set* if every vertex is either in C or adjacent to a vertex in C. If, furthermore, the subgraph induced by C is connected, then C is called a *connected dominating set*.

Given a connected graph G, find a connecting dominating set of minimum cardinality. This problem is denoted by MCDS and is NP-hard. Its optimal solution is called a *minimum*

connected dominating set. The following is a greedy approximation with potential function f.

Greedy Algorithm A:
$C \leftarrow \emptyset$;
while $f(C) > 2$**do**
choose a vertex x to maximize $f(C) - f(C \cup \{x\})$ and
$C \leftarrow C \cup \{x\}$; output C.

Here, f is defined as $f(C) = p(C) + q(C)$ where $p(C)$ is the number of connected components of subgraph induced by C and $q(C)$ is the number of connected components of subgraph with vertex set V and edge set $\{(u, v) \in E \mid u \in C \text{ or v} \in \text{C}\}$. f has an important property that C is a connected dominating set if and only if $f(C) = 2$.

If C is a connected dominating set, then $p(C) = q(C) = 1$, and hence $f(C) = 2$. Conversely, suppose $f(C \cup \{x\}) = 2$. Since $p(C) \geq 1$ and $q(C) \geq 1$, one has $p(C) = q(C) = 1$ which implies that C is a connected dominating set. f has another property, for G with at least three vertices, that if $f(C) > 2$, then there exists $x \in V$ such that $f(C) - f(C \cup \{x\}) > 0$. In fact, for $C = \emptyset$, since G is a connected graph with at least three vertices, there must exist a vertex x with degree at least two, and for such a vertex x, $f(C \cup \{x\}) < f(C)$. For $C \neq \emptyset$, consider a connected component of the subgraph induced by C. Let B denote its vertex set which is a subset of C. For every vertex y adjacent to B, if y is adjacent to a vertex not adjacent to B and not in C, then $p(C \cup \{y\}) < p(C)$ and $q(C \cup \{y\}) \leq q(C)$; if y is adjacent to a vertex in $C - B$, then $p(C \cup \{y\}) \leq p(C)$ and $q(C \cup \{y\}) < q(C)$.

Now, look at a possible analysis for the above greedy algorithm: Let $x_1, \ldots, X_g$ be vertices chosen by the greedy algorithm in the ordering of their appearance in the algorithm. Denote $C_i = \{x_1, \ldots, x_i\}$. Let $C^* = \{y_1, \ldots, y_{opt}\}$ be a minimum connected dominating set. Since adding C^* to C_i will reduce the potential function value from $f(C_i)$ to 2, the value of f reduced by a vertex in C^* would be $(f(C_i) - 2)/opt$ in average. By the greedy rule for choosing $x_i + 1$, one has

$$f(C_i) - f(C_{i+1}) \geq \frac{f(C_i) - 2}{opt}.$$

Hence,

$$\begin{aligned} f(C_{i+1}) - 2 &\leq (f(C_i) - 2)\left(1 - \tfrac{1}{opt}\right) \\ &\leq (f(\emptyset) - 2)\left(1 - \tfrac{1}{opt}\right)^{i+1} \\ &= (n-2)\left(1 - \tfrac{1}{opt}\right)^{i+1}, \end{aligned}$$

where $n = |V|$. Note that $1 - 1/opt \leq \mathrm{e}^{-1/opt}$. Hence,

$$f(C_i) - 2 \leq (n-2)\mathrm{e}^{-i/opt}.$$

Choose i such that $f(C_i) \geq opt + 2 > f(C_{i+1})$. Then

$$opt \leq (n-2)\,\mathrm{e}^{-i/opt}$$

and

$$g - i \leq opt.$$

Therefore,

$$g \leq opt + i \leq opt\left(1 + \ln\frac{n-2}{opt}\right).$$

Is this analysis correct? The answer is NO. Why? How could one give a correct analysis? This entry will answer those questions and introduce a new general technique, analysis of greedy approximation with nonsubmodular potential function.

Key Results

The Role of Submodularity

Consider a set X and a function f defined on the power set 2^X, i.e., the family of all subsets of X. f is said to be *submodular* if for any two subsets A and B in 2^X,

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B).$$

For example, consider a connected graph G. Let X be the vertex set of G. The function $-q(C)$

defined in the last section is submodular. To see this, first mention a property of submodular functions.

A submodular function f is *normalized* if $f(\emptyset) = 0$. Every submodular function f can be normalized by setting $g(A) = f(A) - f(\emptyset)$. A function f is *monotone increasing* if $f(A) \leq f(B)$ for $A \subset B$. Denote $\Delta_x f(A) = f(A \cup \{x\}) - f(A)$.

Lemma 1 *A function $f : 2^X \to R$ is submodular if and only if $\Delta_x f(A) \leq \Delta_x f(B)$ for any $x \in X - B$ and $A \subseteq B$. Moreover, f is monotone increasing if and only if $\Delta_x f(A) \leq \Delta_x f(B)$ for any $x \in B$ and $A \subseteq B$.*

Proof If f is submodular, then for $x \in X - B$ and $A \subseteq B$, one has

$$\begin{aligned} & f(A \cup \{x\}) + f(B) \\ & \quad \geq f((A \cup \{x\}) \cup B) + f(A \cup \{x\}) \cap B) \\ & \quad = f(B \cup \{x\}) + f(A), \end{aligned}$$

that is,

$$\Delta_x f(A) \geq \Delta_x f(B). \tag{1}$$

Conversely, suppose (1) holds for any $x \in B$ and $A \subseteq B$. Let C and D be two sets and $C/D = \{x_1, \ldots, x_k\}$. Then

$$\begin{aligned} f(C \cup D) - f(D) & = \sum_{i=1}^{k} \Delta_{x_i} f(D \cup \{x_1, \ldots, x_{i-1}) \\ & \leq \sum_{i=1}^{k} \Delta_{x_i} f((C \cap D) \cup \{x_1, \ldots, x_{i-1}) \\ & = f(C) - f(C \cap D). \end{aligned}$$

If f is monotone increasing, then for $A \subseteq B$, $f(A) \leq f(B)$. Hence, for $x \in B$,

$$\Delta) x f(A) \geq 0 = \Delta_x f(B).$$

Conversely, if $\Delta_x f(A) \geq \Delta_x f(B)$ for any $x \in B$ and $A \subseteq B$, then for any x and A, $\Delta_x f(A) \geq \Delta_x f(A \cup \{x\}) = 0$, that is, $f(A) \leq f(A \cup \{x\})$. Let $B - A = \{x_1, \ldots, x_k\}$. Then

$$\begin{aligned} f(A) & \leq f(A \cup \{x_1\}) \\ & \leq f(A \cup \{x_1, x_2\}) \leq \cdots \leq f(B). \end{aligned}$$

Next, the submodularity of $-q(A)$ is studied. □

Lemma 2 *If $A \subset B$, then $\Delta_y q(A) \geq \Delta_y q(B)$.*

Proof Note that each connected component of graph $(V, D(B))$ is constituted by one or more connected components of graph $(V, D(A))$ since $A \subset B$. Thus, the number of connected components of $(V, D(B))$ dominated by y is no more than the number of connected components of $(V, D(A))$ dominated by y. Therefore, the lemma holds.

The relationship between submodular functions and greedy algorithms has been established for a long time [2].

Let f be a normalized, monotone increasing, submodular integer function. Consider the minimization problem

$$\begin{aligned} \min \quad & c(A) \\ \text{subject to } & A \in \mathcal{C}_f. \end{aligned}$$

where c is a nonnegative cost function defined on 2^X and $\mathcal{C}_f = \{C \,\mathrm{mid}\, f(C \cup \{x\}) - f(C) = 0 \text{ for all } x \in X\}$. The following is a greedy algorithm to produce approximation solution for this problem. □

Greedy Algorithm B

input submodular function f and cost function c;
$A \leftarrow \emptyset$;
while there exists $x \in E$ such that $\Delta_x f(A) > 0$
do select a vertex x that maximizes $\Delta_x f(A)/c(x)$ and set
$A \leftarrow A \cup \{x\}$;
return A.

The following two results are well known.

Theorem 1 *If f is a normalized, monotone increasing, submodular integer function, then Greedy Algorithm B produces an approximation solution within a factor of $H(\gamma)$ from optimal, where $\gamma = max_{x \in E} f(\{x\})$.*

Theorem 2 *Let f be a normalized, monotone increasing, submodular function and c a nonnegative cost function. If in Greedy Algorithm B,*

selected x always satisfies $\Delta_x f(A_{i-1})/c(x) \geq 1$, then it produces an approximation solution within a factor of $1 + \ln(f^*/opt)$ from optimal for the above minimization problem where $f^* = f(A^*)$ and $opt = c(A^*)$ for optimal solution A^*.

Now, come back to the analysis of Greedy Algorithm A for the MCDS. It looks like that the submodularity of f is not used. Actually, the submodularity was implicitly used in the following statement:

"Since adding C^* to C_i will reduce the potential function value from $f(C_i)$ to 2, the value of f reduced by a vertex in C^* would be $(f(C_i - 2)/opt$ in average. By the greedy rule for choosing $x_i + 1$, one has

$$f(C_i) - f(C_{i+1}) \geq \frac{f(C_i) - 2}{opt}."$$

To see this, write this argument more carefully.

Let $C^* = \{y_1, \ldots, y_{opt}\}$ and denote $C_j^* = \{y_1, \ldots, y_j\}$. Then

$$\begin{aligned} f(C_i) - 2 &= f(C_i) - f(C_i \cup C^*) \\ &= \sum_{j=1}^{opt} [f(C_i \cup C_{j-1}^*) - f(C_i \cup C_j^*)] \end{aligned}$$

where $C_0^* = \emptyset$. By the greedy rule for choosing $x_i + 1$, one has

$$f(C_i) - f(C_{i+1}) \geq f(C_i) - f(C_i \cup \{y_j\})$$

for $j = 1, \ldots, opt$. Therefore, it needs to have

$$\begin{aligned} -\Delta_{y_j} f(C_i) &= f(C_i) - f(C_i \cup \{y_j\}) \\ &\geq f(C_i \cup C_{j-1}^*) - f(C_i \cup C_j^*) \\ &= -\Delta_{y_j} f(C_i \cup C_{j-1}^*) \end{aligned} \tag{2}$$

in order to have

$$f(C_i) - f(C_{i+1}) \geq \frac{f(C_i) - 2}{opt}.$$

Equation (2) asks the submodularity of $-f$. Unfortunately, $-f$ is not submodular. A counterexample can be found in [2]. This is why the analysis of Greedy Algorithm A in section "Problem Definition" is incorrect.

Giving Up Submodularity

Giving up submodularity is a challenge task since it is open for a long time. But, it is possible based on the following observation on (2) by Du et al. [1]: **The submodularity of $-f$ is applied to increment of a vertex y_j belonging to optimal solution C^*.**

Since the ordering of y_j's is flexible, one may arrange it to make $\Delta_{yj} f(C_i) - \Delta_{yj} f(C_i \cup C_{j-1}^*)$ under control. This is a successful idea for the MCDS.

Lemma 3 *Let y_j's be ordered in the way that for any $j = 1, \ldots, opt$, $\{y_1, \ldots, y_j\}$ induces a connected subgraph. Then*

$$\Delta_{y_j} f(C_i) - \Delta_{y_j} f(C_i \cup C_{j-1}^*) \leq 1.$$

Proof Since all $y_1, \ldots, y_{j-1}$ are connected, y_j can dominate at most one additional connected component in the subgraph induced by $C_{i-1} \cup C_{j-1}^*$ than in the subgraph induced by $c_i - 1$. Hence,

$$\Delta_{y_j} p(C_i) - \Delta_{y_j} f(C_i \cup C_{j-1}^*) \leq 1.$$

Moreover, since $-q$ is submodular,

$$\Delta_{y_j} q(C_i) - \Delta_{y_j} q(C_i \cup C_{j-1}^*) \leq 0.$$

Therefore,

$$\Delta_{y_j} f(C_i) - \Delta_{y_j} f(C_i \cup C_{j-1}^*) \leq 1.$$

Now, one can give a correct analysis for the greedy algorithm for the MCDS [3].

By Lemma 3,

$$f(C_i) - f(C_{i+1}) \geq \frac{f(C_i) - 2}{opt} - 1.$$

Hence,

$$
\begin{aligned}
f(C_{i+1}) - 2 - opt &\\
&\le (f(C_i) - 2 + opt)\left(1 - \tfrac{1}{opt}\right)\\
&\le (f(\emptyset) - 2 - opt)\left(1 - \tfrac{1}{opt}\right)^{i+1}\\
&= (n - 2 - opt)\left(1 - \tfrac{1}{opt}\right)^{i+1},
\end{aligned}
$$

where $n = |V|$. Note that $1 - 1/opt \le \mathrm{e}^{-1/opt}$. Hence,

$$f(C_i) - 2 - opt \le (n-2)\mathrm{e}^{-i/opt}.$$

Choose i such that $f(C_i) \ge 2 \cdot opt + 2 > f(C_{i+1})$. Then

$$opt \le (n-2)\mathrm{e}^{-i/opt}$$

and

$$g - i \le 2 \cdot opt.$$

Therefore,

$$g \le 2 \cdot opt + i \le opt\left(2 + \ln\frac{n-2}{opt}\right) \le opt(2 + \ln\delta)$$

where δ is the maximum degree of input graph G. □

Applications

The technique introduced in the previous section has many applications, including analysis of iterated 1-Steiner trees for minimum Steiner tree problem and analysis of greedy approximations for optimization problems in optical networks [3] and wireless networks [2].

Open Problems

Can one show the performance ratio $1 + H(\delta)$ for Greedy Algorithm B for the MCDS? The answer is unknown. More generally, it is unknown how to get a clean generalization of Theorem 1.

Cross-References

- Connected Dominating Set
- Exact Algorithms for k SAT Based on Local Search
- Steiner Trees

Acknowledgments Weili Wu is partially supported by NSF grant ACI-0305567.

Recommended Reading

1. Du D-Z, Graham RL, Pardalos PM, Wan P-J, Wu W, Zhao W (2008) Analysis of greedy approximations with nonsubmodular potential functions. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), San Francisco
2. Nemhauser GL, Wolsey LA (1999) Integer and combinatorial optimization. Wiley, Hoboken
3. Ruan L, Du H, Jia X, Wu W, Li Y, Ko K-I (2004) A greedy approximation for minimum connected dominating set. Theor Comput Sci 329:325–330
4. Ruan L, Wu W (2005) Broadcast routing with minimum wavelength conversion in WDM optical networks. J Comb Optim 9:223–235

Greedy Set-Cover Algorithms

Neal E. Young
Department of Computer Science and Engineering, University of California, Riverside, CA, USA

Keywords

Dominating set; Greedy algorithm; Hitting set; Minimizing a linear function subject to a submodular constraint; Set cover

Years and Authors of Summarized Original Work

1974–1979; Chvátal, Johnson, Lovász, Stein

Problem Definition

Given a collection $\mathcal{S}$ of sets over a universe U, a *set cover* $C \subseteq \mathcal{S}$ is a subcollection of the

sets whose union is U. The *set-cover problem* is, given $\mathcal{S}$, to find a minimum-cardinality set cover. In the *weighted set-cover problem*, for each set $s \in \mathcal{S}$, a weight $w_s \geq 0$ is also specified, and the goal is to find a set-cover C of minimum total weight $\sum_{S \in C} w_S$.

Weighted set cover is a special case of *minimizing a linear function subject to a submodular constraint*, defined as follows. Given a collection $\mathcal{S}$ of objects, for each object s a nonnegative weight w_s, and a nondecreasing submodular function $f : 2^{\mathcal{S}} \to \mathbb{R}$, the goal is to find a subcollection $C \subseteq \mathcal{S}$ such that $f(C) = f(\mathcal{S})$ minimizing $\sum_{s \in C} w_s$. (Taking $f(C) = |\cup_{s \in C} s|$ gives weighted set cover.)

Key Results

The *greedy algorithm* for weighted set cover builds a cover by repeatedly choosing a set s that minimizes the weight w_s divided by the number of elements in s not yet covered by chosen sets. It stops and returns the chosen sets when they form a cover:

Let H_k denote $\sum_{i=1}^{k} 1/i \approx \ln k$, where k is the largest set size.

greedy-set-cover(S, w)
1. Initialize $C \leftarrow \emptyset$. Define $f(C) \doteq |\cup_{s \in C} s|$.
2. Repeat until $f(C) = f(S)$:
3. Choose $s \in S$ minimizing the price per element $w_s/[f(C \cup \{s\}) - f(C)]$.
4. Let $C \leftarrow C \cup \{s\}$.
5. Return C.

Theorem 1 *The greedy algorithm returns a set cover of weight at most H_k times the minimum weight of any cover.*

Proof When the greedy algorithm chooses a set s, imagine that it charges the price per element for that iteration to each element newly covered by s. Then, the total weight of the sets chosen by the algorithm equals the total amount charged, and each element is charged once.

Consider any set $s = \{x_k, x_{k-1}, \ldots, x_1\}$ in the optimal set cover C^*. Without loss of generality, suppose that the greedy algorithm covers the elements of s in the order given: $x_k, x_{k-1}, \ldots, x_1$. At the start of the iteration in which the algorithm covers element x_i of s, at least i elements of s remain uncovered. Thus, if the greedy algorithm were to choose s in that iteration, it would pay a cost per element of at most w_s / i. Thus, in this iteration, the greedy algorithm pays at most w_s / i per element covered. Thus, it charges element x_i at most w_s / i to be covered. Summing over i, the total amount charged to elements in s is at most $w_s H_k$. Summing over $s \in C^*$ and noting that every element is in some set in C^*, the total amount charged to elements overall is at most $\sum_{s \in C^*} W_s H_k = H_k \mathrm{OPT}$. □

The theorem was shown first for the unweighted case (each $w_s = 1$) by Johnson [5], Lovász [8], and Stein [13] and then extended to the weighted case by Chvátal [2].

Since then a few refinements and improvements have been shown, including the following:

Theorem 2 *Let $\mathcal{S}$ be a set system over a universe with n elements and weights $w_s \leq 1$. The total weight of the cover C returned by the greedy algorithm is at most $[1 + \ln(n/OPT)]\,OPT + 1$ (compare to [12]).*

Proof Assume without loss of generality that the algorithm covers the elements in order $x_n, x_{n-1}, \ldots, x_1$. At the start of the iteration in which the algorithm covers x_i, there are at least i elements left to cover, and all of them could be covered using multiple sets of total cost OPT. Thus, there is some set that covers not-yet-covered elements at a cost of at most OPT/i per element.

Recall the charging scheme from the previous proof. By the preceding observation, element x_i is charged at most OPT/i. Thus, the total charge to elements $x_n, \ldots, x_i$ is at most $(H_n - H_{i-1})\mathrm{OPT}$. Using the assumption that each $w_s \leq 1$, the charge to each of the remaining elements is at most 1 per element. Thus, the total charge to all elements is at most $i - 1 + (H_n -$

H_{i-1})OPT. Taking $i = 1 + \lceil \text{OPT} \rceil$, the total charge is at most $\lceil \text{OPT} \rceil + (H_n - H_{\lceil \text{OPT} \rceil})\text{OPT} \leq 1 + \text{OPT}(1 + \ln(n/\text{OPT}))$. □

Each of the above proofs implicitly constructs a linear-programming primal-dual pair to show the approximation ratio. The same approximation ratios can be shown with respect to any fractional optimum (solution to the fractional set-cover linear program).

Other Results

The greedy algorithm has been shown to have an approximation ratio of $\ln n - \ln \ln n + O(1)$ [11]. For the special case of set systems whose duals have finite Vapnik-Chervonenkis (VC) dimension, other algorithms have substantially better approximation ratio [1]. Constant-factor approximation algorithms are known for geometric variants of the closely related k-median and facility location problems.

The greedy algorithm generalizes naturally to many problems. For example, for minimizing a linear function subject to a submodular constraint (defined above), the natural extension of the greedy algorithm gives an H_k -approximate solution, where $k = \max_{s \in \mathcal{S}} f(\{s\}) - f(\emptyset)$, assuming f is integer valued [10].

The set-cover problem generalizes to allow each element x to require an arbitrary number r_x of sets containing it to be in the cover. This generalization admits a polynomial-time $O(\log n)$-approximation algorithm [7].

The special case when each element belongs to at most r sets has a simple r-approximation algorithm ([15] § 15.2). When the sets have uniform weights ($w_s = 1$), the algorithm reduces to the following: select any maximal collection of elements, no two of which are contained in the same set; return all sets that contain a selected element.

The variant "Max k-coverage" asks for a set collection of total weight at most k covering as many of the elements as possible. This variant has a $(1 - 1/e)$-approximation algorithm ([15] Problem 2.18) (see [6] for sets with nonuniform weights).

For a general discussion of greedy methods for approximate combinatorial optimization, see ([4] Ch. 4).

Finally, under likely complexity-theoretic assumptions, the ln n approximation ratio is essentially the best possible for any polynomial-time algorithm [3,9].

Applications

Set cover and its generalizations and variants are fundamental problems with numerous applications. Examples include:

- Selecting a small number of nodes in a network to store a file so that all nodes have a nearby copy
- Selecting a small number of sentences to be uttered to tune all features in a speech-recognition model [14]
- Selecting a small number of telescope snapshots to be taken to capture light from all galaxies in the night sky
- Finding a short string having each string in a given set as a contiguous sub-string

Recommended Reading

1. Brönnimann H, Goodrich MT (1995) Almost optimal set covers in finite VC-dimension. Discret Comput Geom 14(4):463–479
2. Chvátal V (1979) A greedy heuristic for the set-covering problem. Math Oper Res 4(3): 233–235
3. Feige U (1998) A threshold of ln n for approximating set cover. J ACM 45(4):634–652
4. Gonzalez TF (2007) Handbook of approximation algorithms and metaheuristics. Chapman & Hall/CRC computer & information science series. Chapman & Hall/CRC, Boca Raton
5. Johnson DS (1974) Approximation algorithms for combinatorial problems. J Comput Syst Sci 9:256–278
6. Khuller S, Moss A, Naor J (1999) The budgeted maximum coverage problem. Inform Process Lett 70(1):39–45
7. Kolliopoulos SG, Young NE (2001) Tight approximation results for general covering integer programs.

In: Proceedings of the forty-second annual IEEE symposium on foundations of computer science, Las Vegas, pp 522–528
8. Lovász L (1975) On the ratio of optimal integral and fractional covers. Discret Math 13:383–390
9. Lund C, Yannakakis M (1994) On the hardness of approximating minimization problems. J ACM 41(5):960–981
10. Nemhauser GL, Wolsey LA (1988) Integer and combinatorial optimization. Wiley, New York
11. Slavik P (1997) A tight analysis of the greedy algorithm for set cover. J Algorithms 25(2): 237–254
12. Srinivasan A (1995) Improved approximations of packing and covering problems. In: Proceedings of the twenty-seventh annual ACM symposium on theory of computing, Heraklion, pp 268–276
13. Stein SK (1974) Two combinatorial covering theorems. J Comb Theor A 16:391–397
14. van Santen JPH, Buchsbaum AL (1997) Methods for optimal text selection. In: Proceedings of the European conference on speech communication and technology, Rhodos, vol 2, pp 553–556
15. Vazirani VV (2001) Approximation algorithms. Springer, Berlin/Heidelberg

H

Hamilton Cycles in Random Intersection Graphs

Charilaos Efthymiou[1] and Paul (Pavlos) Spirakis[2,3,4]
[1]Department of Computer Engineering and Informatics, University of Patras, Patras, Greece
[2]Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece
[3]Computer Science, University of Liverpool, Liverpool, UK
[4]Computer Technology Institute (CTI), Patras, Greece

Keywords

Stochastic order relations between Erdös–Rényi random graph model and random intersection graphs; Threshold for appearance of Hamilton cycles in random intersection graphs

Years and Authors of Summarized Original Work

2005; Efthymiou, Spirakis

Problem Definition

E. Marczewski proved that every graph can be represented by a list of sets where each vertex corresponds to a set and the edges to nonempty intersections of sets. It is natural to ask what sort of graphs would be most likely to arise if the list of sets is generated randomly.

Consider the model of random graphs where each vertex chooses randomly from a universal set the members of its corresponding set, each independently of the others. The probability space that is created is the space of random intersection graphs, $G_{n,m,p}$, where n is the number of vertices, m is the cardinality of a universal set of elements and p is the probability for each vertex to choose an element of the universal set. The model of random intersection graphs was first introduced by M. Karońsky, E. Scheinerman, and K. Singer-Cohen in [4]. A rigorous definition of the model of random intersection graphs follows:

Definition 1 Let n, m be positive integers and $0 \leq p \leq 1$. The random intersection graph $G_{n,m,p}$ is a probability space over the set of graphs on the vertex set $\{1, \ldots, n\}$ where each vertex is assigned a random subset from a fixed set of m elements. An edge arises between two vertices when their sets have at least a common element. Each random subset assigned to a vertex is determined by

$$\mathbf{Pr}\,[\text{vertex } i \text{ chooses element } j] = p$$

with these events mutually independent.

A common question for a graph is whether it has a cycle, a set of edges that form a path so that the

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

first and the last vertex is the same, that visits *all* the vertices of the graph exactly once. We call this kind of cycle the *Hamilton cycle* and the graph that contains such a cycle is called a *Hamiltonian graph*.

Definition 2 Consider an undirected graph $G = (V, E)$ where V is the set of vertices and E the set of edges. This graph contains a Hamilton cycle if and only if there is a simple cycle that contains each vertex in V.

Consider an instance of $G_{n,m,p}$, for specific values of its parameters n, m, and p, what is the probability of that instance to be Hamiltonian? Taking the parameter p, of the model, to be a function of n and m, in [2], a threshold function $P(n, m)$ has been found for the graph property "Contains a Hamilton cycle"; i.e., a function $P(n, m)$ is derived such that

if $p(n, m) \ll P(n, m)$

$$\lim_{n,m\to\infty} \mathbf{Pr}\left[G_{n,m,p} \text{Contains Hamilton cycle}\right] = 0$$

if $p(n, m) \gg P(n, m)$

$$\lim_{n,m\to\infty} \mathbf{Pr}\left[G_{n,m,p} \text{Contains Hamilton cycle}\right] = 1$$

When a graph property, such as "Contains a Hamilton cycle," holds with probability that tends to 1 (or 0) as n, m tend to infinity, then it is said that this property holds (does not hold), "almost surely" or "almost certainly."

If in $G_{n,m,p}$ the parameter m is very small compared to n, the model is not particularly interesting and when m is exceedingly large (compared to n) the behavior of $G_{n,m,p}$ is essentially the same as the Erdös–Rényi model of random graphs (see [3]). If someone takes $m = \lceil n^\alpha \rceil$, for fixed real $\alpha > 0$, then there is some deviation from the standard models, while allowing for a natural progression from sparse to dense graphs. Thus, the parameter m is assumed to be of the form $m = \lceil n^\alpha \rceil$ for some fixed positive real α.

The proof of existence of a Hamilton cycle in $G_{n,m,p}$ is mainly based on the establishment of a *stochastic order relation* between the model $G_{n,m,p}$ and the Erdös–Rényi random graph model $G_{n,\hat{p}}$.

Definition 3 Let n be a positive integer, $0 \le \hat{p} \le 1$. The random graph $G(n, \hat{p})$ is a probability space over the set of graphs on the vertex set $\{1, \ldots, n\}$ determined by

$$\mathbf{Pr}[i, j] = \hat{p}$$

with these events mutually independent.

The stochastic order relation between the two models of random graphs is established in the sense that if $\mathcal{A}$ is an increasing graph property, then it holds that

$$\mathbf{Pr}\left[G_{n,\hat{p}} \in \mathcal{A}\right] \le \mathbf{Pr}\left[G_{n,m,p} \in \mathcal{A}\right]$$

where $\hat{p} = f(p)$. A graph property $\mathcal{A}$ is increasing if and only if given that $\mathcal{A}$ holds for a graph $G(V, E)$ then $\mathcal{A}$ holds for any $G(V, E')$: $E' \supseteq E$.

Key Results

Theorem 1 *Let $m = \lceil n^\alpha \rceil$, where α is a fixed real positive, and C_1, C_2 be sufficiently large constants. If*

$$p \ge C_1 \frac{\log n}{m} \quad \text{for} \quad 0 < \alpha < 1 \quad \text{or}$$

$$p \ge C_2 \sqrt{\frac{\log n}{nm}} \quad \text{for} \quad \alpha > 1$$

then almost all $G_{n,m,p}$ are Hamiltonian. Our bounds are asymptotically tight.

Note that the theorem above says nothing when $m = n$, i.e., $\alpha = 1$.

Applications

The Erdös–Rényi model of random graphs, $G_{n,p}$, is exhaustively studied in computer science because it provides a framework for studying

practical problems such as "reliable network computing" or it provides a "typical instance" of a graph and thus it is used for average case analysis of graph algorithms. However, the simplicity of $G_{n,p}$ means it is not able to capture satisfactorily many practical problems in computer science. Basically, this is because of the fact that in many problems independent edge-events are not well justified. For example, consider a graph whose vertices represent a set of objects that either are placed or move in a specific geographical region, and the edges are radio communication links. In such a graph, we expect that, any two vertices u, w are more likely to be adjacent to each other, than any other, arbitrary, pair of vertices, if both are adjecent to a third vertex v. Even epidemiological phenomena (like the spread of disease) tend to be more accurately captured by this proximity-sensitive random intersection graph model. Other applications may include oblivious resource sharing in a distributive setting, interaction of mobile agents traversing the web etc.

The model of random intersection graphs $G_{n,m,p}$ was first introduced by M. Karońsky, E. Scheinerman, and K. Singer-Cohen in [4] where they explored the evolution of random intersection graphs by studying the thresholds for the appearance and disappearance of small induced subgraphs. Also, J.A. Fill, E.R. Scheinerman, andK. Singer Cohen in [3] proved an equivalence theorem relating the evolution of $G_{n,m,p}$ and $G_{n,p}$, in particular they proved that when $m = n^{\alpha}$ where $\alpha > 6$, the total variation distance between the graph random variables has limit 0. S. Nikoletseas, C. Raptopoulos, and P. Spirakis in [8] studied the existence and the efficient algorithmic construction of close to optimal independent sets in random intersection graphs. D. Stark in [11] studied the degree of the vertices of the random intersection graphs. However, after [2], Spirakis and Raptopoulos, in [10], provide algorithms that construct Hamilton cycles in instances of $G_{n,m,p}$, for p above the Hamiltonicity threshold. Finally, Nikoletseas et al. in [7] study the mixing time and cover time as the parameter p of the model varies.

Open Problems

As in many other random structures, e.g., $G_{n,p}$ and random formulae, properties of random intersection graphs also appear to have threshold behavior. So far threshold behavior has been studied for the induced subgraph appearance and hamiltonicity.

Other fields of research for random intersection graphs may include the study of connectivity behavior, of the model i.e., the path formation, the formation of giant components. Additionally, a very interesting research question is how cover and mixing times vary with the parameter p, of the model.

Cross-References

▸ Independent Sets in Random Intersection Graphs

Recommended Reading

1. Alon N, Spencer JH (2000) The probabilistic method, 2nd edn. Wiley, New York
2. Efthymiou C, Spirakis PG (2005) On the existence of Hamilton cycles in random intersection graphs. In: Proceedings of the 32nd ICALP. LNCS, vol 3580. Springer, Berlin/Heidelberg, pp 690–701
3. Fill JA, Scheinerman ER, Singer-Cohen KB (2000) Random intersection graphs when m = ω(n): an equivalence theorem relating the evolution of the G(n, m, p) and G(n, p) models. Random Struct Algorithms 16:156–176
4. Karoński M, Scheinerman ER, Singer-Cohen K (1999) On random intersection graphs: the subgraph problem. Comb Probab Comput 8:131–159
5. Komlós J, Szemerédi E (1983) Limit distributions for the existence of Hamilton cycles in a random graph. Discret Math 43:55–63
6. Korshunov AD (1977) Solution of a problem of P. Erdös and A. Rényi on Hamilton cycles in non-oriented graphs. Metody Diskr Anal Teoriy Upr Syst Sb Trubov Novosibrirsk 31:17–56
7. Nikoletseas S, Raptopoulos C, Spirakis P (2007) Expander properties and the cover time of random intersection graphs. In: Proceedings of the 32nd MFCS. Springer, Berlin/Heidelberg, pp 44–55
8. Nikoletseas S, Raptopoulos C, Spirakis P (2004) The existence and efficient construction of large independent sets in general random intersection graphs.

In: Proceedings of the 31st ICALP. LNCS, vol 3142. Springer, Berlin/Heidelberg, pp 1029–1040
9. Singer K (1995) Random intersection graphs. PhD thesis, The Johns Hopkins University, Baltimore
10. Spirakis PG, Raptopoulos C (2005) Simple and efficient greedy algorithms for Hamilton cycles in random intersection graphs. In: Proceedings of the 16th ISAAC. LNCS, vol 3827. Springer, Berlin/Heidelberg, pp 493–504
11. Stark D (2004) The vertex degree distribution of random intersection graphs. Random Struct Algorithms 24:249–258

Haplotype Inference on Pedigrees Without Recombinations

Mee Yee Chan[1], Wun-Tat Chan[2], Francis Y.L. Chin[1], Stanley P.Y. Fung[3], and Ming-Yang Kao[4]
[1]Department of Computer Science, University of Hong Kong, Hong Kong, China
[2]College of International Education, Hong Kong Baptist University, Hong Kong, China
[3]Department of Computer Science, University of Leicester, Leicester, UK
[4]Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA

Keywords

Computational biology; Haplotype inference; Pedigree; Recombination

Years aud Authors of Summarized Original Work

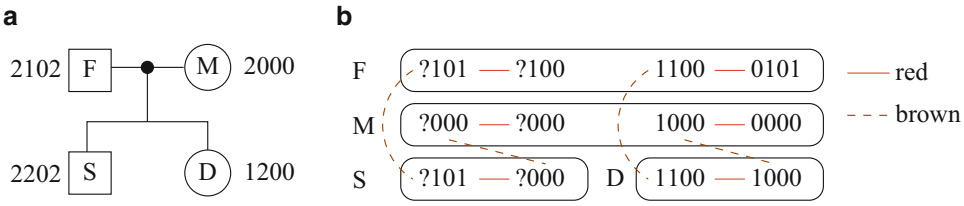
2009; Chan, Chan, Chin, Fung, Kao

Problem Definition

In many diploid organisms like humans, chromosomes come in pairs. Genetic variation occurs in some "positions" along the chromosomes. These genetic variations are commonly modelled in the form of *single nucleotide polymorphisms* (SNPs) [5], which are the nucleotide sites where more than one nucleotide can occur. A *haplotype* is the sequence of linked SNP genetic markers (small segments of DNA) on a single chromosome. However, experiments often yield *genotypes*, which is a blend of the two haplotypes of the chromosome pair. It is more useful to have information on the haplotypes, thus giving rise to the computational problem of inferring haplotypes from genotypes.

The physical position of a marker on a chromosome is called a *locus* and its state is called an *allele*. SNP are often *biallelic*, i.e., the allele can take on two different states, corresponding to two different nucleotides. In the language of computer science, the allele of a biallelic SNP can be denoted by 0 and 1, and a haplotype with m loci is represented as a length-m string in $\{0,1\}^m$ and a genotype as a length-m string in $\{0,1,2\}^m$. Consider a haplotype pair $\langle h_1, h_2\rangle$ and a corresponding genotype g. For each locus, if both haplotypes show a 0, then the genotype must also be 0, and if both haplotypes show a 1, the genotype must also be 1. These loci are called *homozygous*. If however one of the haplotypes shows a 0 and the other a 1, the genotype shows a 2 and the locus is called *heterozygous*. This is called *SNP consistency*. For example, considering a single individual, the genotype $g = 012212$ has four SNP-consistent haplotype pairs: $\{\langle 01\underline{1}1\underline{1}1, 01\underline{0}0\underline{1}0\rangle, \langle 01\underline{1}1\underline{1}0, 01\underline{0}0\underline{1}1\rangle, \langle 01\underline{1}0\underline{1}1, 01\underline{0}1\underline{1}0\rangle, \langle 01\underline{1}0\underline{1}0, 01\underline{0}1\underline{1}1\rangle\}$. In general, if a genotype has s heterozygous loci, it can have 2^{s-1} SNP-consistent haplotype solutions.

Haplotypes are passed down from an individual to its descendants. *Mendelian consistency* requires that, in the absence of *recombinations* or mutations, each child inherits one haplotype from one of the two haplotypes of the father and inherits the other haplotype from the mother similarly. This gives us more information to infer haplotypes when we are given a *pedigree*. The computational problem is therefore, given a pedigree with n individuals where each individual is associated with a genotype of length m, find an assignment of a pair of haplotypes to each individual such that SNP consistency



Haplotype Inference on Pedigrees Without Recombinations, Fig. 1 (**a**) Example of a pedigree with four nodes. (**b**) The graph G with 12 vertices, 6 *red edges*, and 4 *brown edges*. Each vector is a vertex in G. Vector pairs enclosed by *rounded rectangles* belong to the same individual

and Mendelian consistency are obeyed for each individual. In rare cases (especially for humans) [3], the pedigree may contain *mating loops*: a mating loop is formed when, for example, there is a marriage between descendants of a common ancestor.

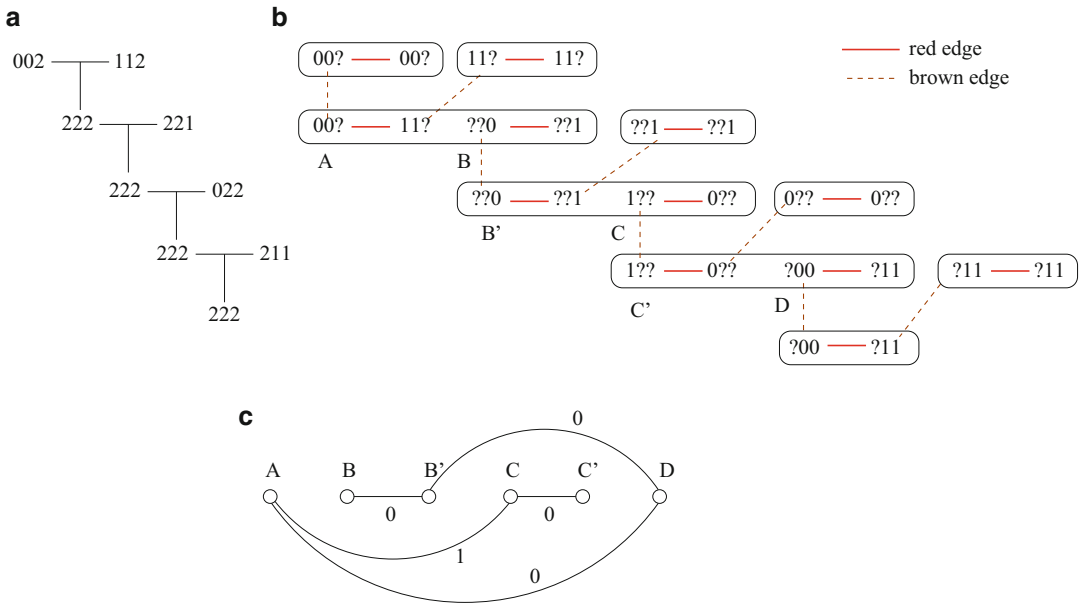
As a simple example, consider the pedigree in Fig. 1a for a family of four individuals and their genotypes. Due to SNP consistency, mother M's haplotypes must be $\langle 0000, 1000 \rangle$ (the order does not matter). Similarly, daughter D's haplotypes must be $\langle 1000, 1100 \rangle$. Now we apply Mendelian consistency to deduce that D must obtain the 1000 haplotype from M since neither of father F's haplotypes can be 1000 (considering locus 2). Therefore, D obtains 1100 from F, and F's haplotypes must be $\langle 0101, 1100 \rangle$. With F's and M's haplotypes known, the only solution for the haplotypes of son S that is consistent with his genotype 2202 is $\langle 0101, 1000 \rangle$.

Key Results

While this kind of deduction might appear to be enough to resolve all haplotype values, it is not the case. As we will shortly see, there are "long-distance" constraints that need to be considered. These constraints can be represented by a system of linear equations in GF(2) and solved using Gaussian elimination. This gives a $O(m^3n^3)$ time algorithm [3]. Subsequent papers try to capture or solve the constraints more economically. The time complexity was improved in [6] to $O(mn^2 + n^3 \log^2 n \log\log n)$ by eliminating redundant equations and using low-stretch spanning trees. A different approach was used in [1], representing the constraints by the parity of edge labels of some auxiliary graphs and finding solutions of these constraints using graph traversal without (directly) solving a system of linear equations. This gives a linear $O(mn)$ time algorithm, although it only works for the case with no mating loops and only produces one particular solution even when the pedigree admits more than one solution. Later algorithms include [4] which returns the full set of solutions in optimal time (again without mating loops) and [2] which can handle mating loops and runs in $O(kmn + k^2m)$ time where k is the number of mating loops.

In the following we sketch the idea behind the linear time algorithm in [1]. Each individual only has a pair of haplotypes, but the algorithm first produces a number of vector pairs for each individual, one vector pair for each trio (a father-mother-child triplet) that this individual belongs to. Each vector pair represents the information about the two haplotypes of this individual that can be derived by considering this trio only. These vector pairs will eventually be "unified" to become a single pair.

For the pedigree in Fig. 1a, the algorithm first produces the graph G in Fig. 1b, which has two connected components for the two trios F-M-S and F-M-D. The rule for enforcing SNP consistency (Mendelian consistency) is that the unresolved loci values, i.e., the ? values, must be different (same) at opposite ends of a red (brown) edge. There is only one way to unify the vector pairs of F consistently (due to locus 4): ?101 must correspond to 0101. We add an edge between these two vectors to represent the fact that they should be identical. Then all ? values



Haplotype Inference on Pedigrees Without Recombinations, Fig. 2 An example showing how constraints are represented by labeled edges in another graph. (**a**) The pedigree. (**b**) The local graph G. (**c**) The parity constraint graph J. Three constraints are added

can be resolved by traversing the now-connected graph and applying the aforementioned rules for enforcing consistency.

However, consider another pedigree in Fig. 2a. The previous steps can only produce Fig. 2b, which has four connected components and unresolved loci. We need to decide for A and B whether $A = 00?$ should connect to $B = ??0$ or its complement ??1 and similarly for B' and C, etc. Observe that a path between A and C must go through an odd number of red edges since locus 1 changes from 0 to 1. To capture this type of long-distance constraints, we construct a *parity constraint graph* J where the edge labels represent the parity constraints; see Fig. 2c. In effect, J represents a set of linear equations in GF(2); in Fig. 2c, the equations are $x_{AB}+x_{B'C} = 1, x_{B'C}+x_{C'D} = 0$, and $x_{AB}+x_{B'C}+x_{C'D} = 0$.

Finally, we can traverse J along the unique path between any two nodes; the parity of this path tells us how to merge the vector pairs in G. For example, the parity between A and B should be 0, indicating 00? in A should connect to ??0 in B (so both become 000), while the parity between B' and C is 1, so B' and C should be 000 and 111, respectively.

Cross-References

- Beyond Evolutionary Trees
- Musite: Tool for Predicting Protein Phosphorylation Sites
- Sequence and Spatial Motif Discovery in Short Sequence Fragments

Recommended Reading

1. Chan MY, Chan WT, Chin FYL, Fung SPY, Kao MY (2009) Linear-time haplotype inference on pedigrees without recombinations and mating loops. SIAM J Comput 38(6):2179–2197
2. Lai EY, Wang WB, Jiang T, Wu KP (2012) A linear-time algorithm for reconstructing zero-recombinant haplotype configuration on a pedigree. BMC Bioinformatics 13(S-17):S19
3. Li J, Jiang T (2003) Efficient inference of haplotypes from genotypes on a pedigree. J. Bioinformatics Comput Biol 1(1):41–69
4. Liu L and Jiang T (2010) A linear-time algorithm for reconstructing zero-recombinant haplotype configuration on pedigrees without mating loops. J Combin Optim 19:217–240
5. Russo E, Smaglik P (1999) Single nucleotide polymorphism: big pharmacy hedges its bets. The Scientist, 13(15):1, July 19, 1999

6. Xiao J, Liu L, Xia L, Jiang T (2009) Efficient algorithms for reconstructing zero-recombinant haplotypes on a pedigree based on fast elimination of redundant linear equations. SIAM J Comput 38(6):2198–2219

Hardness of Proper Learning

Vitaly Feldman
IBM Research – Almaden, San Jose, CA, USA

Keywords

DNF; function representation; NP-hardness of learning; PAC learning; Proper learning; representation-based hardness

Years and Authors of Summarized Original Work

1988; Pitt, Valiant

Problem Definition

The work of Pitt and Valiant [18] deals with learning Boolean functions in the Probably Approximately Correct (PAC) learning model introduced by Valiant [19]. A learning algorithm in Valiant's original model is given random examples of a function $f : \{0,1\}^n \to \{0,1\}$ from a representation class $\mathcal{F}$ and produces a hypothesis $h \in \mathcal{F}$ that closely approximates f. Here, a *representation class* is a set of functions and a language for describing the functions in the set. The authors give examples of natural representation classes that are NP-hard to learn in this model, whereas they can be learned if the learning algorithm is allowed to produce hypotheses from a richer representation class $\mathcal{H}$. Such an algorithm is said to learn $\mathcal{F}$ by $\mathcal{H}$; learning $\mathcal{F}$ by $\mathcal{F}$ is called *proper* learning.

The results of Pitt and Valiant were the first to demonstrate that the choice of representation of hypotheses can have a dramatic impact on the computational complexity of a learning problem. Their specific reductions from NP-hard problems are the basis of several other follow-up works on the hardness of proper learning [1,3,7].

Notation

Learning in the PAC model is based on the assumption that the unknown function (or *concept*) belongs to a certain class of concepts $\mathcal{C}$. In order to discuss algorithms that learn and output functions, one needs to define how these functions are represented. Informally, a representation for a concept class $\mathcal{C}$ is a way to describe concepts from $\mathcal{C}$ that defines a procedure to evaluate a concept in $\mathcal{C}$ on any input. For example, one can represent a conjunction of input variables by listing the variables in the conjunction. More formally, a representation class can be defined as follows.

Definition 1 A *representation class* $\mathcal{F}$ is a pair $(L, \mathcal{R})$ where

- L is a language over some fixed finite alphabet (e.g., $\{0,1\}$);
- $\mathcal{R}$ is an algorithm that for $\sigma \in L$, on input $(\sigma, 1^n)$ returns a Boolean circuit over $\{0,1\}^n$.

In the context of efficient learning, only efficient representations are considered, or, representations for which $\mathcal{R}$ is a polynomial-time algorithm. The concept class represented by $\mathcal{F}$ is the set of functions over $\{0,1\}^n$ defined by the circuits in $\{\mathcal{R}(\sigma, 1^n) \mid \sigma \in L\}$. For a Boolean function f, "$f \in \mathcal{F}$" means that f belongs to the concept class represented by $\mathcal{F}$ and that there is a $\sigma \in L$ whose associated Boolean circuit computes f. For most of the representations discussed in the context of learning, it is straightforward to construct a language L and the corresponding translating function $\mathcal{R}$, and therefore, they are not specified explicitly.

Associated with each representation is the complexity of describing a Boolean function using this representation. More formally, for a Boolean function $f \in \mathcal{C}$, $\mathcal{F}\text{-}\mathtt{size}(f)$ is the length of the shortest way to represent f using $\mathcal{F}$, or $\min\{|\sigma| \mid \sigma \in L, \mathcal{R}(\sigma, 1^n) \equiv f\}$.

We consider Valiant's PAC model of learning [19], as generalized by Pitt and Valiant [18].

In this model, for a function f and a distribution $\mathcal{D}$ over X, an *example oracle* $\mathrm{EX}(f, \mathcal{D})$ is an oracle that, when invoked, returns an example $\langle x, f(x)\rangle$, where x is chosen randomly with respect to $\mathcal{D}$, independently of any previous examples. For $\epsilon \geq 0$, we say that function g ϵ-approximates a function f with respect to distribution $\mathcal{D}$ if $\mathbf{Pr}_{\mathcal{D}}[f(x) \neq g(x)] \leq \epsilon$.

Definition 2 A representation class $\mathcal{F}$ is *PAC learnable* by representation class $\mathcal{H}$ if there exists an algorithm that for every $\epsilon > 0$, $\delta > 0$, n, $f \in \mathcal{F}$, and distribution $\mathcal{D}$ over X, given ϵ, δ, and access to $\mathrm{EX}(f, \mathcal{D})$, runs in time polynomial in $n, s = \mathcal{F}\text{-}\mathtt{size}(c), 1/\epsilon$ and $1/\delta$, and outputs, with probability at least $1-\delta$, a hypothesis $h \in \mathcal{H}$ that ϵ-approximates f.

A DNF expression is defined as an OR of ANDs of literals, where a *literal* is a possibly negated input variable. We refer to the ANDs of a DNF formula as its *terms*. Let $\mathsf{DNF}(k)$ denote the representation class of k-term DNF expressions. Similarly, a CNF expression is an OR of ANDs of literals. Let k-CNF denote the representation class of CNF expressions with each AND having at most k literals.

For a real-valued vector $c \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$, a *linear threshold function* (also called a *halfspace*) $T_{c,\theta}(x)$ is the function that equals 1 if and only if $\sum_{i \leq n} c_i x_i \geq \theta$. The representation class of Boolean threshold functions consists of all linear threshold functions with $c \in \{0,1\}^n$ and θ an integer.

Key Results

Theorem 1 ([18]) *For every $k \geq 2$, the representation class of $\mathsf{DNF}(k)$ is not properly learnable unless $\mathsf{RP} = \mathsf{NP}$.*

More specifically, Pitt and Valiant show that learning $\mathsf{DNF}(k)$ by $\mathsf{DNF}(\ell)$ is at least as hard as coloring a k-colorable graph using ℓ colors. For the case $k = 2$, they obtain the result by reducing from Set Splitting (see [9] for details on the problems). Theorem 1 is in sharp contrast with the fact that $\mathsf{DNF}(k)$ is learnable by k-CNF [19].

Theorem 2 ([18]) *The representation class of Boolean threshold functions is not properly learnable unless $\mathsf{RP} = \mathsf{NP}$.*

This result is obtained via a reduction from the NP-complete Zero-One Integer Programming problem (see [9] (p.245) for details on the problem). The result is contrasted by the fact that general linear thresholds are properly learnable [4].

These results show that using a specific representation of hypotheses forces the learning algorithm to solve a combinatorial problem that can be NP-hard. In most machine learning applications it is not important which representation of hypotheses is used as long as the value of the unknown function is predicted correctly. Therefore, learning in the PAC model is now defined without any restrictions on the output hypothesis (other than it being efficiently evaluatable). Hardness results in this setting are usually based on cryptographic assumptions (cf. [15]).

Hardness results for proper learning based on assumption $\mathsf{NP} \neq \mathsf{RP}$ are now known for several other representation classes and for other variants and extensions of the PAC learning model. Blum and Rivest show that for any $k \geq 3$, unions of k halfspaces are not properly learnable [3]. Hancock et al. prove that decision trees (cf. [16] for the definition of this representation) are not learnable by decision trees of somewhat larger size [11]. This result was strengthened by Alekhnovich et al. who also proved that intersections of two halfspaces are not learnable by intersections of k halfspaces for any constant k, general DNF expressions are not learnable by unions of halfspaces (and in particular are not properly learnable) and *k-juntas* are not properly learnable [1]. Further, DNF expressions remain NP-hard to learn properly even if *membership queries*, or the ability to query the unknown function at any point, are allowed [7]. Khot and Saket show that the problem of learning intersections of two halfspaces remains NP-hard even if a hypothesis with any constant error smaller than $1/2$ is required [17]. No efficient algorithms or hardness results are known for any of the above learning problems if no restriction is placed on the representation of hypotheses.

The choice of representation is important even in powerful learning models. Feldman proved that n^c-term DNF are not properly learnable for any constant c even when the distribution of examples is assumed to be uniform and membership queries are available [7]. This contrasts with Jackson's celebrated algorithm for learning DNF in this setting [13], which is not proper.

In the *agnostic learning* model of Haussler [12] and Kearns et al. [14], even the representation classes of conjunctions, decision lists, halfspaces, and parity functions are NP-hard to learn properly (cf. [2, 6, 8, 10] and references therein). Here again the status of these problems in the representation-independent setting is largely unknown.

Applications

A large number of practical algorithms use representations for which hardness results are known (most notably decision trees, halfspaces, and neural networks). Hardness of learning $\mathcal{F}$ by $\mathcal{H}$ implies that an algorithm that uses $\mathcal{H}$ to represent its hypotheses will not be able to learn $\mathcal{F}$ in the PAC sense. Therefore such hardness results elucidate the limitations of algorithms used in practice. In particular, the reduction from an NP-hard problem used to prove the hardness of learning $\mathcal{F}$ by $\mathcal{H}$ can be used to generate hard instances of the learning problem.

Open Problems

A number of problems related to proper learning in the PAC model and its extensions are open. Almost all hardness of proper learning results are for learning with respect to unrestricted distributions. For most of the problems mentioned in section "Key Results" it is unknown whether the result is true if the distribution is restricted to belong to some natural class of distributions (e.g., product distributions). It is unknown whether decision trees are learnable properly in the PAC model or in the PAC model with membership queries. This question is open even in the PAC model restricted to the uniform distribution only. Note that decision trees are learnable (non-properly) if membership queries are available [5] and are learnable properly in time $O(n^{\log s})$, where s is the number of leaves in the decision tree [1].

An even more interesting direction of research would be to obtain hardness results for learning by richer representation classes, such as AC^0 circuits, classes of neural networks and, ultimately, unrestricted circuits.

Cross-References

- Cryptographic Hardness of Learning
- Graph Coloring
- Learning DNF Formulas
- PAC Learning

Recommended Reading

1. Alekhnovich M, Braverman M, Feldman V, Klivans A, Pitassi T (2008) The complexity of properly learning simple classes. J Comput Syst Sci 74(1):16–34
2. Ben-David S, Eiron N, Long PM (2003) On the difficulty of approximately maximizing agreements. J Comput Syst Sci 66(3):496–514
3. Blum AL, Rivest RL (1992) Training a 3-node neural network is NP-complete. Neural Netw 5(1):117–127
4. Blumer A, Ehrenfeucht A, Haussler D, Warmuth M (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM 36(4):929–965
5. Bshouty N (1995) Exact learning via the monotone theory. Inform Comput 123(1):146–153
6. Feldman V, Gopalan P, Khot S, Ponuswami A (2009) On agnostic learning of parities, monomials and halfspaces. SIAM J Comput 39(2):606–645
7. Feldman V (2009) Hardness of approximate two-level logic minimization and pac learning with membership queries. J Comput Syst Sci 75(1):13–26
8. Feldman V, Guruswami V, Raghavendra P, Wu Y (2012) Agnostic learning of monomials by halfspaces is hard. SIAM J Comput 41(6):1558–1590
9. Garey M, Johnson DS (1979) Computers and intractability. W.H. Freeman, San Francisco
10. Guruswami V, Raghavendra P (2009) Hardness of learning halfspaces with noise. SIAM J Comput 39(2):742–765
11. Hancock T, Jiang T, Li M, Tromp J (1995) Lower bounds on learning decision lists and trees. In: Mayr

H

EW, Puech C (eds) STACS 95: 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, pp 527–538. Springer, Berlin/Heidelberg
12. Haussler D (1992) Decision theoretic generalizations of the PAC model for neural net and other learning applications. Inform Comput 100(1):78–150
13. Jackson J (1997) An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J Comput Syst Sci 55:414–440
14. Kearns M, Schapire R, Sellie L (1994) Toward efficient agnostic learning. Mach Learn 17(2–3):115–141
15. Kearns M, Valiant L (1994) Cryptographic limitations on learning boolean formulae and finite automata. J ACM 41(1):67–95
16. Kearns M, Vazirani U (1994) An introduction to computational learning theory. MIT, Cambridge
17. Khot S, Saket R (2011) On the hardness of learning intersections of two halfspaces. J Comput Syst Sci 77(1):129–141
18. Pitt L, Valiant L (1988) Computational limitations on learning from examples. J ACM 35(4):965–984
19. Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142

Harmonic Algorithm for Online Bin Packing

Leah Epstein
Department of Mathematics, University of Haifa, Haifa, Israel

Keywords

Bin packing; Bounded space algorithms; Competitive ratio

Years and Authors of Summarized Original Work

1985; Lee, Lee

Problem Definition

One of the goals of the design of the harmonic algorithm (or class of algorithms) was to provide an online algorithm for the classic bin packing problem that performs well with respect to the asymptotic competitive ratio, which is the standard measure for online algorithms for bin packing type problems. The competitive ratio for a given input is the ratio between the costs of the algorithm and of an optimal off-line solution. The asymptotic competitive ratio is the worst-case competitive ratio of inputs for which the optimal cost is sufficiently large. In the *online* (standard) bin packing problem, items of rational sizes in $(0, 1]$ are presented one by one. The algorithm must pack each item into a bin before the following item is presented. The total size of items packed into a bin cannot exceed 1, and the goal is to use the minimum number of bins, where a bin is used if at least one item was packed into it. All items must be packed, and the supply of bins is unlimited.

When an algorithm acts on an input, it can decide to *close* some of its bins and never use them again. A bin is called *closed* in such a case, while otherwise a used bin (which already has at least one item) is called *open*. The motivation for closing bins is to obtain fast running times per item (so that the algorithm will pack it into a bin selected out of a small number of options). Simple algorithms such as First Fit (FF), Best Fit (BF), and Worst Fit (WF) have worst-case running times of $O(\log N)$ per item, where N is the number of items at the time of assignment of the new item. On the other hand, the simple algorithm Next Fit (NF), which keeps at most of open bin and closes it when a new item cannot be packed there (before it uses a new bin for the new item), has a worst-case running time of $O(1)$ per item. Algorithms that keep a constant number of open bins are called *bounded space*. In many practical applications, this property is desirable, since the number of candidate bins for a new item is small and it does not increase with the input size.

Algorithm HARM_k (for an integer $k \geq 3$) was defined by Lee and Lee [7]. The fundamental and natural idea of "harmonic-based" algorithms is classify each item by size first (for online algorithms, the classification of an item must be done immediately upon arrival) and then pack it according to its class (instead of letting the exact size influence packing decisions). For the classifi-

cation of items, HARM_k splits the interval $(0, 1]$ into subintervals. There are $k-1$ subintervals of the form $(\frac{1}{i+1}, \frac{1}{i}]$ for $i = 1, \ldots, k-1$ and one final subinterval $(0, \frac{1}{k}]$. Each bin will contain only items from one subinterval (type). Every type is packed independently into its own bins using NF. Thus, there are at most $k-1$ open bins at each time (since for items of sizes above $\frac{1}{2}$, two items cannot share a bin, and any bin can be closed once it receives an item). Moreover, for $i < k$, as the items of type i have sizes no larger than $\frac{1}{i}$ but larger than $\frac{1}{i+1}$, every closed bin of this type will have exactly i items. For type k, a closed bin will contain at least k items, but it may contain many more items. This defines a class of algorithms (containing one algorithm for any $k \geq 3$). The term *the harmonic algorithm* (or simply HARM) refers to HARM_k for a sufficiently large value of k, and its asymptotic competitive ratio is the infimum value that can be achieved as the asymptotic competitive ratio of any algorithm of this class.

Key Results

It was shown in paper [7] that for k tending to infinity, the asymptotic ratio of HARM is a sum of series denoted by Π_∞ (see below), and it is equal to approximately 1.69103. Moreover, this is the best possible asymptotic competitive ratio of any online bounded space algorithm for standard bin packing.

The crucial item sizes are of the form $\frac{1}{\ell} + \varepsilon$, where $\varepsilon > 0$ is small and ℓ is an integer. These are items of type $\ell - 1$, and bins consisting of such items contain $\ell - 1$ items (except for the last bin used for this type that may contain a smaller number of items). However, a bin (of an off-line solution) that already contains an item of size $\frac{1}{2} + \varepsilon_1$ and an item of size $\frac{1}{3} + \varepsilon_2$ (for some small $\varepsilon_1, \varepsilon_2 > 0$) cannot contain also an item whose size is slightly above $\frac{1}{4}$. The largest item of this form would be slightly larger than $\frac{1}{7}$. Thus, the following sequence was defined [7]. Let $\pi_1 = 1$ and, for $j > 1$, $\pi_j = \pi_{j-1}(\pi_{j-1} + 1)$ (note that $\pi_{j'}$ is divisible by any π_j for $j < j'$). It turns out that the crucial item sizes are just above $\frac{1}{\pi_j + 1}$. The series $\sum_{j=1}^{\infty} \frac{1}{\pi_j}$ give the asymptotic competitive ratio of the HARM, Π_∞. For a long time the best lower bound on the asymptotic competitive ratio of (unbounded space) online algorithms was the one by van Vliet [8, 13], proved using this sequence (but the current best lower bound was proved using another set of inputs [1]).

In order to prove the upper bound Π_∞ on the competitive ratio, weights were used [12]. In this case weights are defined (for a specific value of k) quite easily such that all bins (except for the bins that remain open when the algorithm terminates) have total weights of at least 1. The weight of an item of type $i < k$ is $\frac{1}{i}$. The bins of type k are almost full for sufficiently large values of k (a bin can be closed only if the total size of its items exceeds $1 - \frac{1}{k}$). Assigning such an item a weight that is $\frac{k}{k-1}$ times its size will allow one to show that all bins except for a constant number of bins (at most $k-1$ bins) have total weights of at least 1. It is possible to show that the total weight of any packed bin is sufficiently close to Π_∞ for large values of k. As both HARM_k and an optimal solution pack the same items, the competitive ratio is implied. To show the upper bound on the total weight of any packed bin, it is required to show that the worst-case bin contains exactly one item of size just above $\frac{1}{\pi_j + 1}$ for $\pi_j \leq k - 1$ (and the remaining space can only contain items of type k). Roughly speaking, this holds as once it was proved that the bin contains the largest such items, the largest possible additional weight can be obtained only by adding the next such item.

Proving that no better bounded space algorithms exist can be done as follows. Let j' be a fixed integer. Let N be a large integer and consider a sequence containing N items of each size $\frac{1}{\pi_j} + \delta$ for a sufficiently small $\delta > 0$, for any $j = j', j' - 1, \cdots, 1$. If δ is chosen appropriately, we have $\sum_{j=1}^{j'} \frac{1}{\pi_j + 1} + j'\delta < 1$, so the items can be packed (off-line) into N bins. However, if items are presented in this order (sorted by nondecreasing size), after all items of one size have been presented, only a constant number of bins can receive larger items, and

thus the items of each size are packed almost independently.

Related Results

The space of a bounded space algorithm is the number of open bins that it can have. The space of NF is 1, while the space of harmonic algorithms increases with k. A bounded space algorithm with space 2 and the same asymptotic competitive ratio as FF and BF have been designed [3] (for comparison, $HARM_3$ has an asymptotic competitive ratio of $\frac{7}{4}$). A modification where smaller space is used to obtain the same competitive ratios of harmonic algorithms (or alternatively, smaller competitive ratios were obtained using the same space) was designed by Woeginger [15]. Thus, there exists another sequence of bounded space algorithms, with an increasing sequence of open bins, where their sequence of competitive ratios tends to Π_∞ such that the space required for every competitive ratio is much smaller than that of [7].

One drawback of the model above is that an off-line algorithm can rearrange the items and does not have to process them as a sequence. The variant where it must process them in the same order as an online algorithm was studied as well [2]. Algorithms that are based on partitioning into classes and have smaller asymptotic competitive ratios (but they are obviously not bounded space) were designed [7, 9, 11].

Generalizations have been studied too, in particular, bounded space bin packing with cardinality constraints (where an item cannot receive more than t items for a fixed integer $t \geq 2$) [5], parametric bin packing (where there is an upper bound strictly smaller than 1 on item sizes) [14], bin packing with rejection (where an item i has a rejection penalty r_i associated with it, and it can be either packed, or rejected for the cost r_i) [6], variable-sized bin packing (where bins of multiple sizes are available for packing) [10], and bin packing with resource augmentation (where the online algorithm can use bins of size $b > 1$ for a fixed rational number b, while an off-line algorithm still uses bins of size 1) [4]. In this last variant, the sequences of critical item sizes were redefined as a function of b, while variable-sized bin packing required a more careful partition into intervals.

Cross-References

▸ Current Champion for Online Bin Packing

Recommended Reading

1. Balogh J, Békési J, Galambos G (2012) New lower bounds for certain classes of bin packing algorithms. Theor Comput Sci 440–441:1–13
2. Chrobak M, Sgall J, Woeginger GJ (2011) Two-bounded-space bin packing revisited. In: Proceedings of the 19th annual European symposium on algorithms (ESA2011), Saarbrücken, Germany, pp 263–274
3. Csirik J, Johnson DS (2001) Bounded space on-line bin packing: best is better than first. Algorithmica 31:115–138
4. Csirik J, Woeginger GJ (2002) Resource augmentation for online bounded space bin packing. J Algorithms 44(2):308–320
5. Epstein L (2006) Online bin packing with cardinality constraints. SIAM J Discret Math 20(4):1015–1030
6. Epstein L (2010) Bin packing with rejection revisited. Algorithmica 56(4):505–528
7. Lee CC, Lee DT (1985) A simple online bin packing algorithm. J ACM 32(3):562–572
8. Liang FM (1980) A lower bound for on-line bin packing. Inf Process Lett 10(2):76–79
9. Ramanan P, Brown DJ, Lee CC, Lee DT (1989) On-line bin packing in linear time. J Algorithms 10:305–326
10. Seiden SS (2001) An optimal online algorithm for bounded space variable-sized bin packing. SIAM J Discret Math 14(4):458–470
11. Seiden SS (2002) On the online bin packing problem. J ACM 49(5):640–671
12. Ullman JD (1971) The performance of a memory allocation algorithm. Technical report 100, Princeton University, Princeton
13. van Vliet A (1992) An improved lower bound for online bin packing algorithms. Inf Process Lett 43(5):277–284
14. van Vliet A (1996) On the asymptotic worst case behavior of Harmonic Fit. J Algorithms 20(1):113–136
15. Woeginger GJ (1993) Improved space for bounded-space online bin packing. SIAM J Discret Math 6(4):575–581

Hierarchical Self-Assembly

David Doty
Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA, USA

Keywords

Hierarchical assembly; Intrinsic universality; Running time; Self-assembly; Verification

Years and Authors of Summarized Original Work

2005; Aggarwal, Cheng, Goldwasser, Kao, Espanes, Schweller
2012; Chen, Doty
2013; Cannon, Demaine, Demaine, Eisenstat, Patitz, Schweller, Summers, Winslow

Problem Definition

The general idea of *hierarchical* self-assembly (a.k.a., *multiple tile* [2], *polyomino* [8, 10], *two-handed* [3, 5, 6]) is to model self-assembly of tiles in which attachment of two multi-tile assemblies is allowed, as opposed to all attachments being that of a single tile onto a larger assembly. Several problems concern comparing hierarchical self-assembly to its single-tile-attachment variant (called the "seeded" model of self-assembly), so we define both models here. The model of hierarchical self-assembly was first defined (in a slightly different form that restricted the size of assemblies that could attach) by Aggarwal, Cheng, Goldwasser, Kao, Moisset de Espanes, and Schweller [2]. Several generalizations of the model exist that incorporated staged mixing of test tubes, "dissolvable" tiles, active signaling across tiles, etc., but here we restrict attention to the model closest to the seeded model of Winfree [9], different from that model only in the absence of a seed and the ability of two large assemblies to attach.

Supported by NSF grants CCF-1219274, CCF-1162589, and 1317694.

Definitions

A *tile type* is a unit square with four sides, each consisting of a *glue label* (often represented as a finite string) and a nonnegative integer *strength*. We assume a finite set T of tile types, but an infinite number of copies of each tile type, each copy referred to as a *tile*. An *assembly* is a positioning of tiles on the integer lattice $\mathbb{Z}^2$, i.e., a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$. We write $|\alpha|$ to denote $|\text{dom } \alpha|$. Write $\alpha \sqsubseteq \beta$ to denote that α is a *subassembly* of β, which means that $\text{dom } \alpha \subseteq \text{dom } \beta$ and $\alpha(p) = \beta(p)$ for all points $p \in \text{dom } \alpha$. We abuse notation and take a tile type t to be equivalent to the single-tile assembly containing only t (at the origin if not otherwise specified). Two adjacent tiles in an assembly *interact* if the glue labels on their abutting sides are equal and have positive strength. Each assembly induces a *binding graph*, a grid graph whose vertices are tiles, with an edge between two tiles if they interact. The assembly is τ*-stable* if every cut of its binding graph has strength at least τ, where the weight of an edge is the strength of the glue it represents. That is, the assembly is stable if at least energy τ is required to separate the assembly into two parts.

We now define both the seeded and hierarchical variants of the tile assembly model. A *seeded tile system* is a triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, $\sigma : \mathbb{Z}^2 \dashrightarrow T$ is a finite, τ-stable *seed assembly*, and τ is the *temperature*. If $\mathcal{T}$ has a single seed tile $s \in T$ (i.e., $\sigma(0, 0) = s$ for some $s \in T$ and is undefined elsewhere), then we write $\mathcal{T} = (T, s, \tau)$. Let $|\mathcal{T}|$ denote $|T|$. An assembly α is *producible* if either $\alpha = \sigma$ or if β is a producible assembly and α can be obtained from β by the stable binding of a single tile. In this case, write $\beta \to_1 \alpha$ (α is producible from β by the attachment of one tile), and write $\beta \to \alpha$ if $\beta \to_1^* \alpha$ (α is producible from β by the attachment of zero or more tiles). An assembly is *terminal* if no tile can be τ-stably attached to it.

A *hierarchical tile system* is a pair $\mathcal{T} = (T, \tau)$, where T is a finite set of tile types and $\tau \in \mathbb{N}$

is the temperature. An assembly is *producible* if either it is a single tile from T or it is the τ-stable result of translating two producible assemblies without overlap. Therefore, if an assembly α is producible, then it is produced via an *assembly tree*, a full binary tree whose root is labeled with α, whose $|\alpha|$ leaves are labeled with tile types, and each internal node is a producible assembly formed by the stable attachment of its two child assemblies. An assembly α is *terminal* if for every producible assembly β, α and β cannot be τ-stably attached. If α can grow into β by the attachment of zero or more assemblies, then we write $\alpha \to \beta$.

In either model, let $\mathcal{A}[\mathcal{T}]$ be the set of producible assemblies of $\mathcal{T}$, and let $\mathcal{A}_{\square}[\mathcal{T}] \subseteq \mathcal{A}[\mathcal{T}]$ be the set of producible, terminal assemblies of $\mathcal{T}$. A TAS $\mathcal{T}$ is *directed* (a.k.a., *deterministic*, *confluent*) if $|\mathcal{A}_{\square}[\mathcal{T}]| = 1$. If $\mathcal{T}$ is directed with unique producible terminal assembly α, we say that $\mathcal{T}$ *uniquely produces* α. It is easy to check that in the seeded aTAM, $\mathcal{T}$ uniquely produces α if and only if every producible assembly $\beta \sqsubseteq \alpha$. In the hierarchical model, a similar condition holds, although it is more complex since hierarchical assemblies, unlike seeded assemblies, do not have a "canonical translation" defined by the seed position. $\mathcal{T}$ uniquely produces α if and only if for every producible assembly β, there is a translation β' of β such that $\beta' \sqsubseteq \alpha$. In particular, if there is a producible assembly $\beta \neq \alpha$ such that dom α = dom β, then α is not uniquely produced. Since dom β = dom α, every nonzero translation of β has some tiled position outside of dom α, whence no such translation can be a subassembly of α, implying α is not uniquely produced.

Power of Hierarchical Assembly Compared to Seeded

One sense in which we can conclude that one model of computation M is at least as powerful as another model of computation M' is to show that any machine defined by M' can be "simulated efficiently" by a machine defined by M. In self-assembly, there is a natural definition of what it means for one tile system $\mathcal{S}$ to "simulate" another $\mathcal{T}$. We now discuss intuitively how to define such a notion. There are several intricacies to the full formal definition that are discussed in further detail in [3,5].

First, we require that there is a constant $k \in \mathbb{Z}^+$ (the "resolution loss") such that each tile type t in $\mathcal{T}$ is "represented" by one or more $k \times k$ blocks β of tiles in $\mathcal{S}$. In this case, we write $r(\beta) = t$, where $\beta : \{1, \ldots, k\}^2 \dashrightarrow S$ and S is the tile set of $\mathcal{S}$. Then β represents a $k \times k$ block of such tiles, possibly with empty positions at points $\mathbf{x}$ where $\beta(\mathbf{x})$ is undefined. We call such a $k \times k$ block in $\mathcal{S}$ a "macrotile." We can extend r to a function R that, given an assembly $\alpha_{\mathcal{S}}$ partitioned into $k \times k$ macrotiles, outputs an assembly $\alpha_{\mathcal{T}}$ of $\mathcal{T}$ such that, for each macrotile β of $\alpha_{\mathcal{S}}$, $r(\beta) = t$, where t is the tile type at the corresponding position in $\alpha_{\mathcal{T}}$.

Given such a representation function R indicating how to interpret assemblies of $\mathcal{S}$ as representing assemblies of $\mathcal{T}$, we now define what it means to say that $\mathcal{S}$ *simulates* $\mathcal{T}$. For each producible assembly $\alpha_{\mathcal{T}}$ of $\mathcal{T}$, there is a producible assembly $\alpha_{\mathcal{S}}$ of $\mathcal{S}$ such that $R(\alpha_{\mathcal{S}}) = \mathcal{T}$, and furthermore, for every producible assembly $\alpha_{\mathcal{S}}$, if $R(\alpha_{\mathcal{S}}) = \mathcal{T}$, then $\mathcal{T}$ is producible in $\mathcal{T}$. Finally, we require that R respects the "single attachment" dynamics of $\mathcal{T}$: there is a single tile that can be attached to $\alpha_{\mathcal{T}}$ to result in $\alpha'_{\mathcal{T}}$ if and only if there is some sequence of attachments to $\alpha_{\mathcal{S}}$ that results in assembly $\alpha'_{\mathcal{S}}$ such that $R(\alpha'_{\mathcal{S}}) = \alpha'_{\mathcal{T}}$.

With such an idea in mind, we can ask, "Is the hierarchical model at least as powerful as the seeded model?"

Problem 1 For every seeded tile system $\mathcal{T}$, design a hierarchical tile system $\mathcal{S}$ that simulates $\mathcal{T}$.

Another interpretation of a solution to Problem 1 is that, to the extent that the hierarchical model is more realistic than the seeded model by incorporating the reality that tiles may aggregate even in the absence of a seed, such a solution shows how to enforce seeded growth even in such an unfriendly environment that permits non-seeded growth.

Assembly Time

We now define time complexity for hierarchical systems (this definition first appeared in [4],

where it is explained in more detail). We treat each assembly as a single molecule. If two assemblies α and β can attach to create an assembly γ, then we model this as a chemical reaction $\alpha + \beta \to \gamma$, in which the rate constant is assumed to be equal for all reactions (and normalized to 1). In particular, if α and β can be attached in two different ways, this is modeled as two different reactions, even if both result in the same assembly.

At an intuitive level, the model we define can be explained as follows. We imagine dumping all tiles into solution at once, and at the same time, we grab one particular tile and dip it into the solution as well, pulling it out of the solution when it has assembled into a terminal assembly. Under the seeded model, the tile we grab will be a seed, assumed to be the only copy in solution (thus requiring that it appears only once in any terminal assembly). In the seeded model, no reactions occur other than the attachment of individual tiles to the assembly we are holding. In the hierarchical model, other reactions are allowed to occur in the background (we model this using the standard mass-action model of chemical kinetics [7]), but only those reactions with the assembly we are holding move it "closer" to completion. The other background reactions merely change concentrations of other assemblies (although these indirectly affect the time it will take our chosen assembly to complete, by changing the rate of reactions with our chosen assembly).

More formally, let $\mathcal{T} = (T, \tau)$ be a hierarchical TAS, and let $\rho : T \to [0, 1]$ be a concentrations function, giving the *initial* concentration of each tile type (we require that $\sum_{t \in T} \rho(t) = 1$, a condition known as the "finite density constraint"). Let $\mathbb{R}^+ = [0, \infty)$, and let $t \in \mathbb{R}^+$. For $\alpha \in \mathcal{A}[\mathcal{T}]$, let $[\alpha]_\rho(t)$ (abbreviated $[\alpha](t)$ when ρ is clear from context) denote the concentration of α at time t with respect to initial concentrations ρ, defined as follows. Given two assemblies α and β that can attach to form γ, we model this event as a chemical reaction $R : \alpha + \beta \to \gamma$. Say that a reaction $\alpha + \beta \to \gamma$ is *symmetric* if $\alpha = \beta$. Define the *propensity* (a.k.a., *reaction rate*) of R at time $t \in \mathbb{R}^+$ to be $\rho_R(t) = [\alpha](t) \cdot [\beta](t)$ if R is not symmetric and $\rho_R(t) = \frac{1}{2} \cdot [\alpha](t)^2$ if R is symmetric.

If α is consumed in reactions $\alpha + \beta_1 \to \gamma_1, \ldots, \alpha + \beta_n \to \gamma_n$ and produced in asymmetric reactions $\beta'_1 + \gamma'_1 \to \alpha, \ldots, \beta'_m + \gamma'_m \to \alpha$ and symmetric reactions $\beta''_1 + \beta''_1 \to \alpha, \ldots, \beta''_p + \beta''_p \to \alpha$, then the concentration $[\alpha](t)$ of α at time t is described by the differential equation:

$$\frac{d[\alpha](t)}{dt} = \sum_{i=1}^{m} [\beta'_i](t) \cdot [\gamma'_i](t) + \sum_{i=1}^{p} \frac{1}{2} \cdot [\beta''_i](t)^2 - \sum_{i=1}^{n} [\alpha](t) \cdot [\beta_i](t), \tag{1}$$

with boundary conditions $[\alpha](0) = \rho(r)$ if α is an assembly consisting of a single tile r and $[\alpha](0) = 0$ otherwise. In other words, the propensities of the various reactions involving α determine its rate of change, negatively if α is consumed and positively if α is produced.

This completes the definition of the dynamic evolution of concentrations of producible assemblies; it remains to define the time complexity of assembling a terminal assembly. Although we have distinguished between seeded and hierarchical systems, for the purpose of defining a model of time complexity in hierarchical systems and comparing them to the seeded system time complexity model of [1], it is convenient to introduce a seedlike "timekeeper tile" into the hierarchical system, in order to stochastically analyze the growth of this tile when it reacts in a solution that is itself evolving according to the continuous model described above. The seed does not have the purpose of nucleating growth but is introduced merely to focus attention on a single molecule that has not yet assembled anything, in order to ask how long it will take to assemble

into a terminal assembly. The choice of which tile type to pick will be a parameter of the definition, so that a system may have different assembly times depending on the choice of timekeeper tile.

Fix a copy of a tile type s to designate as a "timekeeper seed." The assembly of s into some terminal assembly $\hat{\alpha}$ is described as a time-dependent continuous-time Markov process in which each state represents a producible assembly containing s, and the initial state is the size-1 assembly with only s. For each state α representing a producible assembly with s at the origin, and for each pair of producible assemblies β, γ such that $\alpha + \beta \to \gamma$ (with the translation assumed to happen only to β so that α stays "fixed" in position), there is a transition in the Markov process from state α to state γ with transition rate $[\beta](t)$.

We define $\mathbf{T}_{\mathcal{T},\rho,s}$ to be the random variable representing the time taken for the copy of s to assemble into a terminal assembly via some sequence of reactions as defined above. We define the time complexity of a directed hierarchical TAS $\mathcal{T}$ with concentrations ρ and timekeeper s to be $\mathsf{T}(\mathcal{T}, \rho, s) = \mathrm{E}\left[\mathbf{T}_{\mathcal{T},\rho,s}\right]$.

For a shape $S \subset \mathbb{Z}^2$ (finite and connected), define the *diameter* of S to be $\mathrm{diam}(S) = \max_{\mathbf{u},\mathbf{v}\in S} \|\mathbf{u}-\mathbf{v}\|_1$, where $\|\mathbf{w}\|_1$ is the L_1 norm of $\mathbf{w}$.

Problem 2 Design a hierarchical tile system $\mathcal{T} = (T, \tau)$ such that every producible terminal assembly $\hat{\alpha}$ has the same shape S, and for some $s \in T$ and concentrations function $\rho : T \to [0, 1]$, $\mathsf{T}(\mathcal{T}, \rho, s) = o(\mathrm{diam}(S))$.

It is provably impossible to achieve this with the seeded model [1, 4], since all assemblies in that model require expected time at least proportional to their diameter.

Key Results

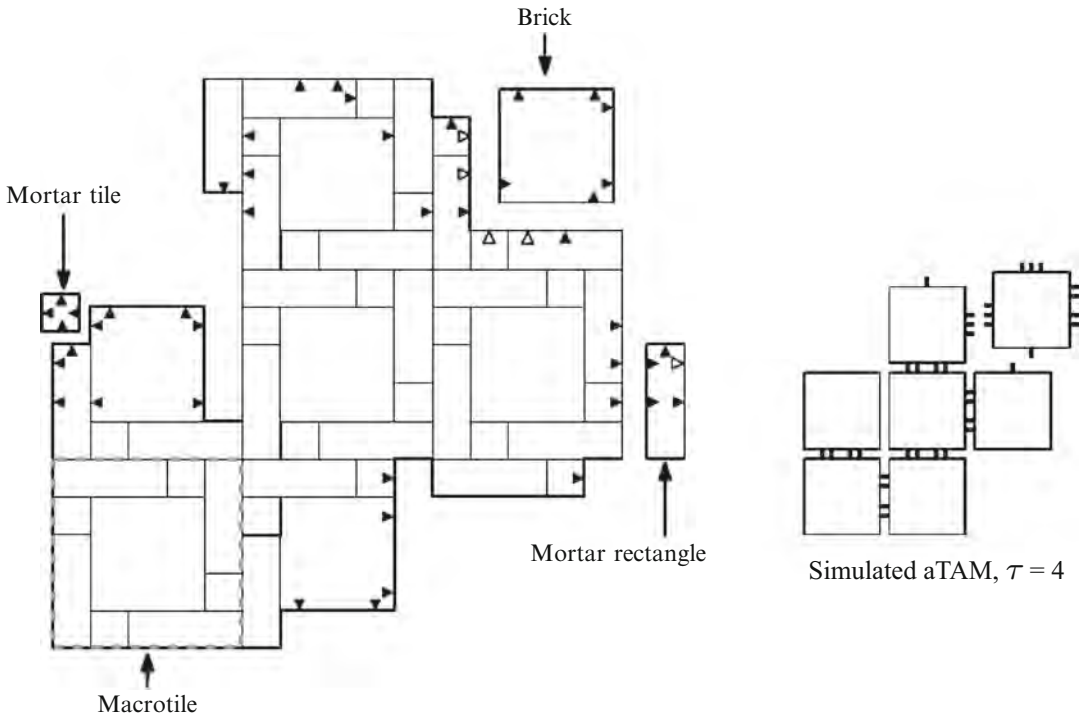
Power of Hierarchical Assembly Compared to Seeded

Cannon, Demaine, Demaine, Eisenstat, Patitz, Schweller, Summers, and Winslow [3] showed a solution to Problem 1. (They also showed several other ways in which the hierarchical model is more powerful than the seeded model, but we restrict attention to simulation here.) For the most part, temperature 2 seeded systems are as powerful as those at higher temperatures, but the simulation results of [3] hold for higher temperatures as well. In particular, they showed that every seeded temperature ≥ 4 tile system $\mathcal{T}$ can be simulated by a hierarchical temperature 4 tile system (as well as showing it is possible for temperature τ hierarchical tile systems to simulate temperature τ seeded tile systems for $\tau \in \{2, 3\}$, using similar logic to the higher-temperature construction). The definition of simulation has a parameter k indicating the resolution loss of the simulation. In fact, the simulation described in [3] requires only resolution loss $k = 5$.

Figure 1 shows an example of $\mathcal{S}$ simulating $\mathcal{T}$. The construction enforces the "simulation of dynamics" constraint that if and only if a single tile can attach in $\mathcal{T}$, and then a 5×5 macrotile representing it in $\mathcal{S}$ can assemble. It is critical that each tile type in $\mathcal{T}$ is represented by *more than one* type of macrotile in $\mathcal{S}$: each different type of macrotile represents a different subset of sides that can cooperate to allow the tile to bind. To achieve this, each macrotile consists of a central "brick" (itself a 3×3 block composed of 9 unique tile types with held together with strength-4 glues) surrounded by "mortar" (forming a ring around the central brick). Figure 1 shows "mortar rectangles" but, similarly to the brick, these are just 3×1 assemblies of 3 individual tile types with strength-4 glues. The logic of the system is such that if a brick B designed for a subset of cooperating sides $C \subseteq \{\mathsf{N}, \mathsf{S}, \mathsf{E}, \mathsf{W}\}$, then only if the mortar for all sides in C is present can B attach. Its attachment is required to fill in the remaining mortar representing the other sides in $\{\mathsf{N}, \mathsf{S}, \mathsf{E}, \mathsf{W}\} \setminus C$ that may not be present. Finally, those tiles enable the assembly of mortar in *adjacent* 5×5 blocks, to be ready for possible cooperation to bind bricks in those blocks.

Assembly Time

Chen and Doty [4] showed a solution to Problem 2, by proving that for infinitely many $n \in \mathbb{N}$, there is a (non-directed) hierarchical TAS



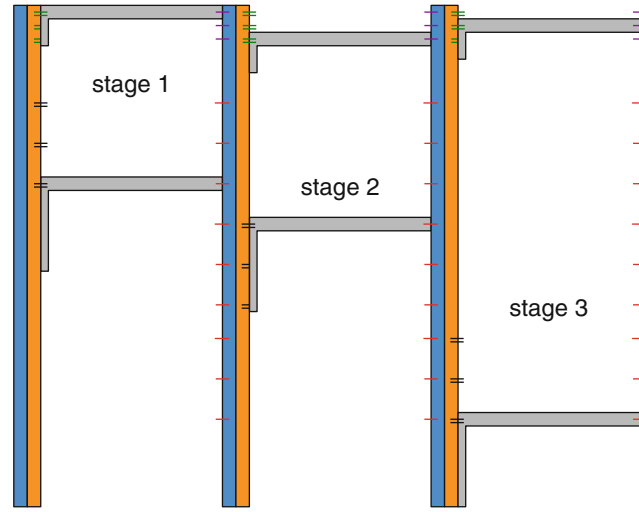
Hierarchical Self-Assembly, Fig. 1 Simulation of a seeded tile system $\mathcal{T}$ of temperature ≥ 4 by a hierarchical tile system $\mathcal{S}$ of temperature 4 (Figure taken from [3]). *Filled arrows* represent glues of strength 2, and *unfilled arrows* represent glues of strength 1. In the seeded tile system, the number of *dashes* on the side of a tile represent its strength

$\mathcal{T} = (T, 2)$ that strictly self-assembles an $n \times n'$ rectangle S, where $n' = o(n)$ (hence $\text{diam}(S) = \Theta(n)$), such that $|T| = O(\log n)$ and there is a tile type $s \in T$ and concentrations function $\rho : T \to [0, 1]$ such that $\mathsf{T}(\mathcal{T}, \rho, s) = O(n^{4/5} \log n)$.

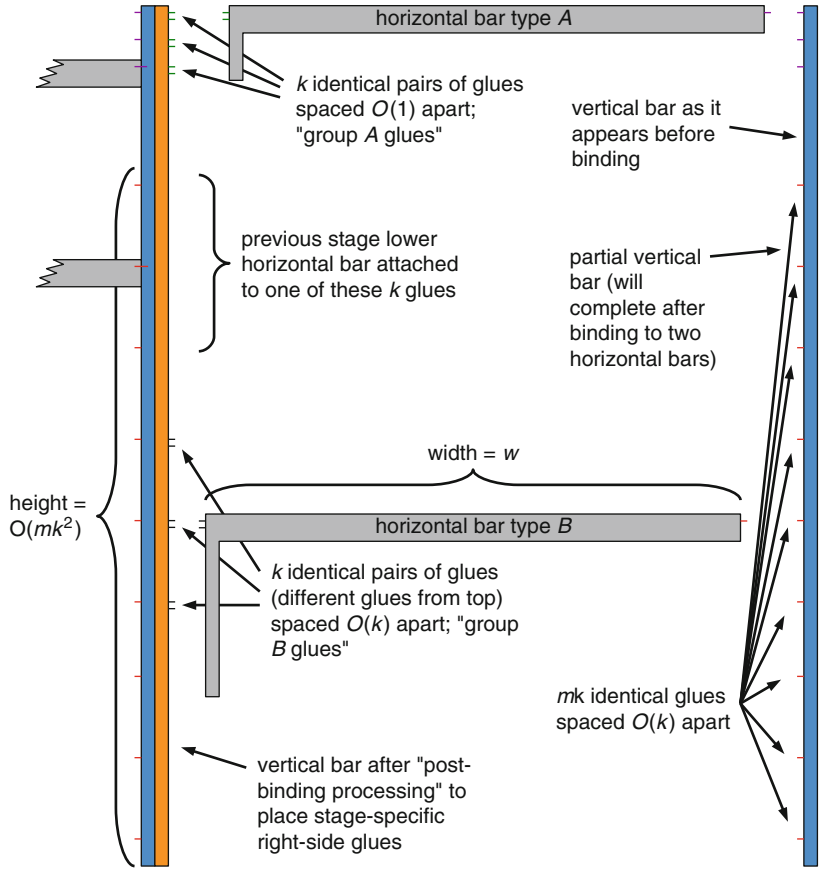
The construction consists of $m = n^{1/5}$ stages shown in Fig. 2, where each stage consists of the attachment of two "horizontal bars" to a single "vertical bar" as shown in Fig. 3. The vertical bar of the next stage then attaches to the right of the two horizontal bars, which cooperate to allow the binding because they each have a single strength 1 glue. All vertical bars are identical when they attach, but attachment triggers the growth of some tiles (shown in orange in Figs. 2 and 3) that make the attachment sites on the right side different from their locations in the previous stage, which is how the stages "count down" from m to 1.

The bars themselves are assembled in a "standard" way that requires time linear in the diameter of the bar, which is $w = n^{4/5}$ for a horizontal bar and $mk^2 = n^{3/5}$ (where k is a parameter that we set to be $n^{1/5}$) for a vertical bar. The speedup comes from the fact that each horizontal bar can attach to one of k different binding sites on a vertical bar, so the expected time for this to happen is factor k lower than if there were only a single binding site. The vertical "arm" on the left of each horizontal bar has the purpose of preventing any other horizontal bars from binding near it. Each stage also requires filler tiles to fill in the gap regions, but the time required for this is negligible compared to the time for all vertical and horizontal bars to attach.

Note that this construction is not directed: although every producible terminal assembly has the shape of an $n \times n'$ rectangle, there are many such terminal assemblies. Chen and Doty [4] also showed that for a class of directed systems called "partial order tile systems," no solution to Problem 2 exists: provably any such tile system



Hierarchical Self-Assembly, Fig. 2 High-level overview of interaction of "vertical bars" and "horizontal bars" to create the rectangle in the solution to Problem 2 that assembles in time sublinear in its diameter. Filler tiles fill in the empty regions. If glues overlap two regions then represent a formed bond. If glues overlap one region but not another, they are glues from the former region but are mismatched (and thus "covered and protected") by the latter region



Hierarchical Self-Assembly, Fig. 3 "Vertical bars" for the construction of a fast-assembling square, and their interaction with horizontal bars, as shown for a single stage of Fig. 2. "Type B" horizontal bars have a longer vertical arm than "Type A" since the glues they must block are farther apart

assembling a shape of diameter d requires expected time $\Omega(d)$.

Open Problems

It is known [2] that the tile complexity of assembling an $n \times k$ rectangle in the seeded aTAM, if $k < \frac{\log n}{\log\log n - \log\log\log n}$, is asymptotically lower bounded by $\Omega\left(\frac{n^{1/k}}{k}\right)$ and upper bounded by $O(n^{1/k})$. For the hierarchical model, the upper bound holds as well [2], but the strongest known lower bound is the information-theoretic $\Omega\left(\frac{\log n}{\log\log n}\right)$.

Question 1 What is the tile complexity of assembling an $n \times k$ rectangle in the hierarchical model, when $k < \frac{\log n}{\log\log n - \log\log\log n}$?

Cross-References

- Experimental Implementation of Tile Assembly
- Patterned Self-Assembly Tile Set Synthesis
- Robustness in Self-Assembly
- Self-Assembly at Temperature 1
- Self-Assembly of Fractals
- Self-Assembly with General Shaped Tiles
- Staged Assembly
- Temperature Programming in Self-Assembly

Recommended Reading

1. Adleman LM, Cheng Q, Goel A, Huang M-D (2001) Running time and program size for self-assembled squares. In: STOC 2001: proceedings of the thirty-third annual ACM symposium on theory of computing, Hersonissos. ACM, pp 740–748
2. Aggarwal G, Cheng Q, Goldwasser MH, Kao M-Y, Moisset de Espanés P, Schweller RT (2005) Complexities for generalized models of self-assembly. SIAM J Comput 34:1493–1515. Preliminary version appeared in SODA 2004
3. Cannon S, Demaine ED, Demaine ML, Eisenstat S, Patitz MJ, Schweller RT, Summers SM, Winslow A (2013) Two hands are better than one (up to constant factors). In: STACS 2013: proceedings of the thirtieth international symposium on theoretical aspects of computer science, Kiel, pp 172–184
4. Chen H-L, Doty D (2012) Parallelism and time in hierarchical self-assembly. In: SODA 2012: proceedings of the 23rd annual ACM-SIAM symposium on discrete algorithms, Kyoto, pp 1163–1182
5. Demaine ED, Patitz MJ, Rogers T, Schweller RT, Summers SM, Woods D (2013) The two-handed tile assembly model is not intrinsically universal. In: ICALP 2013: proceedings of the 40th international colloquium on automata, languages and programming, Riga, July 2013
6. Doty D, Patitz MJ, Reishus D, Schweller RT, Summers SM (2010) Strong fault-tolerance for self-assembly with fuzzy temperature. In: FOCS 2010: proceedings of the 51st annual IEEE symposium on foundations of computer science, Las Vegas, pp 417–426
7. Epstein IR, Pojman JA (1998) An introduction to nonlinear chemical dynamics: oscillations, waves, patterns, and chaos. Oxford University Press, Oxford
8. Luhrs C (2010) Polyomino-safe DNA self-assembly via block replacement. Nat Comput 9(1):97–109. Preliminary version appeared in DNA 2008
9. Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology, June 1998
10. Winfree E (2006) Self-healing tile sets. In: Chen J, Jonoska N, Rozenberg G (eds) Nanotechnology: science and computation. Natural computing series. Springer, Berlin/New York, pp 55–78

Hierarchical Space Decompositions for Low-Density Scenes

Mark de Berg
Department of Mathematics and Computer Science, TU Eindhoven, Eindhoven, The Netherlands

Keywords

Binary space partitions; Compressed quadtrees; Computational geometry; Hierarchical space decompositions; Realistic input models

Years and Authors of Summarized Original Work

2000; De Berg
2010; De Berg, Haverkort, Thite, Toma

Problem Definition

Many algorithmic problems on spatial data can be solved efficiently if a suitable decomposition of the ambient space is available. Two desirable properties of the decomposition are that its cells have a nice shape – convex and/or of constant complexity – and that each cell intersects only a few objects from the given data set. Another desirable property is that the decomposition is hierarchical, meaning that the space is partitioned in a recursive manner. Popular hierarchical space decompositions include quadtrees and binary space partitions.

When the objects in the given data set are nonpoint objects, they can be fragmented by the partitioning process. This fragmentation has a negative impact on the storage requirements of the decomposition and on the efficiency of algorithms operating on it. Hence, it is desirable to minimize fragmentation. In this chapter, we describe methods to construct linear-size compressed quadtrees and binary space partitions for so-called low-density sets. To simplify the presentation, we describe the constructions in the plane. We use S to denote the set of n objects for which we want to construct a space decomposition and assume for simplicity that the objects in S are disjoint, convex, and of nonzero area.

Binary Space Partitions

A *binary space partition* for a set S of n objects in the plane is a recursive decomposition of the plane by lines, typically such that each cell in the final decomposition intersects only a few objects from S. The tree structure modeling this decomposition is called a *binary space partition tree*, or BSP *tree* for short – see Fig. 1 for an illustration. Thus, a BSP tree $\mathcal{T}$ for S can be defined as follows.

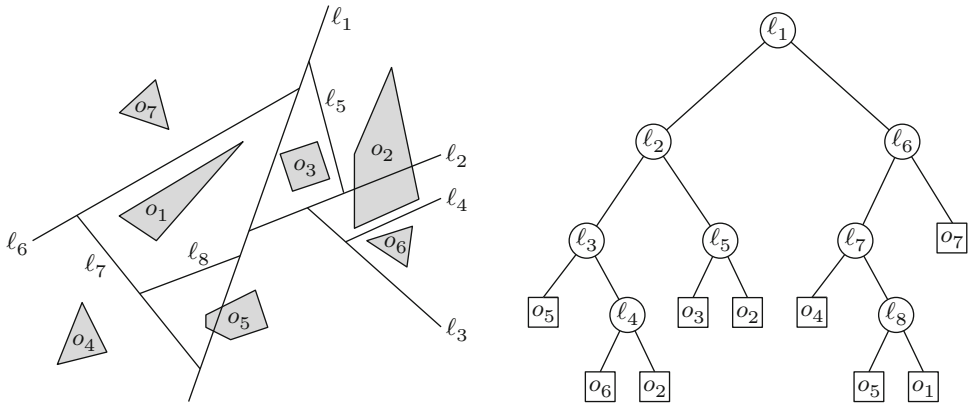
- If a predefined stopping criterion is met – often this is when $|S|$ is sufficiently small – then $\mathcal{T}$ consists of a single leaf where the set S is stored.
- Otherwise the root node v of $\mathcal{T}$ stores a suitably chosen *splitting line* ℓ. Let ℓ^- and ℓ^+ denote the half-planes lying to the left and to the right of ℓ, respectively (or, if ℓ is horizontal, below and above ℓ).
  - The left subtree of v is a BSP tree for $S^- := \{o \cap \ell^- : o \in S\}$, the set of object fragments lying in the half-plane ℓ^-.
  - The right subtree of v is a BSP tree for $S^+ := \{o \cap \ell^+ : o \in S\}$, the set of object fragments lying in the half-plane ℓ^+.

The *size* of a BSP tree is the total number of object fragments stored in the tree.

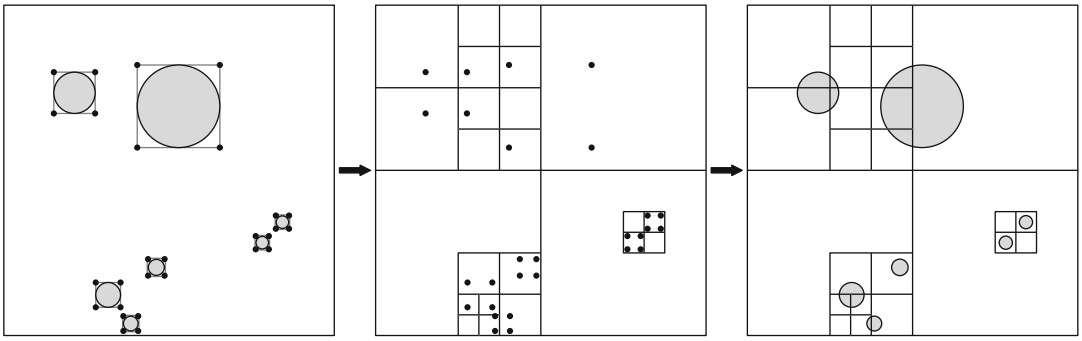
Compressed Quadtrees

Let $U = [0, 1]^2$ be the unit square. We say that a square $\sigma \subseteq U$ is a *canonical square* if there is an integer $k \geqslant 0$ such that σ is a cell of the regular subdivision of U into $2^k \times 2^k$ squares. A *donut* is the set-theoretic difference $\sigma_{\text{out}} \setminus \sigma_{\text{in}}$ of a canonical square σ_{out} and a canonical square $\sigma_{\text{in}} \subset \sigma_{\text{out}}$. A *compressed quadtree* $\mathcal{T}$ for a set P of points inside a canonical square σ defined as follows; see also Fig. 2 (middle).

- If a predefined stopping criterion is met – usually this is when $|P|$ is sufficiently small – then $\mathcal{T}$ consists of a single leaf storing the set P.
- If the stopping criterion is not met, then $\mathcal{T}$ is defined as follows. Let σ_{NE} denote the northeast quadrant of σ and let $P_{\text{NE}} := P \cap \sigma_{\text{NE}}$. Define σ_{SE}, σ_{SW}, σ_{NW} and P_{SE}, P_{SW}, P_{NW} similarly for the other three quadrants. (Here we should make sure that points on the boundary between quadrants are assigned to quadrants in a consistent manner.) Now $\mathcal{T}$ consists of a root node v with four or two children, depending on how many of the sets P_{NE}, P_{SE}, P_{SW}, P_{NW} are nonempty:
  - If at least two of the sets P_{NE}, P_{SE}, P_{SW}, P_{NW} are nonempty, then v has four children v_{NE}, v_{SE}, v_{SW}, v_{NW}. The child v_{NE} is the root of a compressed quadtree for the set P_{NE} inside the square σ_{NW}; the other three children are defined similarly for the point sets inside the other quadrants.
  - If only one of P_{NE}, P_{SE}, P_{SW}, P_{NW} is nonempty, then v has two children v_{in} and v_{out}. The child v_{in} is the root of a



Hierarchical Space Decompositions for Low-Density Scenes, Fig. 1 A binary space partition for a set of polygons (*left*) and the corresponding BSP tree (*right*)



Hierarchical Space Decompositions for Low-Density Scenes, Fig. 2 Construction of a compressed quadtree for a set of disks: take the bounding-box vertices (*left*), construct a compressed quadtree for the vertices (*middle*), and put the disks back in (*right*)

compressed quadtree for P inside σ_{in}, where σ_{in} is the smallest canonical square containing all points from P. The other child is a leaf corresponding to the donut $\sigma \setminus \sigma_{\text{in}}$.

A compressed quadtree for a set of n points has size $O(n)$.

Above we defined compressed quadtrees for point sets. In this chapter, we are interested in compressed quadtrees for nonpoint objects. These are defined similarly: each internal node corresponds to a canonical square, and each leaf is a canonical square or a donut. This time donuts need not be empty, but may intersect objects (although not too many). The right picture in Fig. 2 shows a compressed quadtree for a set of disks. The *size* of a compressed quadtree for a set of nonpoint objects is defined as the total number of object fragments stored in the tree. Because nonpoint objects may be split into fragments during the subdivision process, a compressed quadtree for nonpoint objects is not guaranteed to have linear size.

Low-Density Scenes

The main question we are interested in is the following: given a set S of n objects, can we construct a compressed quadtree or BSP tree with $O(n)$ leaves such that each leaf region intersects $O(1)$ objects? In general, the answer to this question is no. For compressed quadtrees, this can be seen by considering a set S of slanted parallel segments that are very close

together. A linear-size BSP tree cannot be guaranteed either: there are sets of n disjoint segments in the plane for which any BSP tree has size $\Omega(n \log n / \log\log n)$ [7]. In $\mathbb{R}^3$ the situation is even worse: there are sets of n disjoint triangles for which any BSP tree has size $\Omega(n^2)$ [5]. (Both bounds are tight: there are algorithms that guarantee a BSP tree of size $O(n \log n / \log\log n)$ in the plane [8] and of size $O(n^2)$ in $\mathbb{R}^3$ [6].) Fortunately, in practice, the objects for which we want to construct a space decomposition are often distributed nicely, which allows us to construct much smaller decompositions than for the worst-case examples mentioned above. To formalize this, we define the concept of *density* of a set of objects in $\mathbb{R}^d$.

Definition 1 The *density* of a set S of objects in $\mathbb{R}^d$, denoted density(S), is defined as the smallest number λ such that the following holds: any ball $b \subset \mathbb{R}^d$ intersects at most λ objects $o \in S$ such that $\text{diam}(o) \geqslant \text{diam}(b)$, where diam$(\cdot)$ denotes the diameter of an object.

As illustrated in Fig. 3(i), a set of n parallel segments can have density n if the segments are very close together. In most practical situations, however, the input objects are distributed nicely and the density will be small. For many classes of objects, one can even prove that the density is $O(1)$. For example, a set of disjoint disks in the plane has density at most 5. More generally, any set of disjoint objects that are *fat* – examples of fat objects are disks, squares, triangles whose minimum angle is lower bounded – has density $O(1)$ [3]. The main question now is: Is low density sufficient to guarantee a hierarchical space decomposition of linear size? The answer is yes, and constructing the space decomposition is surprisingly easy.

Key Results

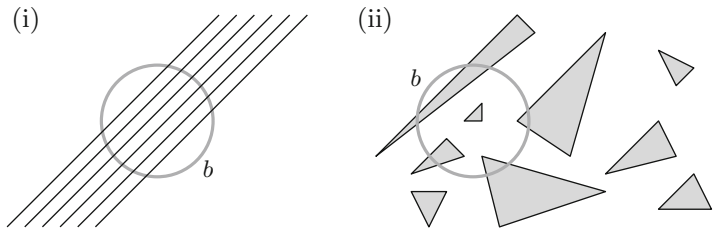
The construction of space decompositions for low-density sets is based on the following lemma. In the lemma, the square σ is considered to be open, that is, σ does not include its boundary. Let bb(o) denote the axis-aligned bounding box of an object o.

Lemma 1 *Let S be a set of n objects in the plane and let B_S denote the set of $4n$ vertices of the bounding boxes* bb(o) *of the objects $o \in S$. Let σ be any square region in the plane. Then the number of objects in S intersecting σ is at most $k + 4\lambda$, where k is the number of bounding-box vertices inside σ and $\lambda :=$* density(S).

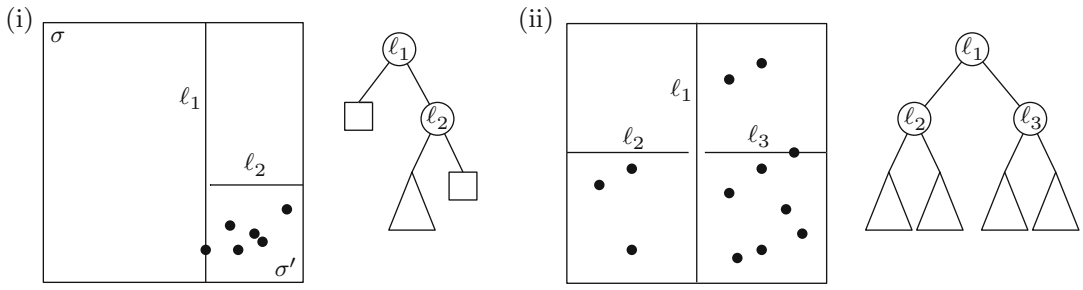
With Lemma 1 in hand, it is surprisingly simple to construct BSP trees or compressed quadtrees of small size for any given set S whose density is small.

Binary Space Partitions

For BSP trees we proceed as follows. Let B_S be the set of vertices of the bounding boxes of the objects in S. In a generic step in the recursive construction of $\mathcal{T}$, we are given a square σ and the set of points $B_S(\sigma) := B_S \cap \sigma$. Initially σ is a square containing all points of B_S. When $B_S = \emptyset$, then $\mathcal{T}$ consists of a single leaf and the



Hierarchical Space Decompositions for Low-Density Scenes, Fig. 3 (**i**) The ball b intersects all n segments and the segments have diameter larger than diam(b), so the density of the set of segments is n. (**ii**) Any ball b, no matter where it is placed or what its size is, intersects at most three triangles with diameter at least diam(b), so the density of the set of triangles is 3



Hierarchical Space Decompositions for Low-Density Scenes, Fig. 4 Two cases in the construction of the BSP tree

recursion ends; otherwise we proceed as follows. Let σ_{NE}, σ_{SE}, σ_{SW}, and σ_{NW} denote the four quadrants of σ. We now have two cases, illustrated in Fig. 4.

Case (i): all points in $B_S(\sigma)$ lie in the same quadrant. Let σ' be the smallest square sharing a corner with σ and containing all points from $B_S(\sigma)$ in its interior or on its boundary. Split σ into three regions using a vertical and a horizontal splitting line such that σ' is one of those regions; see Fig. 4(i). Recursively construct a BSP tree for the square σ' with respect to the set $B_S(\sigma')$ of points lying in the interior of σ'.

Case (ii): not all points in $B_S(\sigma)$ lie in the same quadrant. Split σ into four quadrants using a vertical and two horizontal splitting lines; see Fig. 4(ii). Recursively construct a BSP tree for each quadrant with respect to the points lying in its interior.

The construction produces a subdivision of the initial square into $O(n)$ leaf regions, which are squares or rectangles and which do not contain points from B_S in their interior. Using Lemma 1, one can argue that each leaf region intersects $O(\lambda)$ objects.

Compressed Quadtrees

The construction of a compressed quadtree for a low-density set S is also based on the set B_S of bounding-box vertices: we construct a compressed quadtree for B_S, where we stop the recursive construction when a square contains bounding-box vertices from at most one object in S or when all bounding-box vertices inside the square coincide. Figure 2 illustrates the process. The resulting compressed quadtree has $O(n)$ leaf regions, which are canonical squares or donuts. Again using Lemma 1, one can argue that each leaf region intersects $O(\lambda)$ objects.

Improvements and Generalizations

The constructions above guarantee that each region in the space decomposition is intersected by $O(\lambda)$ objects and that the number of regions is $O(n)$. Hence, the total number of the object fragments is $O(\lambda n)$. The main idea behind the introduction of the density λ is that in practice λ is often a small constant. Nevertheless, it is (at least from a theoretical point of view) desirable to get rid of the dependency on λ in the number of fragments. This is possible by reducing the number of regions in the decomposition to (n/λ). To this end, we allow leaf regions to contain up to $O(\lambda)$ bounding-box vertices. Note that Lemma 1 implies that a square with $O(\lambda)$ bounding-box vertices inside intersects $O(\lambda)$ objects. If implemented correctly, this idea leads to decompositions with $O(n/\lambda)$ regions each of which intersects $O(\lambda)$ objects, both for binary space partitions [2, Section 12.5] and for compressed quadtrees [4]. The results can also be generalized to higher dimensions, giving the following theorem.

Theorem 1 *Let S be a set of n objects in $\mathbb{R}^d$ and let $\lambda :=$ density(S). There is a binary space partition for S consisting of $O(n/\lambda)$ leaf regions, each intersecting $O(\lambda)$ objects. Similarly, there is a compressed quadtree with $O(n/\lambda)$ leaf regions, each intersecting $O(\lambda)$ objects.*

Cross-References

- ▸ Binary Space Partitions
- ▸ Quadtrees and Morton Indexing

Recommended Reading

1. De Berg M (2000) Linear size binary space partitions for uncluttered scenes. Algorithmica 28(3):353–366
2. De Berg M, Cheong O, Van Kreveld M, Overmars M (2008) Computational geometry: algorithms and applications, 3rd edn. Springer, Berlin/Heidelberg
3. De Berg M, Katz M, Van der Stappen AF, Vleugels J (2002) Realistic input models for geometric algorithms. Algorithmica 34(1):81–97
4. De Berg M, Haverkort H, Thite S, Toma L (2010) Star-quadtrees and guard-quadtrees: I/O-efficient indexes for fat triangulations and low-density planar subdivisions. Comput Geom Theory Appl 43:493–513
5. Chazelle B (1984) Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. SIAM J Comput 13(3):488–507
6. Paterson MS and Yao FF (1990) Efficient binary space partitions for hidden-surface removal and solid modeling. Discret Comput Geom 5(5):485–503
7. Tóth CD (2003) A note on binary plane partitions. Discret Comput Geom 30(1):3–16
8. Tóth CD (2011) Binary plane partitions for disjoint line segments. Discret Comput Geom 45(4):617–646

High Performance Algorithm Engineering for Large-Scale Problems

David A. Bader
College of Computing, Georgia Institute of Technology, Atlanta, GA, USA

Keywords

Experimental algorithmics

Years and Authors of Summarized Original Work

2005; Bader

Problem Definition

Algorithm engineering refers to the process required to transform a pencil-and-paper algorithm into a robust, efficient, well tested, and easily usable implementation. Thus it encompasses a number of topics, from modeling cache behavior to the principles of good software engineering; its main focus, however, is experimentation. In that sense, it may be viewed as a recent outgrowth of *Experimental Algorithmics* [14], which is specifically devoted to the development of methods, tools, and practices for assessing and refining algorithms through experimentation. The *ACM Journal of Experimental Algorithmics (JEA)*, at URL www.jea.acm.org, is devoted to this area.

High-performance algorithm engineering [2] focuses on one of the many facets of algorithm engineering: speed. The high-performance aspect does not immediately imply parallelism; in fact, in any highly parallel task, most of the impact of high-performance algorithm engineering tends to come from refining the serial part of the code.

The term *algorithm engineering* was first used with specificity in 1997, with the organization of the first *Workshop on Algorithm Engineering (WAE 97)*. Since then, this workshop has taken place every summer in Europe. The 1998 *Workshop on Algorithms and Experiments (ALEX98)* was held in Italy and provided a discussion forum for researchers and practitioners interested in the design, analyzes and experimental testing of exact and heuristic algorithms. A sibling workshop was started in the Unites States in 1999, the *Workshop on Algorithm Engineering and Experiments (ALENEX99)*, which has taken place every winter, colocated with the *ACM/SIAM Symposium on Discrete Algorithms (SODA)*.

Key Results

Parallel computing has two closely related main uses. First, with more memory and storage resources than available on a single workstation, a parallel computer can solve correspondingly larger instances of the same problems. This

increase in size can translate into running higher-fidelity simulations, handling higher volumes of information in data-intensive applications, and answering larger numbers of queries and datamining requests in corporate databases. Secondly, with more processors and larger aggregate memory subsystems than available on a single workstation, a parallel computer can often solve problems faster. This increase in speed can also translate into all of the advantages listed above, but perhaps its crucial advantage is in turnaround time. When the computation is part of a real-time system, such as weather forecasting, financial investment decision-making, or tracking and guidance systems, turnaround time is obviously the critical issue. A less obvious benefit of shortened turnaround time is higher-quality work: when a computational experiment takes less than an hour, the researcher can afford the luxury of exploration – running several different scenarios in order to gain a better understanding of the phenomena being studied.

In algorithm engineering, the aim is to present repeatable results through experiments that apply to a broader class of computers than the specific make of computer system used during the experiment. For sequential computing, empirical results are often fairly machine-independent. While machine characteristics such as word size, cache and main memory sizes, and processor and bus speeds differ, comparisons across different uniprocessor machines show the same trends. In particular, the number of memory accesses and processor operations remains fairly constant (or within a small constant factor). In high-performance algorithm engineering with parallel computers, on the other hand, this portability is usually absent: each machine and environment is its own special case. One obvious reason is major differences in hardware that affect the balance of communication and computation costs – a true shared-memory machine exhibits very different behavior from that of a cluster based on commodity networks.

Another reason is that the communication libraries and parallel programming environments (e.g., MPI [12], OpenMP [16], and High-Performance Fortran [10]), as well as the parallel algorithm packages (e.g., fast Fourier transforms using FFTW [6] or parallelized linear algebra routines in ScaLAPACK [4]), often exhibit differing performance on different types of parallel platforms. When multiple library packages exist for the same task, a user may observe different running times for each library version even on the same platform. Thus a running-time analysis should clearly separate the time spent in the user code from that spent in various library calls. Indeed, if particular library calls contribute significantly to the running time, the number of such calls and running time for each call should be recorded and used in the analysis, thereby helping library developers focus on the most cost-effective improvements. For example, in a simple message-passing program, one can characterize the work done by keeping track of sequential work, communication volume, and number of communications. A more general program using the collective communication routines of MPI could also count the number of calls to these routines. Several packages are available to instrument MPI codes in order to capture such data (e.g., MPICH's nupshot [8], Pablo [17], and Vampir [15]). The SKaMPI benchmark [18] allows running-time predictions based on such measurements even if the target machine is not available for program development. SKaMPI was designed for robustness, accuracy, portability, and efficiency; For example, SKaMPI adaptively controls how often measurements are repeated, adaptively refines message-length and step-width at "interesting" points, recovers from crashes, and automatically generates reports.

Applications

The following are several examples of algorithm engineering studies for high-performance and parallelcomputing.

1. Bader's prior publications (see [2] and http://www.cc.gatech.edu/~bader) contain many empirical studies of parallel algorithms for

combinatorial problems like sorting, selection, graph algorithms, and image processing.

2. In a recent demonstration of the power of high-performance algorithm engineering, a million-fold speed-up was achieved through a combination of a 2,000-fold speedup in the serial execution of the code and a 512-fold speedup due to parallelism (a speed-up, however, that will scale to any number of processors) [13]. (In a further demonstration of algorithm engineering, additional refinements in the search and bounding strategies have added another speedup to the serial part of about 1,000, for an overall speedup in excess of 2 billion)
3. JáJá and Helman conducted empirical studies for prefix computations, sorting, and list-ranking, on symmetric multiprocessors. The sorting research (see [9]) extends Vitter's external Parallel Disk Model to the internal memory hierarchy of SMPs and uses this new computational model to analyze a general-purpose sample sort that operates efficiently in shared-memory. The performance evaluation uses nine well-defined benchmarks. The benchmarks include input distributions commonly used for sorting benchmarks (such as keys selected uniformly and at random), but also benchmarks designed to challenge the implementation through load imbalance and memory contention and to circumvent algorithmic design choices based on specific input properties (such as data distribution, presence of duplicate keys, pre-sorted inputs, etc.).
4. In [3] Blelloch et al. compare through analysis and implementation three sorting algorithms on the Thinking Machines CM-2. Despite the use of an outdated (and no longer available) platform, this paper is a gem and should be required reading for every parallel algorithm designer. In one of the first studies of its kind, the authors estimate running times of four of the machine's primitives, then analyze the steps of the three sorting algorithms in terms of these parameters. The experimental studies of the performance are normalized to provide clear comparison of how the algorithms scale with input size on a $32K$-processor CM-2.
5. Vitter et al. provide the canonical theoretic foundation for I/O-intensive experimental algorithmics using external parallel disks (e.g., see [1, 19, 20]). Examples from sorting, FFT, permuting, and matrix transposition problems are used to demonstrate the parallel disk model.
6. Juurlink and Wijshoff [11] perform one of the first detailed experimental accounts on the preciseness of several parallel computation models on five parallel platforms. The authors discuss the predictive capabilities of the models, compare the models to find out which allows for the design of the most efficient parallel algorithms, and experimentally compare the performance of algorithms designed with the model versus those designed with machine-specific characteristics in mind. The authors derive model parameters for each platform, analyses for a variety of algorithms (matrix multiplication, bitonic sort, sample sort, all-pairs shortest path), and detailed performance comparisons.
7. The LogP model of Culler et al. [5] provides a realistic model for designing parallel algorithms for message-passing platforms. Its use is demonstrated for a number of problems, including sorting.
8. Several research groups have performed extensive algorithm engineering for high-performance numerical computing. One of the most prominent efforts is that led by Dongarra for ScaLAPACK [4], a scalable linear algebra library for parallel computers. ScaLAPACK encapsulates much of the high-performance algorithm engineering with significant impact to its users who require efficient parallel versions of matrix–matrix linear algebra routines. New approaches for automatically tuning the sequential library (e.g., LAPACK) are now available as the ATLAS package [21].

Open Problems

All of the tools and techniques developed over the last several years for algorithm engineering are applicable to high-performance algorithm engineering. However, many of these tools need

further refinement. For example, cache-efficient programming is a key to performance but it is not yet well understood, mainly because of complex machine-dependent issues like limited associativity, virtual address translation, and increasingly deep hierarchies of high-performance machines. A key question is whether one can find simple models as a basis for algorithm development. For example, cache-oblivious algorithms [7] are efficient at all levels of the memory hierarchy in theory, but so far only few work well in practice. As another example, profiling a running program offers serious challenges in a serial environment (any profiling tool affects the behavior of what is being observed), but these challenges pale in comparison with those arising in a parallel or distributed environment (for instance, measuring communication bottlenecks may require hardware assistance from the network switches or at least reprogramming them, which is sure to affect their behavior). Designing efficient and portable algorithms for commodity multicore and manycore processors is an open challenge.

Cross-References

- Analyzing Cache Misses
- Cache-Oblivious B-Tree
- Cache-Oblivious Model
- Cache-Oblivious Sorting
- Engineering Algorithms for Computational Biology
- Engineering Algorithms for Large Network Applications
- Engineering Geometric Algorithms
- Experimental Methods for Algorithm Analysis
- External Sorting and Permuting
- Implementation Challenge for Shortest Paths
- Implementation Challenge for TSP Heuristics
- I/O-Model
- Visualization Techniques for Algorithm Engineering

Recommended Reading

1. Aggarwal A, Vitter J (1988) The input/output complexity of sorting and related problems. Commun ACM 31:1116–1127
2. Bader DA, Moret BME, Sanders P (2002) Algorithm engineering for parallel computation. In: Fleischer R, Meineche-Schmidt E, Moret BME (eds) Experimental algorithmics. Lecture notes in computer science, vol 2547. Springer, Berlin, pp 1–23
3. Blelloch GE, Leiserson CE, Maggs BM, Plaxton CG, Smith SJ, Zagha M (1998) An experimental analysis of parallel sorting algorithms. Theory Comput Syst 31(2):135–167
4. Choi J, Dongarra JJ, Pozo R, Walker DW (1992) ScaLAPACK: a scalable linear algebra library for distributed memory concurrent computers. In: The 4th symposium on the frontiers of massively parallel computations. McLean, pp 120–127
5. Culler DE, Karp RM, Patterson DA, Sahay A, Schauser KE, Santos E, Subramonian R, von Eicken T (1993) LogP: towards a realistic model of parallel computation. In: 4th symposium on principles and practice of parallel programming. ACM SIGPLAN, pp 1–12
6. Frigo M, Johnson SG (1998) FFTW: an adaptive software architecture for the FFT. In: Proceedings of IEEE international conference on acoustics, speech, and signal processing, Seattle, vol 3, pp 1381–1384
7. Frigo M, Leiserson CE, Prokop H, Ramachandran, S (1999) Cacheoblivious algorithms. In: Proceedings of 40th annual symposium on foundations of computer science (FOCS-99), New York. IEEE, pp 285–297
8. Gropp W, Lusk E, Doss N, Skjellum A (1996) A high-performance, portable implementation of the MPI message passing interface standard. Technical report, Argonne National Laboratory, Argonne. www.mcs.anl.gov/mpi/mpich/
9. Helman DR, JáJá J (1998) Sorting on clusters of SMP's. In: Proceedings of 12th international parallel processing symposium, Orlando, pp 1–7
10. High Performance Fortran Forum (1993) High performance Fortran language specification, 1.0 edn., May 1993
11. Juurlink BHH, Wijshoff HAG (1998) A quantitative comparison of parallel computation models. ACM Trans Comput Syst 13(3):271–318
12. Message Passing Interface Forum (1995) MPI: a message-passing interface standard. Technical report, University of Tennessee, Knoxville, June 1995. Version 1.1
13. Moret BME, Bader DA, Warnow T (2002) High-performance algorithm engineering for computational phylogenetics. J Supercomput 22:99–111, Special issue on the best papers from ICCS'01
14. Moret BME, Shapiro HD (2001) Algorithms and experiments: the new (and old) methodology. J Univ Comput Sci 7(5):434–446
15. Nagel WE, Arnold A, Weber M, Hoppe HC, Solchenbach K (1996) VAMPIR: visualization and analysis of MPI resources. Supercomputer 63 12(1):69–80
16. OpenMP Architecture Review Board (1997) OpenMP: a proposed industry standard API for shared memory programming. www.openmp.org

17. Reed DA, Aydt RA, Noe RJ, Roth PC, Shields KA, Schwartz B, Tavera LF (1993) Scalable performance analysis: the Pablo performance analysis environment. In: Skjellum A (ed) Proceedings of scalable parallel libraries conference, Mississippi State University. IEEE Computer Society Press, pp 104–113
18. Reussner R, Sanders P, Träff J (1998, accepted) SKaMPI: a comprehensive benchmark for public benchmarking of MPI. Scientific programming, 2001. Conference version with Prechelt, L., Müller, M. In: Proceedings of EuroPVM/MPI
19. Vitter JS, Shriver EAM (1994) Algorithms for parallel memory. I: two-level memories. Algorithmica 12(2/3):110–147
20. Vitter JS, Shriver EAM (1994) Algorithms for parallel memory II: hierarchical multilevel memories. Algorithmica 12(2/3):148–169
21. Whaley R, Dongarra J (1998) Automatically tuned linear algebra software (ATLAS). In: Proceedings of supercomputing 98, Orlando. www.netlib.org/utk/people/JackDongarra/PAPERS/atlas-sc98.ps

Holant Problems

Jin-Yi Cai[1,2], Heng Guo[2], and Tyson Williams[2]
[1]Beijing University, Beijing, China
[2]Computer Sciences Department, University of Wisconsin–Madison, Madison, WI, USA

Keywords

Computational complexity; Counting complexity; Holant problems; Partition functions

Years and Authors of Summarized Original Work

2011; Cai, Lu, Xia
2012; Huang, Lu
2013; Cai, Guo, Williams
2013; Guo, Lu, Valiant
2014; Cai, Guo, Williams

Problem Definition

The framework of Holant problems is intended to capture a class of sum-of-product computations in a more refined way than counting CSP problems and is inspired by Valiant's holographic algorithms [12] (also cf. entry ▸ Holographic Algorithms). A constraint function f, or *signature*, is a mapping from $[\kappa]^n$ to $\mathbb{C}$, representing a local contribution to a global sum. Here, $[\kappa]$ is a finite domain set, and n is the arity of f. The range is usually taken to be $\mathbb{C}$, but it can be replaced by any commutative semiring. A Holant problem Holant($\mathcal{F}$) is parameterized by a set of constraint functions $\mathcal{F}$. We usually focus on the Boolean domain, namely, $\kappa = 2$. For consideration of models of computation, we restrict function values to be complex algebraic numbers.

We allow multigraphs, namely, graphs with self-loops and parallel edges. A *signature grid* $\Omega = (G, \pi)$ of Holant($\mathcal{F}$) consists of a graph $G = (V, E)$, where π assigns each vertex $v \in V$ and its incident edges with some $f_v \in \mathcal{F}$ and its input variables. We say Ω is a *planar signature grid* if G is planar. The Holant problem on instance Ω is to evaluate

$$\text{Holant}(\Omega; \mathcal{F}) = \sum_{\sigma} \prod_{v \in V} f_v(\sigma \mid_{E(v)}),$$

a sum over all edge labelings $\sigma : E \to [\kappa]$, where $E(v)$ denotes the incident edges of v and $\sigma \mid_{E(v)}$ denotes the restriction of σ to $E(v)$. This is also known as the partition function in the statistical physics literature.

Formally, a set of signatures $\mathcal{F}$ defines the following Holant problem:

Name Holant($\mathcal{F}$)
Instance A *signature grid* $\Omega = (G, \pi)$
Output Holant($\Omega; \mathcal{F}$)

The problem Pl-Holant($\mathcal{F}$) is defined similarly using a planar signature grid.

A function f_v can be represented by listing its values in lexicographical order as in a truth table, which is a vector in $\mathbb{C}^{\kappa^{\deg(v)}}$ or as a tensor in $(\mathbb{C}^{\kappa})^{\otimes \deg(v)}$. Special focus has been put on *symmetric* signatures, which are functions invariant under any permutation of the input. An example is the EQUALITY signature $=_n$ of arity n. A Boolean symmetric function f of arity n can be listed as $[f_0, f_1, \ldots, f_n]$, where f_w is the function value of f when the input has Hamming

weight w. Using this notation, an EQUALITY signature is $[1, 0, \ldots, 0, 1]$. Another example is the EXACTONE signature $[0, 1, 0, \ldots, 0]$. Clearly, the Holant problem defined by this signature counts the number of perfect matchings.

The set $\mathcal{F}$ is allowed to be an infinite set. For Holant($\mathcal{F}$) to be tractable, the problem must be computable in polynomial time even when the description of the signatures in the input Ω is included in the input size. In contrast, we say Holant($\mathcal{F}$) is #P-hard if there exists a finite subset of $\mathcal{F}$ for which the problem is #P-hard.

The Holant framework is a generalization and refinement of both counting graph homomorphisms and counting constraint satisfaction problems (see entry ▸Complexity Dichotomies for Counting Graph Homomorphisms for more details and results).

Key Results

The Holant problem was introduced by Cai, Lu, and Xia [3], which also contains a dichotomy of Holant* for symmetric Boolean complex functions. The notation Holant* means that all unary functions are assumed to be available. This restriction is later weakened to only allow two constant functions that pin a variable to 0 or 1. This framework is called Holant^c. In [5], a dichotomy of Holant^c is obtained. The need to assume some freely available functions is finally avoided in [10]. In this paper, Huang and Lu proved a dichotomy for Holant but with the caveat that the functions must be real weighted. This result was later improved by Cai, Guo, and Williams [6], who proved a dichotomy for Holant parameterized by any set of symmetric Boolean complex functions.

We will give some necessary definitions and then state the dichotomy from [6]. First are several tractable families of functions over the Boolean domain.

Definition 1 A signature f of arity n is *degenerate* if there exist unary signatures $u_j \in \mathbb{C}^2$ $(1 \le j \le n)$ such that $f = u_1 \otimes \cdots \otimes u_n$. A symmetric degenerate signature has the form $u^{\otimes n}$.

Definition 2 A k-ary function $f(x_1, \ldots, x_k)$ is of *affine* type if it has the form

$$\lambda \chi_{Ax=0} \cdot \sqrt{-1}^{\sum_{j=1}^{n} \langle \alpha_j, x \rangle},$$

where $\lambda \in \mathbb{C}$, $x = (x_1, x_2, \ldots, x_k, 1)^{\mathrm{T}}$, A is a matrix over $\mathbb{F}_2$, α_j is a vector over $\mathbb{F}_2$, and χ is a 0–1 indicator function such that $\chi_{Ax=0}$ is 1 iff $Ax = 0$. Note that the dot product $\langle \alpha_j, x \rangle$ is calculated over $\mathbb{F}_2$, while the summation $\sum_{j=1}^{n}$ on the exponent of $i = \sqrt{-1}$ is evaluated as a sum mod 4 of 0–1 terms. We use $\mathscr{A}$ to denote the set of all affine-type functions.

An alternative but equivalent form for an affine-type function is $\lambda \chi_{Ax=0} \cdot \sqrt{-1}^{Q(x_1, x_2, \ldots, x_k)}$ where $Q(\cdot)$ is a quadratic form with integer coefficients that are even for every cross term.

Definition 3 A function is of *product type* if it can be expressed as a product of unary functions, binary equality functions ($[1, 0, 1]$), and binary disequality functions ($[0, 1, 0]$), each applied to some of its variables. We use $\mathscr{P}$ to denote the set of product-type functions.

Definition 4 A function f is called *vanishing* if the value Holant($\Omega; \{f\}$) is 0 for every signature grid Ω. We use $\mathscr{V}$ to denote the set of vanishing functions.

For vanishing signatures, we need some more definitions.

Definition 5 An arity n symmetric signature of the form $f = [f_0, f_1, \ldots, f_n]$ is in $\mathscr{R}_t^+$ for a nonnegative integer $t \ge 0$ if $t > n$ or for any $0 \le k \le n - t$, $f_k, \ldots, f_{k+t}$ satisfy the recurrence relation

$$\binom{t}{t} i^t f_{k+t} + \binom{t}{t-1} i^{t-1} f_{k+t-1} + \cdots + \binom{t}{0} i^0 f_k = 0. \quad (1)$$

We define $\mathscr{R}_t^-$ similarly but with $-i$ in place of i in (1).

With $\mathscr{R}_t^{\pm}$, one can define the recurrence degree of a function f.

Definition 6 For a nonzero symmetric signature f of arity n, it is of *positive* (resp. *negative*) *recurrence degree* $t \le n$, denoted by $\mathrm{rd}^+(f) = t$ (resp. $\mathrm{rd}^-(f) = t$), if and only if $f \in \mathscr{R}_{t+1}^+ - \mathscr{R}_t^+$ (resp. $f \in \mathscr{R}_{t+1}^- - \mathscr{R}_t^-$). If f is the all-zero signature, we define $\mathrm{rd}^+(f) = \mathrm{rd}^-(f) = -1$.

In [6], it is shown that $f \in \mathscr{V}$ if and only if for either $\sigma = +$ or $-$, we have $2\mathrm{rd}^\sigma(f) < \mathrm{arity}(f)$. Accordingly, we split the set $\mathscr{V}$ of vanishing signatures in two.

Definition 7 We define $\mathscr{V}^\sigma$ for $\sigma \in \{+, -\}$ as

$$\mathscr{V}^\sigma = \{f \mid 2\mathrm{rd}^\sigma(f) < \mathrm{arity}(f)\}.$$

To state the dichotomy, we also need the notion of $\mathcal{F}$-transformable. For a matrix $T \in \mathbb{C}^{2\times 2}$, and a signature set $\mathcal{F}$, define $T\mathcal{F} = \{g \mid \exists f \in \mathcal{F}$ of arity n, $g = T^{\otimes n} f\}$. Here, we view the signatures as column vectors. Let $=_2$ be the equality function of arity 2.

Definition 8 A signature set $\mathcal{F}'$ is $\mathcal{F}$-transformable if there exists a non-singular matrix $T \in \mathbb{C}^{2\times 2}$ such that $\mathcal{F}' \subseteq T\mathcal{F}$ and $(=_2)T^{\otimes 2} \in \mathcal{F}$.

If a set of functions $\mathcal{F}'$ is $\mathcal{F}$-transformable and $\mathcal{F}$ is a tractable set, then Holant($\mathcal{F}'$) is tractable as well.

The dichotomy of Holant problems over symmetric Boolean complex functions is stated as follows.

Theorem 1 ([6]) *Let $\mathcal{F}$ be any set of symmetric, complex-valued signatures in Boolean variables. Then,* Holant($\mathcal{F}$) *is #P-hard unless $\mathcal{F}$ satisfies one of the following conditions, in which case the problem is in P:*

1. *All nondegenerate signatures in $\mathcal{F}$ are of arity at most 2;*
2. *$\mathcal{F}$ is $\mathscr{A}$-transformable;*
3. *$\mathcal{F}$ is $\mathscr{P}$-transformable;*
4. *$\mathcal{F} \subseteq \mathscr{V}^\sigma \cup \{f \in \mathscr{R}_2^\sigma \mid \mathrm{arity}(f) = 2\}$ for some $\sigma \in \{+, -\}$;*
5. *All nondegenerate signatures in $\mathcal{F}$ are in $\mathscr{R}_2^\sigma$ for some $\sigma \in \{+, -\}$.*

Theorem 1 is about Holant problems parameterized by symmetric Boolean complex functions over general graphs. Holant problems are studied in other settings as well. For planar graphs, [2] contains a dichotomy for Holant^c with real symmetric functions. There are signature sets that are #P-hard over general graphs but tractable over planar graphs. The algorithms for such sets are due to Valiant's holographic algorithms and the theory of matchgates [1, 12].

Another generalization looks at a broader range of functions. One may consider asymmetric functions as in [4], which contains a dichotomy for Holant* problems defined by asymmetric Boolean complex functions. One can also consider functions of larger domain size. For domain size 3, [7] contains a dichotomy for a single arity 3 symmetric complex function in the Holant* setting. For any constant domain size, [8] contains a dichotomy for a single arity 3 complex weighted function that satisfies a strong symmetry property.

One can consider constraint functions with a range other than $\mathbb{C}$. Replacing $\mathbb{C}$ by some finite field $\mathbb{F}_p$ for some prime p defines counting problems modulo p. The case $p = 2$ is called parity Holant problems. It is of special interest because computing the permanent modulo 2 is tractable, which implies a family of tractable matchgate functions even over general graphs. For parity Holant problems, a complete dichotomy for symmetric functions is obtained by Guo, Lu, and Valiant [9].

Open Problems

Unlike the progress in the general graph setting, the strongest known dichotomy results for planar Holant problems are rather limited. These planar dichotomies showed that newly tractable problems over planar graphs are captured by holographic algorithms with matchgates, but with restrictions like symmetric functions or regular graphs. The theory of holographic algorithms

with matchgates can be applied to planar graphs and asymmetric signatures. A true test of its power would be to obtain an asymmetric complex weighted dichotomy of planar Holant problems. The situation is similarly limited for higher domain sizes, where things seem considerably more complicated. A reasonable first step in this direction would be to consider some restricted (yet still powerful) family of functions.

Despite the success for $\mathbb{F}_2$, little is known about the complexity of Holant problems over other finite fields or semirings. As Valiant showed in [11], counting problems modulo some finite modulus include some interesting and surprising phenomena. It deserves further research.

Cross-References

- ▸ Complexity Dichotomies for Counting Graph Homomorphisms
- ▸ Holographic Algorithms

Recommended Reading

1. Cai JY, Lu P (2011) Holographic algorithms: from art to science. J Comput Syst Sci 77(1):41–61
2. Cai JY, Lu P, Xia M (2010) Holographic algorithms with matchgates capture precisely tractable planar #CSP. In: FOCS, Las Vegas. IEEE Computer Society, pp 427–436
3. Cai JY, Lu P, Xia M (2011) Computational complexity of Holant problems. SIAM J Comput 40(4):1101–1132
4. Cai JY, Lu P, Xia M (2011) Dichotomy for Holant* problems of Boolean domain. In: SODA, San Francisco. SIAM, pp 1714–1728
5. Cai JY, Huang S, Lu P (2012) From Holant to #CSP and back: dichotomy for Holant^c problems. Algorithmica 64(3):511–533
6. Cai JY, Guo H, Williams T (2013) A complete dichotomy rises from the capture of vanishing signatures (extended abstract). In: STOC, Palo Alto. ACM, pp 635–644
7. Cai JY, Lu P, Xia M (2013) Dichotomy for Holant* problems with domain size 3. In: SODA, New Orleans. SIAM, pp 1278–1295
8. Cai JY, Guo H, Williams T (2014) The complexity of counting edge colorings and a dichotomy for some higher domain Holant problems. In: FOCS, Philadelphia. IEEE, pp 601–610
9. Guo H, Lu P, Valiant LG (2013) The complexity of symmetric Boolean parity Holant problems. SIAM J Comput 42(1):324–356
10. Huang S, Lu P (2012) A dichotomy for real weighted Holant problems. In: CCC, Porto. IEEE Computer Society, pp 96–106
11. Valiant LG (2006) Accidental algorithms. In: FOCS, Berkeley. IEEE, pp 509–517
12. Valiant LG (2008) Holographic algorithms. SIAM J Comput 37(5):1565–1594

Holographic Algorithms

Jin-Yi Cai[1,2], Pinyan Lu[3], and Mingji Xia[4]
[1]Beijing University, Beijing, China
[2]Computer Sciences Department, University of Wisconsin–Madison, Madison, WI, USA
[3]Microsoft Research Asia, Shanghai, China
[4]The State Key Laboratory of Computer Science, Chinese Academy of Sciences, Beijing, China

Keywords

Bases; Counting problems; Holographic algorithms; Perfect matchings; Planar graphs

Years and Authors of Summarized Original Work

2006, 2008; Valiant
2008, 2009, 2010, 2011; Cai, Lu
2009; Cai, Choudhary, Lu
2014; Cai, Gorenstein

Problem Definition

Holographic algorithm, introduced by L. Valiant [11], is an algorithm design technique rather than a single algorithm for a particular problem. In essence, these algorithms are reductions to the FKT algorithm [7–9] to count the number of perfect matchings in a planar graph in polynomial time. Computation in these algorithms is expressed and interpreted through a choice of linear basis vectors in an exponential "holographic" mix, and then it is carried out by the FKT method via the Holant Theorem. This methodology has

produced polynomial time algorithms for a variety of problems ranging from restrictive versions of satisfiability, vertex cover, to other graph problems such as edge orientation and node/edge deletion. No polynomial time algorithms were known for these problems, and some minor variations are known to be NP-hard (or even #P-hard).

Let $G = (V, E, W)$ be a weighted undirected planar graph, where V, E, and W are sets of vertices, edges, and edge weights, respectively. A matchgate is a tuple (G, X) where $X \subseteq V$ is a set of external nodes on the outer face. A matchgate is considered a generator or a recognizer matchgate when the external nodes are considered output or input nodes, respectively. They differ mainly in the way they are transformed. The external nodes are ordered clockwise on the external face. Γ is called an odd (resp. even) matchgate if it has an odd (resp. even) number of nodes.

Each matchgate is assigned a *signature* tensor. A generator Γ with m output nodes is assigned a contravariant tensor $\mathbf{G} \in V_0^m$ of type $\binom{m}{0}$, where V_0^m is the tensor space spanned by the m-fold tensor products of the standard basis $\mathbf{b} = [\mathbf{b}_0, \mathbf{b}_1] = \left[\binom{1}{0}, \binom{0}{1}\right]$. The tensor $\mathbf{G}$ under the standard basis $\mathbf{b}$ has the form

$$\sum G^{i_1 i_2 \ldots i_m} \mathbf{b}_{i_1} \otimes \mathbf{b}_{i_2} \otimes \cdots \otimes \mathbf{b}_{i_m},$$

where

$$G^{i_1 i_2 \ldots i_m} = \text{PerfMatch}(G - Z).$$

Here Z is the subset of the output nodes of Γ having the characteristic sequence $\chi_Z = i_1 i_2 \ldots i_m \in \{0,1\}^m$, $\text{PerfMatch}(G - Z) = \sum_M \prod_{(i,j)\in M} w_{ij}$ is a sum over all perfect matchings M in the graph $G - Z$ obtained from G by removing Z and its incident edges, and w_{ij} is the weight of the edge (i, j). Similarly a recognizer Γ' with underlying graph G' having m input nodes is assigned a covariant tensor $\mathbf{R} \in V_m^0$ of type $\binom{0}{m}$. This tensor under the standard (dual) basis $\mathbf{b}^*$ has the form

$$\sum R_{i_1 i_2 \ldots i_m} \mathbf{b}^{i_1} \otimes \mathbf{b}^{i_2} \otimes \cdots \otimes \mathbf{b}^{i_m},$$

where

$$R_{i_1 i_2 \ldots i_m} = \text{PerfMatch}(G' - Z),$$

and Z is the subset of the input nodes of Γ' having the characteristic sequence $\chi_Z = i_1 i_2 \ldots i_m$.

As a contravariant tensor, $\mathbf{G}$ transforms as follows. Under a basis transformation $\boldsymbol{\beta}_j = \sum_i \mathbf{b}_i t_j^i$,

$$(G')^{j_1 j_2 \cdots j_m} = \sum G^{i_1 i_2 \ldots i_m} \tilde{t}_{i_1}^{j_1} \tilde{t}_{i_2}^{j_2} \cdots \tilde{t}_{i_m}^{j_m},$$

where $(\tilde{t}_i^j)$ is the inverse matrix of (t_j^i). Similarly, $\mathbf{R}$ transforms as a covariant tensor, namely,

$$(R')_{j_1 j_2 \ldots j_m} = \sum R_{i_1 i_2 \ldots i_m} t_{j_1}^{i_1} t_{j_2}^{i_2} \cdots t_{j_m}^{i_m}.$$

A signature is *symmetric* if each entry only depends on the Hamming weight of the index $i_1 i_2 \ldots i_m$. This notion is invariant under a basis transformation. A symmetric signature is denoted by $[\sigma_0, \sigma_1, \ldots, \sigma_m]$, where σ_i denotes the value of a signature entry whose Hamming weight of its index is i.

A *matchgrid* $\Omega = (A, B, C)$ is a weighted planar graph consisting of a disjoint union of: a set of g generators $A = (A_1, \ldots, A_g)$, a set of r recognizers $B = (B_1, \ldots, B_r)$, and a set of f connecting edges $C = (C_1, \ldots, C_f)$, where each C_i edge has weight 1 and joins an output node of a generator with an input node of a recognizer, so that every input and output node in every constituent matchgate has exactly one such incident connecting edge.

Let $\mathbf{G} = \bigotimes_{i=1}^{g} \mathbf{G}(A_i)$ be the tensor product of all the generator signatures, and let $\mathbf{R} = \bigotimes_{j=1}^{r} \mathbf{R}(B_j)$ be the tensor product of all the recognizer signatures. Then Holant_Ω is defined to be the contraction of the two product tensors, under some basis $\boldsymbol{\beta}$, where the corresponding indices match up according to the f connecting edges C_k:

$$\text{Holant}_{\Omega} = \langle \mathbf{R}, \mathbf{G} \rangle = \sum_{x \in \boldsymbol{\beta}^{\otimes f}} \left\{ [\Pi_{1 \le i \le g} \mathbf{G}(A_i, x|_{A_i})] \cdot [\Pi_{1 \le j \le r} \mathbf{R}(B_j, x^*|_{B_j})] \right\}. \quad (1)$$

If we write the covariant tensor $\mathbf{R}$ as a row vector of dimension 2^f, write the contravariant tensor $\mathbf{G}$ as a column vector of dimension 2^f, both indexed by some common ordering of the connecting edges, then Holant_{Ω} is just the dot product of these two vectors. Valiant's beautiful Holant Theorem is as follows:

Theorem 1 (Valiant) *For any matchgrid Ω over any basis $\boldsymbol{\beta}$, let G be its underlying weighted graph, then*

$$\text{Holant}_{\Omega} = \text{PerfMatch}(G).$$

The FKT algorithm can compute the perfect matching polynomial $\text{PerfMatch}(G)$ for a planar graph in polynomial time. This gives a polynomial time algorithm to compute Holant_{Ω}.

Key Results

To design a holographic algorithm for a given problem, the creative part is to formalize the given problem as a Holant problem. The theory of holographic algorithms is trying to answer the second question: given a Holant problem, can we find a basis transformation so that all the signatures in the Holant problem can be realized by some matchgates on that basis? More formally, we want to solve the following simultaneous realizability problem (SRP).

Definition 1 Simultaneous Realizability Problem (SRP):

Input: A set of constraint functions for generators and recognizers.

Output: A common basis under which these functions can be simultaneously realized by matchgate signatures, if any exists; "NO" if they are not simultaneously realizable.

The theory of matchgates and holographic algorithms provides a systematic understanding of which constraint functions can be realized by matchgates, the structure for the bases, and finally solve the simultaneous realizability problem.

Matchgate Identities

There is a set of algebraic identities [1, 6] which completely characterizes signatures directly realizable without basis transformation by matchgates for any number of inputs and outputs. These identities are derived from Grassmann-Plücker identities for Pfaffians.

Patterns α, β are m-bit strings, i.e., $\alpha, \beta \in \{0,1\}^m$. A position vector $P = \{p_i\}, i \in [l]$ is a subsequence of $\{1, 2, \ldots, m\}$, i.e., $p_i \in [m]$ and $p_1 < p_2 < \cdots < p_l$. We also use p to denote the m-bit string, whose $(p_1, p_2, \ldots, p_l)$-th bits are 1 and others are 0. Let $e_i \in \{0,1\}^m$ be the pattern with 1 in the i-th bit and 0 elsewhere. Let $\alpha, \beta \in \{0,1\}^m$ be any pattern, and let $P = \{p_i\} = \alpha + \beta$, $i \in [l]$ be their bit-wise XOR as a position vector. Then, we have the following identity:

$$\sum_{i=1}^{l} (-1)^i G^{\alpha + e_{p_i}} G^{\beta + e_{p_i}} = 0. \quad (2)$$

A tensor $\mathbf{G} = (G^{i_1, \ldots, i_m})$ is realizable as the signature, without basis transformation, of some planar matchgate iff it satisfies the matchgate identities (2) for all α and β.

Basis Collapse

When we consider basis transformations for holographic algorithms, we mainly focus on

invertible transformations, and these are bases of dimension 2. However, in a paper called "accidental algorithm" [10], Valiant showed that a basis of dimension 4 can be used to solve in P an interesting (restrictive SAT) counting problem mod 7. In a later paper [4], we have shown, among other things, that for this particular problem, this use of bases of size 2 is unnecessary. Then, in a sequence of two papers [2, 3], we completely resolve the problem of the power of higher dimensional bases. We prove that 2-dimensional bases are universal for holographic algorithms in the Boolean domain.

Theorem 2 (Basis Collapse Theorem) *Any holographic algorithm on a basis of any dimension which employs at least one nondegenerate generator can be efficiently transformed to a holographic algorithm in a basis of dimension 2. More precisely, if generators $G_1, G_2, \ldots, G_s$ and recognizers $R_1, R_2, \ldots, R_t$ are simultaneously realizable on a basis T of any dimension, and not all generators are degenerate, then all the generators and recognizers are simultaneously realizable in a basis $\hat{T}$ of dimension 2.*

From Art to Science

Based on the characterization for matchgate signatures and basis transformations, we can solve the simultaneous realizability problem [5]. In order to investigate the realizability of signatures, it is useful to introduce a basis manifold $\mathcal{M}$, which is defined to be the set of all possible bases modulo an equivalence relation. One can characterize in terms of $\mathcal{M}$ all realizable symmetric signatures under basis transformations. This structural understanding gives: (i) a uniform account of all the previous successes of holographic algorithms using symmetric signatures [10, 11]; (ii) generalizations to solve other problems, when this is possible; and (iii) a proof when this is not possible.

Applications

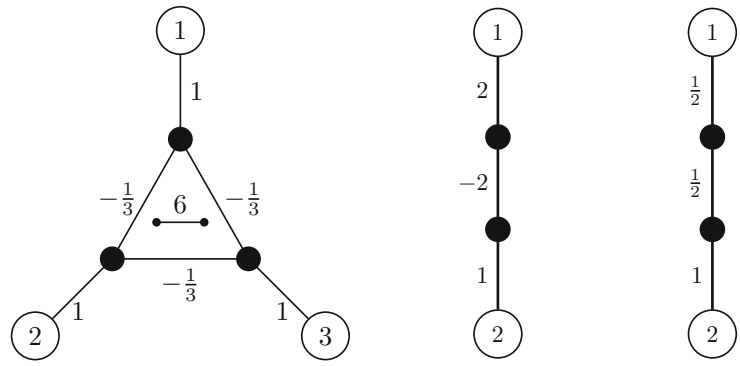
In this section, we list a few problems which can be solved by holographic algorithms.

#PL-3-NAE-ICE

INPUT: A planar graph $G = (V, E)$ of maximum degree 3.

OUTPUT: The number of orientations such that no node has all incident edges directed toward it or all incident edges directed away from it.

Hence, #PL-3-NAE-ICE counts the number of no-sink-no-source orientations. A node of degree one will preclude such an orientation. We assume every node has degree 2 or 3. To solve this problem by a holographic algorithm with matchgates, we design a signature grid based on G as follows: We attach to each node of degree 3 a generator with signature $[0, 1, 1, 0]$. This represents a NOT-ALL-EQUAL or NAE gate of arity 3. For any node of degree 2, we use a generator with the binary NAE (i.e., a binary DISEQUALITY) signature $(\neq_2) = [0, 1, 0]$. For each edge in E, we use a recognizer with signature $(\neq_2)$, which stands for an orientation from one node to the other. (To express such a problem, it is completely arbitrary



Holographic Algorithms, Fig. 1 Some matchgates used in #PL-3-NAE-ICE

to label one side as generators and the other side as recognizers.) From the given planar graph G, we obtain a signature grid Ω, where the underlying graph G' is the edge-vertex incidence graph of G. By definition, Holant_Ω is an exponential sum where each term is a product of appropriate entries of the signatures. Each term is indexed by a 0–1 assignment on all edges of G'; it has a value of 0 or 1, and it has a value of 1 iff it corresponds to an orientation of G such that at every vertex of G the local NAE constraint is satisfied. Therefore, Holant_Ω is precisely the number of valid orientations required by #PL-3-NAE-ICE.

Note that the signature $[0, 1, 1, 0]$ is not the signature of any matchgate. A simple reason for this is that a matchgate signature, being defined in terms of perfect matchings, cannot have nonzero values for inputs of both odd and even Hamming weights.

However, under a holographic transformation using $H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$,

$$H^{\otimes 3}[0, 1, 1, 0] = H^{\otimes 3}\left\{\begin{bmatrix}1\\1\end{bmatrix}^{\otimes 3} - \begin{bmatrix}1\\0\end{bmatrix}^{\otimes 3} - \begin{bmatrix}0\\1\end{bmatrix}^{\otimes 3}\right\} = [6, 0, -2, 0],$$

$$H^{\otimes 2}[0, 1, 0] = H^{\otimes 2}\left\{\begin{bmatrix}1\\1\end{bmatrix}^{\otimes 2} - \begin{bmatrix}1\\0\end{bmatrix}^{\otimes 2} - \begin{bmatrix}0\\1\end{bmatrix}^{\otimes 2}\right\} = [2, 0, -2],$$

and

$$[0, 1, 0](H^{-1})^{\otimes 2} = \frac{1}{2}[1, 0, -1].$$

These signatures are all realizable as matchgate signatures by verifying all the matchgate identities. More concretely, we can exhibit the requisite three matchgates in Fig. 1.

Hence, #PL-3-NAE-ICE is precisely the following Holant problem on planar graphs:

$$\text{Holant}([0, 1, 0] \mid [0, 1, 0], [0, 1, 1, 0])$$
$$\equiv_T \text{Holant}(\tfrac{1}{2}[1, 0, -1] \mid [2, 0, -2], [6, 0, -2, 0]).$$

Now we may replace each signature $\frac{1}{2}[1, 0, -1]$, $[2, 0, -2]$, and $[6, 0, -2, 0]$ in Ω by their corresponding matchgates, and then we can compute Holant_Ω in polynomial time by Kasteleyn's algorithm.

The next problem is a satisfiability problem.

#PL-3-NAE-SAT

INPUT: A planar formula Φ consisting of a conjunction of NAE clauses each of size 3.

OUTPUT: The number of satisfying assignments of Φ.

This is a variant of 3SAT. A Boolean formula is planar if it can be represented by a planar graph where vertices represent variables and clauses, and there is an edge iff the variable or its negation appears in that clause. The SAT problem is when the gate for each clause is the Boolean OR. When SAT is restricted to planar formulae, it is still NP-complete, and its corresponding counting problem is #P-complete. Moreover, for many connectives other than NAE (e.g., EXACTLY ONE), the unrestricted or the planar decision problems are still NP-complete, and the corresponding counting problems are #P-complete.

We design a signature grid as follows: To each NAE clause, we assign a generator with signature $[0, 1, 1, 0]$. To each Boolean variable, we assign a generator with signature $(=_k)$ where k is the number of clauses the variable appears, either negated or unnegated. Further, if a variable occurrence is negated, we have a recognizer $[0, 1, 0]$ along the edge that joins the variable generator and the NAE generator, and if the variable occurrence is unnegated, then we use a recognizer $[1, 0, 1]$ instead. Under a holographic transformation using H, $(=_k)$ is transformed to

$$H^{\otimes k}\left\{\begin{bmatrix}1\\0\end{bmatrix}^{\otimes k}+\begin{bmatrix}0\\1\end{bmatrix}^{\otimes k}\right\}=\begin{bmatrix}1\\1\end{bmatrix}^{\otimes k}+\begin{bmatrix}1\\-1\end{bmatrix}^{\otimes k}$$
$$=2[1,0,1,0,\ldots].$$

It can be verified that all the signatures used satisfy all matchgate identities and thus can be realized by matchgates under the holographic transformation.

Recommended Reading

1. Cai JY, Gorenstein A (2014) Matchgates revisited. Theory Comput 10(7):167–197
2. Cai JY, Lu P (2008) Basis collapse in holographic algorithms. Comput Complex 17(2):254–281
3. Cai JY, Lu P (2009) Holographic algorithms: the power of dimensionality resolved. Theor Comput Sci Comput Sci 410(18):1618–1628
4. Cai JY, Lu P (2010) On symmetric signatures in holographic algorithms. Theory Comput Syst 46(3):398–415
5. Cai JY, Lu P (2011) Holographic algorithms: from art to science. J Comput Syst Sci 77(1):41–61
6. Cai JY, Choudhary V, Lu P (2009) On the theory of matchgate computations. Theory Comput Syst 45(1):108–132
7. Kasteleyn PW (1961) The statistics of dimers on a lattice. Physica 27:1209–1225
8. Kasteleyn PW (1967) Graph theory and crystal physics. In: Harary F (ed) Graph theory and theoretical physics. Academic, London, pp 43–110
9. Temperley HNV, Fisher ME (1961) Dimer problem in statistical mechanics – an exact result. Philos Mag 6:1061–1063
10. Valiant LG (2006) Accidental algorthims. In: FOCS '06: proceedings of the 47th annual IEEE symposium on foundations of computer science. IEEE Computer Society, Washington, pp 509–517. doi:http://dx.doi.org/10.1109/FOCS.2006.7
11. Valiant LG (2008) Holographic algorithms. SIAM J Comput 37(5):1565–1594. doi:http://dx.doi.org/10.1137/070682575

Hospitals/Residents Problem

David F. Manlove
School of Computing Science, University of Glasgow, Glasgow, UK

Keywords

Matching; Stability

Synonyms

College admissions problem; Stable admissions problem; Stable assignment problem; Stable b-matching problem; University admissions problem

Years and Authors of Summarized Original Work

1962; Gale, Shapley

Problem Definition

An instance I of the *Hospitals/Residents problem* (HR) [6, 7, 18] involves a set $R = \{r_1, \ldots, r_n\}$ of *residents* and a set $H = \{h_1, \ldots, h_m\}$ of *hospitals*. Each hospital $h_j \in H$ has a positive integral *capacity*, denoted by c_j. Also, each resident $r_i \in R$ has a *preference list* in which he ranks in strict order a subset of H. A pair $(r_i, h_j) \in R \times H$ is said to be *acceptable* if h_j appears in r_i's preference list; in this case r_i is said to *find* h_j *acceptable*. Similarly each hospital $h_j \in H$ has a preference list in which it ranks in strict order those residents who find h_j acceptable. Given any three agents $x, y, z \in R \cup H$, x is said to *prefer* y to z if x finds each of y and z acceptable, and y precedes z on x's preference list. Let $C = \sum_{h_j \in H} c_j$.

Let A denote the set of acceptable pairs in I, and let $L = |A|$. An *assignment* M is a subset of A. If $(r_i, h_j) \in M$, r_i is said to be *assigned to* h_j, and h_j is *assigned* r_i. For each $q \in R \cup H$, the set of assignees of q in M is denoted by $M(q)$. If $r_i \in R$ and $M(r_i) = \emptyset$, r_i is said to be *unassigned*; otherwise r_i is *assigned*. Similarly, any hospital $h_j \in H$ is *under-subscribed*, *full*, or *over-subscribed* according as $|M(h_j)|$ is less than, equal to, or greater than c_j, respectively.

A *matching* M is an assignment such that $|M(r_i)| \leq 1$ for each $r_i \in R$ and $|M(h_j)| \leq c_j$ for each $h_j \in H$ (i.e., no resident is assigned to an unacceptable hospital, each resident is assigned to at most one hospital, and no hospital is over-subscribed). For notational convenience, given a matching M and a resident $r_i \in R$ such

that $M(r_i) \neq \emptyset$, where there is no ambiguity, the notation $M(r_i)$ is also used to refer to the single member of $M(r_i)$.

A pair $(r_i, h_j) \in A \backslash M$ *blocks* a matching M or is a *blocking pair* for M, if the following conditions are satisfied relative to M:

1. r_i is unassigned or prefers h_j to $M(r_i)$;
2. h_j is under-subscribed or prefers r_i to at least one member of $M(h_j)$ (or both).

A matching M is said to be *stable* if it admits no blocking pair. Given an instance I of HR, the problem is to find a stable matching in I.

Key Results

HR was first defined by Gale and Shapley [6] under the name "College Admissions Problem." In their seminal paper, the authors' primary consideration is the classical *Stable Marriage problem* (SM; see Entries ▸ Stable Marriage and ▸ Optimal Stable Marriage), which is a special case of HR in which $n = m$, $A = R \times H$, and $c_j = 1$ for all $h_j \in H$ – in this case, the residents and hospitals are more commonly referred to as the *men* and *women*, respectively. Gale and Shapley showed that every instance I of HR admits at least one stable matching. Their proof of this result is constructive, i.e., an algorithm for finding a stable matching in I is described. This algorithm has become known as the *Gale/Shapley algorithm*.

An extended version of the Gale/Shapley algorithm for HR is shown in Fig. 1. The algorithm involves a sequence of *apply* and *delete* operations. At each iteration of the while loop, some unassigned resident r_i with a nonempty preference list applies to the first hospital h_j on his list and becomes provisionally assigned to h_j (this assignment could subsequently be broken). If h_j becomes over-subscribed as a result of this assignment, then h_j rejects its worst assigned resident r_k. Next, if h_j is full (irrespective of whether h_j was over-subscribed earlier in the same loop iteration), then for each resident r_l that h_j finds less desirable than its worst assigned resident r_k, the algorithm *deletes the pair* (r_l, h_j), which comprises deleting h_j from r_l's preference list and vice versa.

Given that the above algorithm involves residents applying to hospitals, it has become known as the *Resident-oriented* Gale/Shapley algorithm, or RGS algorithm for short [7, Section 1.6.3]. The RGS algorithm terminates with a stable matching, given an instance of HR [6] [7, Theorem 1.6.2]. Using a suitable choice of data structures (extending those described in [7, Section 1.2.3]), the RGS algorithm can be implemented to run in $O(L)$ time. This algorithm produces the unique stable matching that is simultaneously best possible for all residents [6] [7, Theorem 1.6.2]. These observations may be summarized as follows:

Theorem 1 *Given an instance of HR, the RGS algorithm constructs, in $O(L)$ time, the unique stable matching in which each assigned resident obtains the best hospital that he could obtain in any stable matching, while each unassigned resident is unassigned in every stable matching.*

A counterpart of the RGS algorithm, known as the *Hospital-oriented Gale/Shapley algorithm*, or HGS algorithm for short [7, Section 1.6.2], gives the unique stable matching that similarly satisfies an optimality property for the hospitals [7, Theorem 1.6.1].

Although there may be many stable matchings for a given instance I of HR, some key structural properties hold regarding unassigned residents and under-subscribed hospitals with respect to all stable matchings in I, as follows.

Theorem 2 *For a given instance of HR:*

- *The same residents are assigned in all stable matchings;*
- *Each hospital is assigned the same number of residents in all stable matchings;*
- *Any hospital that is under-subscribed in one stable matching is assigned exactly the same set of residents in all stable matchings.*

These results are collectively known as the "Rural Hospitals Theorem" (see [7, Section 1.6.4] for further details). Furthermore, the set of stable matchings in I forms a distributive lattice under a natural dominance relation [7, Section 1.6.5].

```
M := ∅;
while (some resident r_i is unassigned and r_i has a nonempty list) {
    h_j := first hospital on r_i's list;
    /* r_i applies to h_j */
    M := M ∪ {(r_i, h_j)};
    if (h_j is over-subscribed) {
        r_k := worst resident in M(h_j) according to h_j's list;
        M := M\{(r_k, h_j)};
    }
    if (h_j is full) {
        r_k := worst resident in M(h_j) according to h_j's list;
        for (each successor r_l of r_k on h_j's list)
            delete the pair (r_l, h_j);
    }
}
```

Hospitals/Residents Problem, Fig. 1 Gale/Shapley algorithm for HR

Applications

Practical applications of HR are widespread, most notably arising in the context of centralized automated matching schemes that assign applicants to posts (e.g., medical students to hospitals, school leavers to universities, and primary school pupils to secondary schools). Perhaps the largest and best-known example of such a scheme is the National Resident Matching Program (NRMP) in the USA [8], which annually assigns around 31,000 graduating medical students (known as residents) to their first hospital posts, taking into account the preferences of residents over hospitals and vice versa and the hospital capacities. Counterparts of the NRMP are in existence in other countries, including Canada [9] and Japan [10]. These matching schemes essentially employ extensions of the RGS algorithm for HR.

Centralized matching schemes based largely on HR also occur in other practical contexts, such as school placement in New York [1], university faculty recruitment in France [3], and university admission in Spain [16]. Further applications are described in [15, Section 1.3.7].

Indeed, the Nobel Prize in Economic Sciences was awarded in 2012 to Alvin Roth and Lloyd Shapley, partly for their theoretical work on HR and its variants [6, 18] and partly for their contribution to the widespread deployment of algorithms for HR in practical settings such as junior doctor allocation as noted above.

Extensions of HR

One key extension of HR that has considerable practical importance arises when an instance may involve a set of *couples*, each of which submits a joint preference list over pairs of hospitals (typically in order that the members of the couple can be located geographically close to one another). The extension of HR in which couples may be involved is denoted by HRC; the stability definition in HRC is a natural extension of that in HR (see [15, Section 5.3] for a formal definition of HRC). It is known that an instance of HRC need not admit a stable matching (see [4]). Moreover, the problem of deciding whether an HRC instance admits a stable matching is NP-complete [17].

HR may be regarded as a many-one generalization of SM. A further generalization of SM is to a many-many stable matching problem, in which both residents and hospitals may be multiply assigned subject to capacity constraints. In this case, residents and hospitals are more commonly referred to as *workers* and *firms*, respectively. There are two basic variations of the many-many stable matching problem according to whether workers rank (i) individual acceptable

firms in order of preference and vice versa or (ii) acceptable *subsets* of firms in order of preference and vice versa. Previous work relating to both models is surveyed in [15, Section 5.4].

Other variants of HR may be obtained if preference lists include ties. This extension is again important from a practical perspective, since it may be unrealistic to expect a popular hospital to rank a large number of applicants in strict order, particularly if it is indifferent among groups of applicants. The extension of HR in which preference lists may include ties is denoted by HRT. In this context three natural stability definitions arise, the so-called *weak stability*, *strong stability*, and *super-stability* (see [15, Section 1.3.5] for formal definitions of these concepts). Given an instance I of HRT, it is known that weakly stable matchings may have different sizes, and the problem of finding a maximum cardinality weakly stable matching is NP-hard (see entry ▸ Stable Marriage with Ties and Incomplete Lists for further details). On the other hand, in contrast to the case for weak stability, a super-stable matching in I need not exist, though there is an $O(L)$ algorithm to find such a matching if one does [11]. Analogous results hold in the case of strong stability – in this case, an $O(L^2)$ algorithm [13] was improved by an $O(CL)$ algorithm [14] and extended to the many-many case [5]. Furthermore, counterparts of the Rural Hospitals Theorem hold for HRT under each of the super-stability and strong stability criteria [11, 19].

A further generalization of HR arises when each hospital may be split into several departments, where each department has a capacity, and residents rank individual departments in order of preference. This variant is modeled by the *Student-Project Allocation problem* [15, Section 5.5]. Finally, the *Hospitals/Residents problem under Social Stability* [2] is an extension of HR in which an instance is augmented by a *social network graph* G (a bipartite graph whose vertices correspond to residents and hospitals and whose edges form a subset of A) such that a blocking pair must additionally satisfy the property that it forms an edge of G. Edges in G correspond to resident–hospital pairs that are acquainted with one another and therefore more likely to block a matching in practice.

Open Problems

As noted in Section "Applications," ties in the hospitals' preference lists may arise naturally in practical applications. In an HRT instance, weak stability is the most commonly-studied stability criterion, due to the guaranteed existence of such a matching. Attempting to match as many residents as possible motivates the search for large weakly stable matchings. Several approximation algorithms for finding a maximum cardinality weakly stable matching have been formulated (see ▸ Stable Marriage with Ties and Incomplete Lists and [15, Section 3.2.6] for further details). It remains open to find tighter upper and lower bounds for the approximability of this problem.

URL to Code

Ada implementations of the RGS and HGS algorithms for HR may be found via the following URL: http://www.dcs.gla.ac.uk/research/algorithms/stable.

Cross-References

- ▸ Optimal Stable Marriage
- ▸ Ranked Matching
- ▸ Stable Marriage
- ▸ Stable Marriage and Discrete Convex Analysis
- ▸ Stable Marriage with Ties and Incomplete Lists
- ▸ Stable Partition Problem

Recommended Reading

1. Abdulkadiroğlu A, Pathak PA, Roth AE (2005) The New York city high school match. Am Econ Rev 95(2):364–367
2. Askalidis G, Immorlica N, Kwanashie A, Manlove DF, Pountourakis E (2013) Socially stable matchings in the Hospitals/Residents problem. In: Proceedings

of the 13th Algorithms and Data Structures Symposium (WADS'13), London, Canada. Lecture Notes in Computer Science, vol 8037. Springer, pp 85–96
3. Baïou M, Balinski M (2004) Student admissions and faculty recruitment. Theor Comput Sci 322(2):245–265
4. Biró P, Klijn F (2013) Matching with couples: a multidisciplinary survey. Int Game Theory Rev 15(2):Article number 1340008
5. Chen N, Ghosh A (2010) Strongly stable assignment. In: Proceedings of the 18th annual European Symposium on Algorithms (ESA'10), Liverpool, UK. Lecture Notes in Computer Science, vol 6347. Springer, pp 147–158
6. Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69:9–15
7. Gusfield D, Irving RW (1989) The Stable Marriage Problem: Structure and Algorithms. MIT Press, Cambridge, USA
8. http://www.nrmp.org (National Resident Matching Program website)
9. http://www.carms.ca (Canadian Resident Matching Service website)
10. http://www.jrmp.jp (Japan Resident Matching Program website)
11. Irving RW, Manlove DF, Scott S (2000) The Hospitals/Residents problem with Ties. In: Proceedings of the 7th Scandinavian Workshop on Algorithm Theory (SWAT'00), Bergen, Norway. Lecture Notes in Computer Science, vol 1851. Springer, pp 259–271
12. Irving RW, Manlove DF, Scott S (2002) Strong stability in the Hospitals/Residents problem. Technical report TR-2002-123, Department of Computing Science, University of Glasgow. Revised May 2005
13. Irving RW, Manlove DF, Sott S (2003) Strong stability in the Hospitals/Residents problem. In: Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03), Berlin, Germany. Lecture Notes in Computer Science, vol 2607. Springer, pp 439–450. Full version available as [12]
14. Kavitha T, Mehlhorn K, Michail D, Paluch KE (2007) Strongly stable matchings in time $O(nm)$ and extension to the Hospitals-Residents problem. ACM Trans Algorithms 3(2):Article number 15
15. Manlove DF (2013) Algorithmics of matching under preferences. World Scientific, Hackensack, Singapore
16. Romero-Medina A (1998) Implementation of stable solutions in a restricted matching market. Rev Econ Des 3(2):137–147
17. Ronn E (1990) NP-complete stable matching problems. J Algorithms 11:285–304
18. Roth AE, Sotomayor MAO (1990) Two-sided matching: a study in game-theoretic modeling and analysis. Econometric society monographs, vol 18. Cambridge University Press, Cambridge/New York, USA
19. Scott S (2005) A study of stable marriage problems with ties. PhD thesis, Department of Computing Science, University of Glasgow

Hospitals/Residents Problems with Quota Lower Bounds

Huang Chien-Chung
Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden

Keywords

Hospitals/Residents problem; Lower quotas; Stable matching

Years and Authors of Summarized Original Work

2010; Biró, Fleiner, Irving, Manlove;
2010; Huang;
2011; Hamada, Iwama, Miyazaki;
2012; Fleiner, Kamiyama

Problem Definition

The Hospitals/Residents (**HR**) problem is the many-to-one version of the stable marriage problem introduced by Gale and Shapley. In this problem, a bipartite graph $G = (\mathcal{R} \cup \mathcal{H}, E)$ is given. Each vertex in $\mathcal{H}$ represents a hospital and each vertex in $\mathcal{R}$ a resident. Each vertex has a preference over its neighboring vertices. Each hospital h has an upper quota $u(h)$ specifying the maximum number of residents it can take in a matching. The goal is to find a stable matching while respecting the upper quotas of the hospitals.

The original **HR** has been well studied in the past decades. A recent trend is to assume that each hospital h also comes with a lower quota $l(h)$. In this context, it is required (if possible) that a matching satisfies both the upper and the lower quotas of each hospital. The introduction

of such lower quotas is to enforce some policy in hiring or to make the outcome more fair. It is well-known that hospitals in some rural areas suffer from the shortage of doctors.

With the lower quotas, the definition of stability in **HR** and the objective of the problem depend on the applications. Below we summarize three variants that have been considered in the literature.

Minimizing the Number of Blocking Pairs

In this variant, a matching M is feasible if, for each hospital h, $l(h) \leq |M(h)| \leq u(h)$. Given a feasible matching, a resident r and a hospital h form a blocking pair if the following condition holds. (i) $(r,h) \in E \backslash M$, (ii) r is unassigned in M or r prefers h to his assignment $M(r)$, and (iii) $|M(h)| < u(h)$ or h prefers r to one of its assigned residents. A matching is stable if the number of blocking pairs is 0. It is straightforward to check whether a stable matching exists. We assume that the given instance has no stable matching and the objective is to find a matching with the minimum number of blocking pairs. We call this problem **Min-BP HR**. An alternative objective is to minimize the number of residents that are part of a blocking pair in a matching. We call this problem **Min-BR HR**.

HR with the Option of Closing a Hospital

The following variation of **HR** is motivated by the higher education system in Hungary. Instead of requiring all hospitals to have enough residents to meet their lower quotas, it is allowed that a hospital be closed as long as there is not too much demand for it.

Precisely, in this variant, a matching M is feasible if, for each hospital h, $|M(h)| = 0$ or $l(h) \leq |M(h)| \leq u(h)$. In the former case, a hospital is closed; in the latter case, a hospital is opened. Given a feasible matching M, it is stable if

1. There is no opened hospital h and resident r so that (i) $(h,r) \in E \backslash M$, (ii) r is unassigned in M or r prefers h to his assignment $M(r)$, and (iii) $|M(h)| < u(h)$ or h prefers r to one of its assigned residents;
2. There is no closed hospital h and a set $R \subseteq \mathcal{R}$ of residents so that (i) $|R| \geq |l(h)|$, (ii) for each $r \in R$, $(r,h) \in E \backslash M$, and (iii) each resident $r \in R$ is either unassigned or prefers h to his assigned hospital $M(r)$.

With the above definition of stability, we refer to the question of the existence of a stable matching as **HR woCH**.

Classified HR

Motivated by the practice in academic hiring, Huang introduced a more generalized variant of **HR**. In this variant, a hospital h has a classification $h_{\mathcal{C}}$ over its neighboring residents. Each class $C \in h_{\mathcal{C}}$ comes with a upper quota $u(C)$ and a lower quota $l(C)$. A matching M is feasible if, for each hospital h and for each of its classes $c \in h_{\mathcal{C}}$, $l(C) \leq |M(h)| \leq u(C)$. A feasible matching M is stable if the following condition holds: there is no hospital h such that

1. There exists a resident r so that $(r,h) \in E \backslash M$, and r is either unassigned in M or r prefers h to his assignment $M(r)$;
2. For every class $C \in h_{\mathcal{C}}$, $l(C) \leq |M(h) \cup \{r\}| \leq u(C)$, or there exists another resident $r' \in M(h)$ so that h prefers r to r' and for every class $C \in h_{\mathcal{C}}$, $l(C) \leq |M(h) \cup \{r\} \backslash \{r'\}| \leq u(C)$.

With the above definition of stability, we refer to the question of the existence of a stable matching as **CHR**.

Key Results

For the first variant where the objective is to minimize the number of blocking pairs, Hamada et al. showed the following tight results.

Theorem 1 ([3]) *For any positive constant $\epsilon > 0$, there is no polynomial-time $(|\mathcal{R}| + |\mathcal{H}|)^{1-\epsilon}$-approximation algorithm for **Min-BP HR** unless P=NP. This holds true even if the given bipartite graph is complete and all upper quotas are 1 and all lower quotas are 0 or 1.*

Theorem 2 ([3]) *There is a polynomial-time* $(|\mathcal{R}| + |\mathcal{H}|)$*-approximation algorithm for* ***Min-BP HR****.*

In the case that the objective is to minimize the number of residents involved in blocking pairs, Hamada et al. showed the following.

Theorem 3 ([3]) ***Min-BR HR*** *is NP-hard. This holds true even if the given bipartite graph is complete and all hospitals have the same preference over the residents.*

Theorem 4 ([3]) *There is a polynomial-time* $\sqrt{|\mathcal{R}|}$*-approximation algorithm for* ***Min-BR HR****.*

For the second variant, where a hospital is allowed to be closed, Biró et al. showed the following.

Theorem 5 ([1]) *The problem* ***HR woCH*** *is NP-complete. This holds true even if all upper quotas are at most 3.*

For the last variant where each hospital is allowed to classify the neighboring residents and sets the upper and lower quotas for each of its classes, Huang showed that if all classifications of the hospitals are laminar families, the problem is in P. Fleiner and Kamiyama later proved the same result by a significantly simpler matroid-based technique.

Theorem 6 ([2,4]) *In* ***CHR****, if all classifications of the hospitals are laminar families, then one find a stable matching or detect its absence in the given instance in* $O(nm)$ *time, where* $n = |\mathcal{R} \cup \mathcal{H}|$ *and* $m = |E|$*.*

Recommended Reading

1. Biro P, Fleiner T, Irving RW, Manlove DF (2010) The College Admissions problem with lower and common quotas. Theor Comput Sci 411(34–36):3136–3153
2. Fleiner T, Kamiyama N (2012) A matroid approach to stable matchings with lower quotas In: 23nd annual ACM-SIAM symposium on discrete algorithms, Kyoto, pp 135–142
3. Hamada K, Iwama K, Miyazaki S (2011) The Hospitals/Residents problem with quota lower bounds. In: 19th annual European symposium on algorithms, Saarbrücken, vol 411(34–36), pp 180–191
4. Huang C-C (2010) Classified stable matching. In: 21st annual ACM-SIAM symposium on discrete algorithms, Austin, pp 1235–1253

Hub Labeling (2-Hop Labeling)

Daniel Delling[1], Andrew V. Goldberg[2], and Renato F. Werneck[3]
[1]Microsoft, Silicon Valley, CA, USA
[2]Microsoft Research – Silicon Valley, Mountain View, CA, USA
[3]Microsoft Research Silicon Valley, La Avenida, CA, USA

Keywords

Distance oracles; Labeling algorithms; Shortest paths

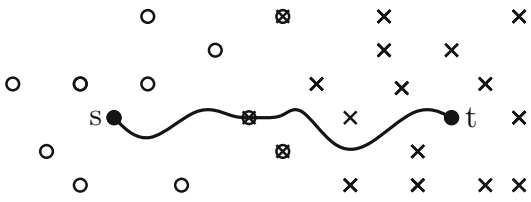
Years and Authors of Summarized Original Work

2003; Cohen, Halperin, Kaplan, Zwick
2012; Abraham, Delling, Goldberg, Werneck
2013; Akiba, Iwata, Yoshida
2014; Delling, Goldberg, Pajor, Werneck
2014; Delling, Goldberg, Savchenko, Werneck

Problem Definition

Given a directed graph $G = (V, A)$ (with $n = |V|$ and $m = |A|$) with a length function $\ell : A \to \mathbf{R}^+$ and a pair of vertices s, t, a *distance oracle* returns the distance $\text{dist}(s, t)$ from s to t. A *labeling algorithm* [18] implements distance oracles in two stages. The *preprocessing* stage computes a *label* for each vertex of the input graph. Then, given s and t, the *query* stage computes $\text{dist}(s, t)$ using only the labels of s and t; the query does not explicitly use G and ℓ.

Hub labeling (HL) (or 2-hop labeling) is a special kind of labeling algorithm. The label $L(v)$ of a vertex v consists of two parts: the *forward label* $L_f(v)$ is a collection of vertices w with their distances $\text{dist}(v, w)$ from v, while the



Hub Labeling (2-Hop Labeling), Fig. 1 Example of a hub labeling. The hubs of s are *circles*; the hubs of t are *crosses* (Taken from [3])

backward label $L_b(v)$ is a collection of vertices u with their distances $\text{dist}(u, v)$ to v. (If the graph is undirected, a single label per vertex suffices.) The vertices in v's label are the *hubs* of v. The labels must obey the *cover property*: for any two vertices s and t, the set $L_f(s) \cap L_b(t)$ must contain at least one hub that is on the shortest $s - t$ path. Given the labels, HL queries are straightforward: to find $\text{dist}(s, t)$, simply find the hub $x \in L_f(s) \cap L_b(t)$ that minimizes $\text{dist}(s, x) + \text{dist}(x, t)$ (see Fig. 1 for an example). If the hubs in each label are sorted by ID, queries consist of a simple linear sweep over the labels, as in mergesort.

The size of a forward (backward) label, $|L_f(v)|$ ($|L_b(v)|$), is the number of hubs it contains. The *size of a labeling* L is the sum of the average label sizes, $(L_f(v) + L_b(v))/2$, over all vertices. The memory footprint of the algorithm is proportional to the size of the labeling, while query times are determined by the maximum label size. Queries themselves are trivial; the hard part is an efficient implementation of a preprocessing algorithm that, given G and ℓ, computes a small hub labeling.

Key Results

We describe an approximation algorithm for finding labelings of size within $O(\log n)$ of the optimal [9], as well as its generalization to other objectives, including the maximum label size [6]. Although polynomial, these approximation algorithms do not scale to large networks. For more practical alternatives, we discuss *hierarchical hub labelings* (HHLs), a subclass of HL. We show that HHLs are closely related to vertex orderings and present efficient algorithms for computing the minimal HHL for a given ordering, as well as heuristics for finding vertex orderings that lead to small labels. In particular, the RXL algorithm uses sampling to efficiently approximate a greedy vertex order, leading to empirically small labels. RXL can handle large problems from several application domains. We then discuss representations of hub labels that allow various trade-offs between space and query time.

General Hub Labelings

The time and space efficiency of the distance oracles we discuss depend on the label size. If labels are big, HL is impractical. Gavoille et al. [15] show that there exist graphs for which general labelings must have size $\tilde{\Theta}(n^2)$. For planar graphs, they give an $\tilde{\Omega}(n^{4/3})$ lower and $\tilde{O}(n^{3/2})$ upper bound. They also show that graphs with k-separators have hub labelings of size $\tilde{O}(nk)$. Abraham et al. [1] show that graphs with small highway dimension (which they conjecture include road networks) have small hub labelings.

Given a particular graph, computing a labeling with the smallest size is NP-hard. Cohen et al. [9] developed an $O(\log n)$-approximation algorithm for the problem. Next we discuss this *general HL* (GHL) algorithm.

A *partial labeling* is a labeling that does not necessarily satisfy the cover property. Given a partial labeling $L = (L_f, L_b)$, we say that a vertex pair $[u, w]$ is *covered* if $L_f(u) \cap L_b(w)$ contains a vertex on a shortest path from u to w and *uncovered* otherwise. GHL maintains a partial labeling L (initially empty) and the corresponding set U of uncovered vertex pairs. Each iteration of the algorithm selects a vertex v and two subsets $X', Y' \subseteq V$, adds $(v, \text{dist}(x, v))$ to $L_f(x)$ for all $x \in X'$, and adds $(y, \text{dist}(v, y))$ to $L_b(y)$ for all $y \in Y'$. Then, GHL deletes from U the set $U(v, X', Y')$ of vertex pairs that become covered by this augmentation. Among all $v \in V$ and $X', Y' \subseteq V$, the triple (v, X', Y') picked in each iteration is one that maximizes $|U(v, X', Y')|/(|X'| + |Y'|)$, i.e., the ratio of the number of paths covered over the increase in label size.

Cohen et al.'s efficient implementation of GHL uses the notion of *center graphs*. Given a set U of vertex pairs and a vertex v, the center graph $G_v = (X, Y, A_v)$ is a bipartite graph with $X = Y = V$ such that an arc $(u, w) \in A_v$ if $[u, w] \in U$ and some shortest path from u to w in G go through v. If U is the set of uncovered vertex pairs, then, for a fixed vertex v, maximizing $|U(v, X', Y')|/(|X'| + |Y'|)$ over all $X', Y' \subseteq V$ is (by definition) the same as finding the vertex induced-subgraph of G_v with maximum *density* (defined as its number of arcs divided by its number of vertices). This *maximum density subgraph* (MDS) problem can be solved in polynomial time using parametric flows (see e.g., [14]). To maximize the ratio over all triples (v, X', Y'), GHL solves an MDS problem for center graphs G_v and picks the densest of the n resulting subgraphs. It then adds the corresponding vertex v^* to the labels of the vertices given by the sides of the MDS. Arcs corresponding to newly covered pairs are removed from center graphs between iterations.

Cohen et al. show that GHL is a special case of the greedy set cover algorithm [8] and thus gives an $O(\log n)$-optimal labeling. They also show that the same guarantee holds if one uses a constant-factor approximation to the MDS. We refer to a k approximation of MDS as a k-AMDS. Using a linear-time 2-AMDS algorithm by Kortsarz and Peleg [17], each GHL iteration is dominated by n AMDS computations on graphs with $O(n^2)$ arcs. Since each iteration increases the size of the labeling, the number of iterations is at most $O(n^2)$. The total running time of GHL is thus $O(n^5)$.

Delling et al. [11] improve the time bound for GHL to $O(n^3 \log n)$ using *eager* and *lazy* evaluation. Intuitively, eager evaluation finds an AMDS G' of G such that deleting G' reduces the MDS value of G by a constant factor. More precisely, given a graph G, an upper bound μ on the MDS value of G and a parameter $\alpha > 1$, *α-eager evaluation* attempts to find a (2α)-AMDS G' of G such that the MDS value of G with the arcs of G' deleted is at most μ/α. If the evaluation fails to find such G', the MDS value of G is at most μ/α. *Lazy evaluation* was introduced by Cohen et al. [9] to speed up their implementation of GHL and refined by Stengel et al. [20]. It is based on the observation that the MDS value of a center graph does not increase as the algorithm adds vertices to labels and removes arcs from center graphs.

The eager-lazy algorithm maintains upper bounds on the center subgraph densities μ_v computed in previous iterations. These values are computed during initialization and updated in a lazy fashion as follows. In each iteration, the algorithm picks the maximum μ_v and applies α-eager evaluation to G_v. If the evaluation succeeds, the labels are updated. Regardless of whether the evaluation succeeds or not, μ_v/α is a valid upper bound on the density of G_v at the end of the iteration. This can be used to show that each vertex is selected by $O(n \log n)$ iterations, each taking $O(n^2)$ time.

Babenko et al. [6] generalize the definition of a labeling size as follows. Suppose vertex IDs are $1, 2, \ldots, n$. Define a $(2n)$-dimensional vector $\mathcal{L}$ by $\mathcal{L}_{2i-1} = |L_f(i)|$ and $\mathcal{L}_{2i} = |L_b(i)|$. The p-norm of $\mathcal{L}$ is defined as $\|\mathcal{L}\|_p = (\sum_{i=0}^{2n-1} \mathcal{L}_i^p)^{1/p}$, where p is a natural number and $\|\mathcal{L}\|_\infty = \max \mathcal{L}_i$. Note that $\|\mathcal{L}\|_1/2$ is the total size of the labeling and $\|\mathcal{L}\|_\infty$ is the maximum label size. Babenko et al. [6] generalize the algorithm of Cohen et al. to obtain an $O(\log n)$-approximation algorithm for this more general problem in $O(n^5)$ time. Delling et al. [11] show that the eager-lazy approach yields an $O(\log n)$-approximation algorithm running in time $O(n^3 \log n \min(p, \log n))$.

Hierarchical Hub Labelings

Even with the performance improvements mentioned above, GHL requires too much time and space to work on large networks. To overcome this problem, one may use heuristics that have no known theoretical guarantees on the label size but produce small labels for large instances from a wide variety of domains. The most successful current heuristics use a restricted class of labelings called *hierarchical hub labeling* (HHL) [4]. Hierarchical labels have the cover property and implement exact distance oracles.

Given a labeling, let $v \lesssim w$ if w is a hub of $L(v)$. HL is *hierarchical* if $\lesssim$ is a partial order. (Intuitively, $v \lesssim w$ if w is "more important" than v.) We say that an HHL *respects* a given (total) order on the vertices if the partial order $\lesssim$ induced by the HHL is consistent with the order.

Consider an order defined by a permutation *rank*, with $rank(v) < rank(w)$ if v appears before (is less important than) w. The *canonical labeling* L for *rank* is defined as follows [4]. Vertex v belongs to $L_f(u)$ if and only if there exists w such that v is the highest-ranked vertex that hits $[u, w]$. Similarly, v belongs to $L_b(w)$ if and only if there exists u such that v is the highest-ranked vertex that hits $[u, w]$.

Abraham et al. [4] prove that the canonical labeling for a given vertex order *rank* is the minimum-sized labeling that respects *rank*. This suggests a two-stage approach for finding a small hierarchical hub labeling: first, find a "good" vertex order, and then compute its corresponding canonical labeling. We first discuss the latter step and then the former.

From Orderings to Labelings

We first consider how, given an order *rank*, one can compute the canonical hierarchical labeling L that respects *rank*.

The straightforward way is to just apply the definition: for every pair $[u, w]$ of vertices, find the maximum-ranked vertex on any shortest u–w path, and then add it to $L_f(u)$ and $L_b(w)$. Although polynomial, this algorithm is too slow in practice.

A faster (but still natural) algorithm is as follows [4]. Start with an empty (partial) labeling L, and process vertices from the most to least important. When processing v, for every uncovered pair $[u, w]$ that v covers, add v to $L_f(u)$ and $L_b(w)$. (In other words, add v to the labels of all end points of arcs in the center graph G_v.) Abraham et al. [4] show how to implement this in $O(mn \log n)$ time and $\Theta(n^2)$ space, which is still impractical for large instances.

When labels are not too large, a much more efficient solution is the *pruned labeling* (PL) algorithm by Akiba et al. [5]. Starting from empty labels, PL also processes vertices from the most to least important, with the iteration that processes vertex v, adding v to all relevant labels. The crucial observation is that, when processing v, one only needs to look at uncovered pairs containing v itself; if $[u, v]$ is not covered, PL adds v to $L_f(u)$; if $[v, w]$ is not covered, it adds v to $L_b(w)$. This is enough because of the subpath optimality property of the shortest paths.

To process v efficiently, PL runs two pruned Dijkstra searches [13] from v. The first search works on the forward graph (out of v) as follows. Before scanning a vertex w (with distance label $d(w)$ within the Dijkstra search), it computes a v–w distance estimate q by performing an HL query with the current partial labels. (If the labels do not intersect, set $q = \infty$.) If $q \leq d(w)$, the $[v, w]$ pair is already covered by previous hubs, so PL prunes the search (ignores w). Otherwise (if $q > d(w)$), PL adds $(v, \text{dist}(v, w))$ to $L_b(w)$ and scans w as usual. The second Dijkstra search uses the reverse graph and is pruned similarly; it adds $(v, \text{dist}(w, v))$ to $L_f(w)$ for all scanned vertices w. Note that the number of Dijkstra scans equals the size of the labeling. Since each visited vertex requires an HL query using partial labels, the running time can be quadratic in the average label size. It is easy to see that PL produces canonical labelings.

The final algorithm we discuss, due to Abraham et al. [4], computes a hierarchical hub labeling from a vertex ordering recursively. Its basic building block is the *shortcut operation* (see e.g., [16]). To shortcut a vertex v, the operation deletes v from the graph and adds arcs to ensure that the distances between the remaining vertices remain unchanged. For every pair consisting of an incoming arc (u, v) and an outgoing arc (v, w), the algorithm checks if $(u, v) \cdot (v, w)$ is the only shortest u–w path (by running a partial Dijkstra search from u or w) and, if so, adds a new arc (u, w) with length $\ell(u, w) = \ell(u, v) + \ell(v, w)$.

The recursive algorithm computes one label at a time, from the bottom up (from the least to the most important vertex). It starts by shortcutting the least important vertex v from G to get a graph

G' (same as G, but without v and its incident arcs and with the added shortcuts). It then recursively finds a labeling for G', which gives correct distances (in G) for all pairs of vertices not containing v. Then, the algorithm computes the label of v from the labels of its neighbors. We describe how to compute $L_f(v)$; $L_b(v)$ is computed similarly. The crucial observation is that any nontrivial shortest path starting at v must go through one of its neighbors. Accordingly, we initialize $L_f(v)$ with entry $(v, 0)$ (to cover the trivial path from v to itself), and then, for every neighbor w of v in G and every entry $(x, \text{dist}(w, x)) \in L_f(w)$, add $(x, \ell(v, w) + \text{dist}(w, x))$ to $L_f(v)$. If x already is a hub of v, we only keep the smallest entry for x. Finally, we prune from $L_f(v)$ the entries $(x, \ell(v, w) + \text{dist}(w, x))$ for which $\ell(v, w) + \text{dist}(w, x) > \text{dist}(v, x)$. (This can happen if the shortest path from v to x through another neighbor w' of v is shorter than the one through w.) Note that $\text{dist}(v, x)$ can be computed using the labels of v and x. In general, the shortcut operation can make the graph dense, limiting the efficiency of the bottom-up approach. On some network classes, such as road networks, the graph remains sparse and the approach scales to large problems.

Vertex Ordering Heuristics

As mentioned above, the size of the labeling is determined by the ordering. The most natural approach to capture the notion of importance is attributed to Abraham et al. [4], whose *greedy ordering algorithm* obtains good orderings on a wide class of problems. It orders vertices from the most to least important using a greedy selection rule. In each iteration, it selects as the next most important hub the vertex v that hits the most vertex pairs not covered by previously selected vertices.

When the shortest paths are unique, this can be implemented relatively efficiently. The algorithm maintains (initially full) the shortest-path trees from each vertex in the graph. The tree T_s rooted at s implicitly represents all shortest paths starting at s. The total number of descendants of a vertex v (in aggregate over all trees) is exactly the number of paths it covers. Once such a vertex v is picked as the next hub, we restore this invariant for the remaining paths by removing all of v's descendants (including v itself) from all trees. Abraham et al. [4] show how the entire greedy order can be found in $O(nm \log n)$ time. An alternative algorithm (in the same spirit) works even if the shortest paths are not unique, but takes $O(n^3)$ time [12].

The *weighted greedy ordering algorithm* is similar but selects v so as to maximize the ratio of the number of uncovered paths that v covers to the increase in the label size if v is selected next. This gives slightly better results and can be implemented in the same time bounds as the greedy ordering algorithm [4, 12]. Although faster than GHL, none of these greedy variants scale to large graphs.

To cope with this problem, Delling et al. [12] developed RXL (Robust eXact Labeling), which can be seen as a sampling version of the greedy ordering algorithm. In each iteration, RXL finds a vertex v that *approximately* maximizes the number of pairs covered. Rather than maintaining n shortest-path trees, RXL maintains shortest-path trees from a small number of roots picked uniformly at random. It estimates the coverage of v based on how many descendants it has in these trees. To reduce the bias in this estimation, the algorithm discards outliers before taking the average number of descendants. Moreover, as the original trees shrink (because some of its subtrees become covered), new subtrees (from other roots) are added. These new trees are not full, however; they are pruned from the start (using PL), ensuring the total space (and time) usage remains under control.

For certain graph classes, simpler ordering techniques can be used. Akiba et al. [5] show that ordering by degree works well on a subclass of complex networks. Abraham et al. [2,4] show that the order induced by the contraction hierarchies (CH) algorithm [16] works well on road networks and other sparse inputs. CH order vertices from the bottom up: using only local information, it determines the least important vertex, shortcuts it, and repeats the process in the remaining graph. The most relevant signals to estimate the importance of v are the arc difference (of number

of arcs removed and added if v were shortcut) and how many neighbors of v have already been shortcut.

Label Representation and Queries

Given a source s and a target t, one can compute the minimum of $\text{dist}(s,v) + \text{dist}(v,t)$ over all $v \in L_f(s) \cap L_b(t)$ in $O(|L_f(s)| + |L_f(t)|)$ time. If vertex labels are represented as arrays sorted by hub IDs, one can compute $L_f(s) \cap L_b(t)$ by a coordinated sweep of the corresponding arrays, as in mergesort. This is very cache efficient and works well when the two labels have similar sizes.

In some applications, label sizes can be very different. Assuming (without loss of generality) that $|L_f(s)| \ll |L_f(t)|$, one can compute $L_f(s) \cap L_b(t)$ in time $O(|L_f(s)| + \log(|L_b(t)|))$ by performing a binary search for each hub $v \in L_f(s)$ to determine if v is in $L_b(t)$. In fact, this *set intersection problem* can be solved even faster, in $O(\min(|L_f(s)|, |L_b(t)|))$ time [19].

As each label can be stored in a contiguous memory block, HL queries are well suited for an external memory (or even distributed) implementations, including relational databases [3] or key-value stores. In such cases, query times depend on the time to fetch two blocks of data.

For in-memory implementations of HL, storage may be a bottleneck. One can trade space for time using label compression, which interprets each label as a tree and stores common subtrees only once; this reduces space consumption by an order of magnitude, but queries become much less cache efficient [10, 12]. Another technique to reduce the space consumption is to store vertices and a constant number of their neighbors as superhubs in the labels [5]; on unweighted and undirected graphs, distances from a vertex v to all elements of a superhub can be represented compactly in difference form. This works well on some social and communication networks [5].

HL has efficient extensions to problems beyond point-to-point shortest paths, including one-to-many and via-point queries. These are important for applications in road networks, such as finding the closest points of interest, ride sharing, and path prediction [3].

Experimental Results

Even for very small (constant) sample sizes, the labels produced by RXL are typically no more than about 10 % bigger [12] than those produced by the full greedy hierarchical algorithms, which in turn are not much worse than those produced by GHL [11]. Scalability is much different, however. In a few hours in a modern CPU, GHL can only handle graphs with about 10,000 vertices [11]; for the greedy hierarchical algorithms, the practical limit is about 100,000 [4]. In contrast, as long as labels remain small, RXL scales to problems with millions of vertices [12] from a wide variety of graph classes, including meshes, grids, random geometric graphs (sensor networks), road networks, social networks, collaboration networks, and web graphs. For example, for a web graph with 18.5 million vertices and almost 300 million arcs, one can find labels with fewer than 300 hubs on average in about half a day [12]; queries then take less than 2 µs.

For some graph classes, other methods have faster preprocessing. For continental road networks with tens of millions of vertices, a hybrid approach combining weighted greedy (for the top few thousand vertices) with the CH order (for all other vertices) provides the best trade-off between preprocessing times and label size [2, 4]. On a benchmark data set representing Western Europe (about 18 million vertices, 42.5 million arcs), it takes roughly an hour to compute labels with about 70 hubs on average, leading to average query times of about 0.5 µs, roughly the time of ten random memory accesses. With additional improvements, one can further reduce query times (but not the label sizes) by half [2], making it the fastest algorithm for this application [7]. For some unweighted and undirected complex (social, communication, and collaboration) networks, simply sorting vertices by degree [5] produces labels that are not much bigger than those computed by a more sophisticated ordering technique.

Overall, RXL is the most robust method. For all instances tested in the literature, its preprocessing is never much slower than any other methods (and often much faster), and query times

are similar. In particular, CH-based ordering is too costly for large complex networks (as contraction tends to create dense graphs), and the degree-based order leads to prohibitively large labels for road networks and web graphs.

Recommended Reading

1. Abraham I, Fiat A, Goldberg AV, Werneck RF (2010) Highway dimension, shortest paths, and provably efficient algorithms. In: Proceedings of 21st ACM-SIAM symposium on discrete algorithms, Austin, pp 782–793
2. Abraham I, Delling D, Goldberg AV, Werneck RF (2011) A hub-based labeling algorithm for shortest paths on road networks. In: Proceedings of the 10th international symposium on experimental algorithms (SEA'11), Chania. Volume 6630 of Lecture notes in computer science. Springer, pp 230–241
3. Abraham I, Delling D, Fiat A, Goldberg AV, Werneck RF (2012) HLDB: location-based services in databases. In: Proceedings of the 20th ACM SIGSPATIAL international symposium on advances in geographic information systems (GIS'12), Redondo Beach. ACM, pp 339–348
4. Abraham I, Delling D, Goldberg AV, Werneck RF (2012) Hierarchical hub labelings for shortest paths. In: Proceedings of the 20th annual European symposium on algorithms (ESA'12), Ljubljana. Volume 7501 of Lecture notes in computer science. Springer, pp 24–35
5. Akiba T, Iwata Y, Yoshida Y (2013) Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data, SIGMOD'13, New York. ACM, pp 349–360
6. Babenko M, Goldberg AV, Gupta A, Nagarajan V (2013) Algorithms for hub label optimization. In: Fomin FV, Freivalds R, Kwiatkowska M, Peleg D (eds) Proceedings of 30th ICALP, Riga. Lecture notes in computer science, vol 7965. Springer, pp 69–80
7. Bast H, Delling D, Goldberg AV, Müller–Hannemann M, Pajor T, Sanders P, Wagner D, Werneck RF (2014) Route planning in transportation networks. Technical report MSR-TR-2014-4, Microsoft research
8. Chvátal V (1979) A greedy heuristic for the set-covering problem. Math Oper Res 4(3): 233–235
9. Cohen E, Halperin E, Kaplan H, Zwick U (2003) Reachability and distance queries via 2-hop labels. SIAM J Comput 32:1338–1355
10. Delling D, Goldberg AV, Werneck RF (2013) Hub label compression. In: Proceedings of the 12th international symposium on experimental algorithms (SEA'13), Rome. Volume 7933 of Lecture notes in computer science. Springer, pp 18–29
11. Delling D, Goldberg AV, Savchenko R, Werneck RF (2014) Hub labels: theory and practice. In: Proceedings of the 13th international symposium on experimental algorithms (SEA'14), Copenhagen. Lecture notes in computer science. Springer
12. Delling D, Goldberg AV, Pajor T, Werneck RF (2014, to appear) Robust distance queries on massive networks. In: Proceedings of the 22nd annual European symposium on algorithms (ESA'14), Wroclaw. Lecture notes in computer science. Springer
13. Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1: 269–271
14. Gallo G, Grigoriadis MD, Tarjan RE (1989) A fast parametric maximum flow algorithm and applications. SIAM J Comput 18:30–55
15. Gavoille C, Peleg D, Pérennes S, Raz R (2004) Distance labeling in graphs. J Algorithms 53(1): 85–112
16. Geisberger R, Sanders P, Schultes D, Vetter C (2012) Exact routing in large road networks using contraction hierarchies. Transp Sci 46(3):388–404
17. Kortsarz G, Peleg D (1994) Generating sparse 2-spanners. J Algorithms 17:222–236
18. Peleg D (2000) Proximity-preserving labeling schemes. J Graph Theory 33(3):167–176
19. Sanders P, Transier F (2007) Intersection in integer inverted indices. SIAM, Philadelphia, pp 71–83
20. Schenkel R, Theobald A, Weikum G (2004) HOPI: an efficient connection index for complex XML document collections. In: Advances in database technology – EDBT 2004. Springer, Berlin/Heidelberg, pp 237–255

Huffman Coding

Alistair Moffat
Department of Computing and Information Systems, The University of Melbourne, Melbourne, VIC, Australia

Keywords

Compression; Huffman code; Minimum-redundancy code

Years and Authors of Summarized Original Work

1952; Huffman
1976; van Leeuwen
1995; Moffat, Katajainen

Problem Definition

A sequence of n positive weights or frequencies is given, $\langle w_i > 0 \mid 0 \le i < n \rangle$, together with an output radix r, with $r = 2$ in the case of binary output strings.

Objective To determine a sequence of integral codeword lengths $\langle \ell_i \mid 0 \le i < n \rangle$ such that: (a) $\sum_{i=0}^{n-1} r^{-\ell_i} \le 1$, and (b) $C = \sum_{i=0}^{n-1} \ell_i \cdot w_i$ is minimized. Any sequence of codeword lengths $\langle \ell_i \rangle$ that satisfies these two properties describes a *minimum-redundancy code* for the weights $\langle w_i \rangle$. Once a set of minimum-redundancy codeword lengths $\langle \ell_i \rangle$ has been identified, a prefix-free r-ary code in which symbol i is assigned a codeword of length ℓ_i can always be constructed.

Constraints

1. **Long messages.** In one application, each weight w_i is the frequency of symbol i in a message M of length $m = |M| = \sum_{i=0}^{n-1} w_i$, and C is the number of symbols required by a compressed representation of M. In this application it is usual to assume that $m \gg n$.
2. **Entropy-based limit.** Define $W = \sum_{i=0}^{n-1} w_i$ to be the sum of the weights and $p_i = w_i / W$ to be the corresponding probability of symbol i. Define $H_0 = -\sum_{i=0}^{n-1} (p_i \log_2 p_i)$ to be the zero-order entropy of the distribution. Then when $r = 2$, $\lceil n H_0 \rceil \le C \le n \lceil \log_2 n \rceil$.

Key Results

A minimum-redundancy code can be identified in $O(n)$ time if the weights w_i are nondecreasing and in $O(n \log n)$ time if the weights must be sorted first.

Example Weights

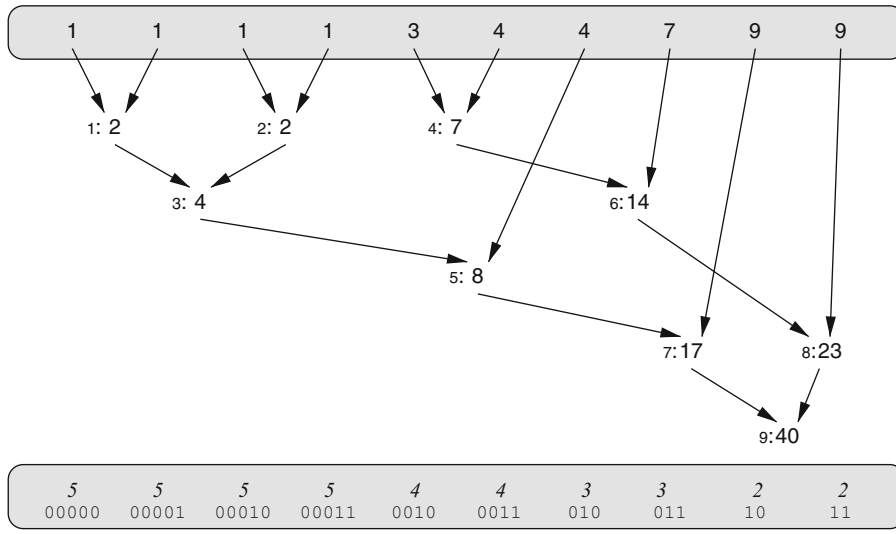
The $n = 10$ weights $\langle 1, 1, 1, 1, 3, 4, 4, 7, 9, 9 \rangle$ with $W = 40$ are used as an example.

Huffman's Algorithm

In 1952 David Huffman [3] described a process for calculating minimum-redundancy codes, developed in response to a term-paper challenge set the year before by his MIT class instructor, Robert Fano, a problem that Fano and his collaborator Claude Shannon had already tackled unsuccessfully [7]. In his solution Huffman created a classic algorithm that is taught to most undergraduate computing students as part of algorithms classes. Initially the sequence of input weights $\langle w_i \rangle$ is regarded as being the leaves of a tree, with no internal nodes, and each leaf the root of its own subtree. The two subtrees (whether singleton leaves or internal nodes) with the smallest root nodes are then combined by making both of them children of a new parent, with an assigned weight calculated as the sum of the two original nodes. The pool of subtrees decreases by one at each cycle of this process; after $n-1$ iterations a total of $n-1$ internal nodes has been added, and all of the original nodes must be leaves in a single tree and descendants of that tree's root node.

Figure 1 shows an example of codeword length computation, with the original weights across the top. Each iteration takes the two least-weight elements (leaf or internal) and combines them to make a new internal node; note that the internal nodes are created in nondecreasing weight order. Once the Huffman tree has been constructed, the sequence $\langle \ell_i \rangle$ can be read from it, by computing the depth of each corresponding leaf node. In Fig. 1, for example, one of the elements of weight 4 is at depth three in the tree, and one is at depth four from the root, hence $\ell_5 = 4$ and $\ell_6 = 3$. A set of codewords can be assigned at the same time as the depths are being computed; one possible assignment of codewords that satisfies the computed sequence $\langle \ell_i \rangle$ is shown in the second row in the lower box. Decoding throughput is considerably faster if codewords are assigned systematically based on codeword length in the manner shown, rather than by strictly following the edge labeling of the Huffman tree from which the codeword lengths were extracted [6].

Because ties can occur and can be broken arbitrarily, different codes are also possible. The sequence $\langle \ell_i \rangle = \langle 6, 6, 6, 6, 4, 3, 3, 3, 2, 2 \rangle$ has the same cost of $C = 117$ bits as the one shown



Huffman Coding, Fig. 1 Example of (binary) codeword lengths calculated using Huffman's algorithm, showing the order in which internal nodes are formed, and their weights. The input weights in the top section are used to compute the corresponding codeword lengths in the bottom box. A valid assignment of prefix-free codewords is also shown

in the figure. For the example weights, $H_0 = 2.8853$ bits per symbol, providing a lower bound of $\lceil 115.41 \rceil = 116$ bits on the total cost C for the input weights. In this case, the minimum-redundancy codes listed are just 1 bit inferior to the entropy-based lower limit.

Implementing Huffman's Algorithm

Huffman's algorithm is often used in algorithms textbooks as an example of a process that requires a dynamic priority queue. If a heap is used, for example, the n initial and $n-2$ subsequent insert operations, take a total of $O(n \log n)$ time, as do the $2(n-1)$ extract-min operations.

A simpler approach is to first sort the n weights into increasing order and then apply an $O(n)$-time algorithm due to van Leeuwen [10]. Two sorted lists are maintained: a static one of original weights, representing the leaves of the Huffman tree, and a dynamic queue of internal nodes that is initially empty, to which new internal nodes are appended as they are created. Each iteration compares front-of-list elements from the two lists and combines the two that have the least weight and then adds the new internal node at the tail of the queue. The algorithm stops when the queue contains only one node; it is the last item that was added and is the root of the Huffman tree.

If the input weights are provided in an array $w_i = A[i \mid 0 \le i < n]$ of sorted integers, that array can be processed in situ into an output array $\ell_i = A[i]$ in $O(n)$ time by van Leeuwen's technique using an implementation described by Moffat and Katajainen [5]. Each array element takes on values that are, variously, input weight, internal node weight, parent pointer, and then, finally, codeword length. Algorithm 1 is taken from Moffat and Katajainen [5] and describes this process in detail. There are three phases of operation. In the first phase, in steps 2–2, leaf weights in $A[leaf \ldots n-1]$ are combined with a queue of internal node weights in $A[root \ldots next-1]$ to form a list of parent pointers in $A[0 \ldots root-1]$. At the end of this phase, $A[0 \ldots n-3]$ is a list of parents, $A[n-2]$ is the sum of the weights, and $A[n-1]$ is unused.

In phase 2 (steps 12–3), the set of parent pointers of internal nodes is converted to a set of internal node depths. This mapping is done by processing the tree from the root down, making the depth of each node one greater than the depth of its parent.

Algorithm 1 Compute Huffman codeword lengths

```
 0: function calc_huff_lens(A, n)                    ▷ Input: A[i − 1] ≤ A[i] for 0 < i < n
 1:     // Phase 1
 2:     set leaf ← 0 and root ← 0
 3:     for next ← 0 to n − 2 do
 4:         if leaf ≥ n or (root < next and A[root] < A[leaf]) then
 5:             set A[next] ← A[root] and A[root] ← next and root ← root + 1   ▷ Use internal node
 6:         else
 7:             set A[next] ← A[leaf] and leaf ← leaf + 1                      ▷ Use leaf node
 8:         end if
 9:         repeat steps 1–8, but adding to A[next] rather than assigning to it   ▷ Find second child
10:     end for
11:     // Phase 2
12:     set A[n − 2] ← 0
13:     for next ← n − 3 downto 0 do
14:         set A[next] ← A[A[next]] + 1                   ▷ Compute depths of internal nodes
15:     end for
16:     // Phase 3
17:     set avail ← 1 and used ← 0 and depth ← 0 and root ← n − 2 and next ← n − 1
18:     while avail > 0 do
19:         while root ≥ 0 and A[root] = depth do
                                          ▷ Count internal nodes used at depth depth
20:             set used ← used + 1 and root ← root − 1
21:         end while
22:         while avail > used do          ▷ Assign as leaves any nodes that are not internal
23:             set A[next] ← d and next ← next − 1 and avail ← avail − 1
24:         end while
25:         set avail ← 2 · used and depth ← depth + 1 and used ← 0        ▷ Move to next depth
26:     end while
27:     return A                ▷ Output: A[i] is the length ℓ_i of the i th codeword
28: end function
```

Phase 3 (steps 17–4) then processes those internal node depths and converts them to a list of leaf depths. At each depth, some total number *avail* of nodes exist, being twice the number of internal nodes at the previous depth. Some number *used* of those are internal nodes; the balance must thus be leaf nodes at this depth and can be assigned as codeword lengths. Initially there is one node available at $depth = 0$, representing the root of the whole Huffman tree. Table 1 shows several snapshots of the Moffat and Katajainen code construction process when applied to the example sequence of weights.

Nonbinary Output Alphabets

The example Huffman tree developed in Fig. 1 and the process shown in Algorithm 1 assume that the output alphabet is binary. Huffman noted in his original paper that for r-ary alphabets all that is required is to add additional dummy symbols of weight zero, so as to bring the total number of symbols to be one more than a multiple of $(r - 1)$. Each merging step then combines r leaf or internal nodes to form a new root node and decreases the number of items by $r - 1$.

Dynamic Huffman Coding

Another assumption made by the processes described so far is that the symbol weights are known in advance and that the code that is computed can be *static*. This assumption can be satisfied, for example, by making a first pass over the message that is to be encoded. In a *dynamic* coding system, symbols must be coded on the fly, as soon as they are received by the encoder. To achieve this, the code must be adaptive, so that it can be altered after each symbol. Vitter [11] summarizes earlier work by Gallager [2], Knuth [4], and Cormack and Horspool [1] and describes a mechanism in which the total encoding cost, including the cost of keeping the code tree up to date, is

Huffman Coding, Table 1 Sequence of values computed by Algorithm 1 for the example weights. The first row shows the initial state of the array, with $A[i] = w_i$. Values "-2-" indicate parent pointers of internal nodes that have already been merged; italic values "*7*" indicate weights of internal nodes before being merged; values "(4)" indicate depths of internal nodes; bold values "**5**" indicate depths of leaves; and values "–" are unused

	i									
	0	1	2	3	4	5	6	7	8	9
Initial arrangement, $A[i] = w_i$	1	1	1	1	3	4	4	7	9	9
Phase 1, $root = 3, next = 5, leaf = 7$	-2-	-2-	-4-	*7*	*8*	–	–	7	9	9
Phase 1, finished, $root = 8$	-2-	-2-	-4-	-5-	-6-	-7-	-8-	-8-	*40*	–
Phase 2, $next = 4$	-2-	-2-	-4-	-5-	-6-	(2)	(1)	(1)	(0)	–
Phase 2, finished	(4)	(4)	(3)	(3)	(2)	(2)	(1)	(1)	(0)	–
Phase 3, $next = 5, avail = 4$	(4)	(4)	(3)	(3)	(2)	(2)	**3**	**3**	**2**	**2**
Final arrangement, $A[i] = \ell_i$	**5**	**5**	**5**	**5**	**4**	**4**	**3**	**3**	**2**	**2**

$O(1)$ per output bit. Turpin and Moffat [9] describe an alternative approximate algorithm that reduces the time required by a constant factor, by collecting the frequency updates into batches and allowing controlled inefficiency in the length of the coded output sequence. Their "GEO" Coding method is faster than dynamic Huffman Coding and also faster than dynamic Arithmetic Coding, which is comparable in speed to dynamic Huffman Coding, but uses less space for the dynamic frequency-counting data structure.

Applications

Minimum-redundancy codes have widespread use in data compression systems. The sequences of weights are usually conditioned according to a *model*, rather than taken as plain symbol frequency counts in the source message. The use of multiple conditioning contexts, and hence multiple codes, one per context, allows improved compression when symbols are not independent in the message, as is the case in natural language data. However, when the contexts are sufficiently specific that highly biased probability distributions arise, Arithmetic Coding will yield superior compression effectiveness.

Turpin and Moffat [8] consider several ancillary components of Huffman Coding, including methods for transmitting the description of the code to the decoder.

Cross-References

▸ Arithmetic Coding for Data Compression
▸ Compressing Integer Sequences

Recommended Reading

1. Cormack GV, Horspool RN (1984) Algorithms for adaptive Huffman codes. Inf Process Lett 18(3):159–165
2. Gallager RG (1978) Variations on a theme by Huffman. IEEE Trans Inf Theory IT-24(6):668–674
3. Huffman DA (1952) A method for the construction of minimum-redundancy codes. Proc Inst Radio Eng 40(9):1098–1101
4. Knuth DE (1985) Dynamic Huffman coding. J Algorithms 6(2):163–180
5. Moffat A, Katajainen J (1995) In-place calculation of minimum-redundancy codes. In: Proceedings of the Workshop on Algorithms and Data Structures, Kingston, pp 393–402
6. Moffat A, Turpin A (1997) On the implementation of minimum-redundancy prefix codes. IEEE Trans Commun 45(10):1200–1207
7. Stix G (1991) Profile: information theorist David A. Huffman. Sci Am 265(3):54–58. Reproduced at http://www.huffmancoding.com/my-uncle/david-bio. Accessed 15 July 2014
8. Turpin A, Moffat A (2000) Housekeeping for prefix coding. IEEE Trans Commun 48(4):622–628
9. Turpin A, Moffat A (2001) On-line adaptive canonical prefix coding with bounded compression loss. IEEE Trans Inf Theory 47(1):88–98
10. van Leeuwen J (1976) On the construction of Huffman trees. In: Proceedings of the International Conference on Automata, Languages, and Programming, Edinburgh University, Edinburgh, pp 382–410
11. Vitter JS (1987) Design and analysis of dynamic Huffman codes. J ACM 34(4):825–845

I/O-Model

Norbert Zeh[1] and Ulrich Meyer[2]
[1]Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada
[2]Department of Computer Science, Goethe University Fankfurt am Main, Fankfurt, Germany

Keywords

Disk access model (DAM); External-memory model

Years and Authors of Summarized Original Work

1988; Aggarwal, Vitter

Definition

The input/output model (I/O model) [1] views the computer as consisting of a *processor*, *internal memory* (RAM), and *external memory* (disk). See Fig. 1. The internal memory is of limited size, large enough to hold M data items. The external memory is of conceptually unlimited size and is divided into *blocks* of B consecutive data items. All computation has to happen on data in internal memory. Data is brought into internal memory and written back to external memory using *I/O operations* (I/Os), which are performed explicitly by the algorithm. Each such operation reads or writes one block of data from or to external memory. The complexity of an algorithm in this model is the number of I/Os it performs.

The *parallel disk model* (PDM) [15] is an extension of the I/O model that allows the external memory to consist of $D \geq 1$ parallel disks. See Fig. 2. In this model, a single I/O operation is capable of reading or writing up to D independent blocks, as long as each of them is stored on a different disk.

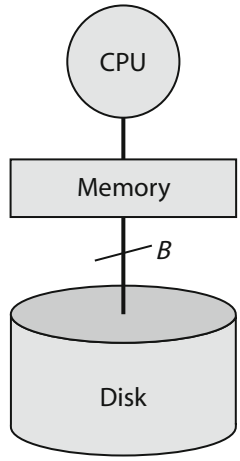
The *parallel external memory* (PEM) [5] model is a simple multiprocessor extension of the I/O model. See Fig. 3. It consists of P processing units, each having a private cache of size M. Data exchange between the processors takes place via a shared main memory of conceptually unlimited size: in a parallel I/O operation, each processor can transfer one block of size B between its private cache and the shared memory.

The relationship between the PEM model and the very popular MapReduce framework is discussed in [8]. A survey of realistic computer models can be found in [2].

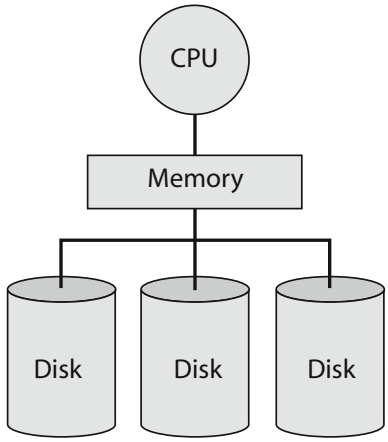
Key Results

A few complexity bounds are of importance to virtually every I/O-efficient algorithm or data structure. The *searching bound* of $\Theta(\log_B n)$ I/Os, which can be achieved using a Btree [6], is

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4



I/O-Model, Fig. 1 The I/O model



I/O-Model, Fig. 2 The parallel disk model

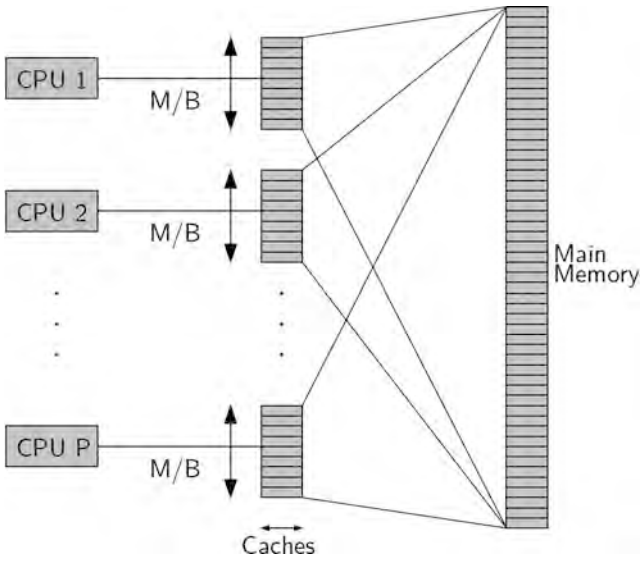
the cost of searching for an element in an ordered collection of n elements using comparisons only. It is thus the equivalent of the $\Theta(\log n)$ searching bound in internal memory.

Scanning a list of n consecutive data items obviously takes $\lceil n/B \rceil$ I/Os. This *scanning bound* is usually referred to as a "linear number of I/Os" because it is the equivalent of the $O(n)$ time bound required to do the same in internal memory. The respective PDM and PEM bounds are $\lceil n/DB \rceil$ and $\lceil n/PB \rceil$.

The *sorting bound* of $\text{sort}(n) = \Theta((n/B)\log_{M/B}(n/B))$ I/Os denotes the cost of sorting n elements using comparisons only. It is thus the equivalent of the $\Theta(n \log n)$ sorting bound in internal memory. In the PDM and PEM model, the sorting bound becomes $\Theta((n/DB)\log_{M/B}(n/B))$ and $\Theta((n/PB)\log_{M/B}(n/B))$, respectively. The sorting bound can be achieved using a range of sorting algorithms, including external merge sort [1, 5, 10] and distribution sort [1,5,9].

Arguably, the most interesting bound is the *permutation bound*, that is, the cost of rearranging n elements in a given order, which is $\Theta(\min(\text{sort}(n), n))$ [1] or, in the PDM, $\Theta(\min(\text{sort}(n), n/D))$ [15]. For all practical purposes, this is the same as the sorting bound. Note the contrast to internal memory where, up to constant factors, permuting has the same cost as a linear scan. Since almost all nontrivial algorithmic problems include a permutation problem, this implies that only exceptionally simple problems can be solved in $O(\text{scan}(n))$ I/Os; most problems have an $\Omega(\text{perm}(n))$, that is, essentially an $\Omega(\text{sort}(n))$ lower bound. Therefore, while internal-memory algorithms aiming for linear time have to carefully avoid the use of sorting as a tool, external-memory algorithms can sort without fear of significantly exceeding the lower bound. This makes the design of I/O-optimal algorithms potentially easier than the design of optimal internal-memory algorithms. It is, however, counterbalanced by the fact that, unlike in internal memory, the sorting bound is *not* equal to n times the searching bound, which implies that algorithms based on querying a tree-based search structure $O(n)$ times usually do not translate into I/O-efficient algorithms. *Buffer trees* [4] achieve an amortized search bound of $O((1/B)\log_{M/B}(N/B))$ I/Os but can be used only if the entire update and query sequence is known in advance and thus provide only a limited solution to this problem.

Apart from these fundamental results, there exist a wide range of interesting techniques, particularly for solving geometric and graph problems. For surveys, refer to [3, 14]. Also, many I/O-efficient algorithms have been derived from fast and work-efficient *parallel* algorithms; see the book entry on external-memory list ranking for a well-known example of this technique.



I/O-Model, Fig. 3 The PEM model

Applications

Modern computers are equipped with memory hierarchies consisting of several levels of cache memory, main memory (RAM), and disk(s). Access latencies increase with the distance from the processor, as do the sizes of the memory levels. To amortize these increasing access latencies, data are transferred between different levels of cache in blocks of consecutive data items. As a result, the cost of a memory access depends on the level in the memory hierarchy currently holding the data item – the difference in access latency between L1 cache and disk is about 10^6 – and the cost of a sequence of accesses to data items stored at the same level depends on the number of blocks over which these items are distributed.

Traditionally, algorithms were designed to minimize the number of computation steps; the access locality necessary to solve a problem using few data transfers between memory levels was largely ignored. Hence, the designed algorithms work well on data sets of moderate size but do not take noticeable advantage of cache memory and usually break down completely in out-of-core computations. Since the difference in access latencies is largest between main memory and disk, the I/O model focuses on minimizing this I/O bottleneck. This two-level view of the memory hierarchy keeps the model simple and useful for analyzing sophisticated algorithms while providing a good prediction of their practical performance. The picture is slightly more complex for flash memory-based solid state disks, which have recently become quite popular (also due to their *energy* efficiency [7]): not only do they internally use different block sizes for reading and writing, but their (reading) latency is also significantly smaller compared to traditional hard disks. Nevertheless, the latency gap of solid state disks compared to main memory remains large, and optimized device controllers or translation layers manage to hide the read/write discrepancy in most practical settings. Thus, the I/O model still provides reasonable estimates on flash memory, but extended models with different block sizes and access costs for reading and writing are more accurate.

Much effort has been made already to translate provably I/O-efficient algorithms into highly efficient implementations. Examples include TPIE [12] and STXXL [11], two libraries that aim to provide highly optimized and powerful primitives for the implementation of I/O-efficient algorithms. In particular, TPIE has been used to realize a number of geometric and GIS applications, whereas STXXL has served as a basis for the implementation of various graph algorithms.

In spite of these efforts, a significant gap between the theory and practice of I/O-efficient algorithms remains (see next section).

Open Problems

There are a substantial number of open problems in the area of I/O-efficient algorithms. The most important ones concern graph and geometric problems.

Traditional graph algorithms usually apply a well-organized graph traversal such as depth-first search or breadth-first search to gain information about the structure of the graph and then use this information to solve the problem at hand. For massive sparse graphs, no I/O-efficient depth-first search algorithm is known, and for breadth-first search and shortest paths, only limited progress has been made on undirected graphs. Some recent results concern dynamic and approximation variants or all-pairs shortest paths problems. For directed graphs, even such simple problems as deciding whether there exists a directed path between two vertices are currently still open. The main research focus in this area is therefore to either develop (or disprove the existence of) I/O-efficient general traversal algorithms or to continue the current strategy of devising graph algorithms that depart from traditional traversal-based approaches.

Techniques for solving geometric problems I/O efficiently are much better understood than is the case for graph algorithms, at least in two dimensions. Nevertheless, there are a few important frontiers that remain. Despite new results on some range reporting problems in three and higher dimensions, arguably the most important frontier is the development of I/O-efficient algorithms and data structures for higher-dimensional geometric problems. Motivated by database applications, results on specialized range searching variants (such as coloured and top-K range searching) have begun to appear in the literature. Little work has been done in the past on solving proximity problems, which pose another frontier currently being explored. Motivated by the need for such structures in a range of application areas and in particular in geographic information systems, there has been some recent focus on the development of multifunctional data structures, that is, structures that can answer different types of queries efficiently. This is in contrast to most existing structures, which are carefully tuned to efficiently support *one* particular type of query.

We also face a significant lack of external-memory lower bounds. Classic results concern permuting and sorting (see [14] for an overview), and more recent results concentrate on I/O-efficient data structure problems such as dynamic membership [13]. The optimality of many basic external-memory algorithms, however, is completely open. For instance, it is unclear whether sparse graph traversal (and hence probably a large number of advanced graph problems) will ever be solvable in an I/O-efficient manner.

For both I/O-efficient graph algorithms and computational geometry, there is still a substantial gap between the obtained theoretical results and what is known to be practical, even though quite some *algorithm engineering* work has been done during the last decade. Thus, if I/O-efficient algorithms in these areas are to have more practical impact, increased efforts are needed to bridge this gap by developing practically I/O-efficient algorithms that are still *provably* efficient.

Cross-References

For details on ▸ External Sorting and Permuting and ▸ List-Ranking, please refer to the corresponding entries. Details on one- and higher-dimensional searching are provided in the entries on ▸ B-trees and ▸ R-Trees. The reader interested in algorithms that focus on efficiency at all levels of the memory hierarchy should consult the entry on the ▸ Cache-Oblivious Model.

Recommended Reading

1. Aggarwal A, Vitter JS (1988) The input/output complexity of sorting and related problems. Commun ACM 31(9):1116–1127
2. Ajwani D, Meyerhenke H (2010) Realistic computer models. In: Müller-Hannemann M, Schirra S (eds) Algorithm engineering: bridging the gap between algorithm theory and practice. Volume 5971 of LNCS. Springer, Berlin/Heidelberg, pp 194–236
3. Arge L (2002) External memory data structures. In: Abello J, Pardalos PM, Resende MGC (eds) Handbook of massive data sets. Kluwer Academic, Dordrecht, pp 313–357
4. Arge L (2003) The buffer tree: a technique for designing batched external data structures. Algorithmica 37(1):1–24
5. Arge L, Goodrich MT, Nelson MJ, Sitchinava N (2008) Fundamental parallel algorithms for private-cache chip multiprocessors. In: Proceedings of the 20th annual ACM symposium on parallelism in algorithms and architectures, Munich, pp 197–206
6. Bayer R, McCreight E (1972) Organization of large ordered indexes. Acta Inform 1:173–189
7. Beckmann A, Meyer U, Sanders P, Singler S (2011) Energy-efficient sorting using solid state disks. Sustain Comput Inform Syst 1(2):151–163
8. Greiner G, Jacob R (2012) The efficiency of MapReduce in parallel external memory. In: Proceedings of the 10th Latin American symposium on theoretical informatic (LATIN). Volume 7256 of LNCS. Springer, Berlin/Heidelberg, pp 433–445
9. Nodine MH, Vitter JS (1993) Deterministic distribution sort in shared and distributed memory multiprocessors. In: Proceedings of the 5th annual ACM symposium on parallel algorithms and architectures, Velen, pp 120–129, June/July 1993
10. Nodine MH, Vitter JS (1995) Greed sort: an optimal sorting algorithm for multiple disks. J ACM 42(4):919–933
11. STXXL: C++ standard library for extra large data sets. http://stxxl.sourceforge.net. Accessed 23 June 2014
12. TPIE – a transparent parallel I/O-environment. http://www.madalgo.au.dk/tpie. Accessed 23 June 2014
13. Verbin E, Zhang Q (2013) The limits of buffering: a tight lower bound for dynamic membership in the external memory model. SIAM J Comput 42(1):212–229
14. Vitter JS (2006) Algorithms and data structures for external memory. Found Trends Theor Comput Sci 2(4):305–474
15. Vitter JS, Shriver EAM (1994) Algorithms for parallel memory I: two-level memories. Algorithmica 12(2–3):110–147

Implementation Challenge for Shortest Paths

Camil Demetrescu[1,2], Andrew V. Goldberg[3], and David S. Johnson[4,5]

[1]Department of Computer and Systems Science, University of Rome, Rome, Italy

[2]Department of Information and Computer Systems, University of Rome, Rome, Italy

[3]Microsoft Research – Silicon Valley, Mountain View, CA, USA

[4]Department of Computer Science, Columbia University, New York, NJ, USA

[5]AT&T Laboratories, Algorithms and Optimization Research Department, Florham Park, NJ, USA

Keywords

DIMACS; Test sets and experimental evaluation of computer programs for solving shortest path problems

Years and Authors of Summarized Original Work

2006; Demetrescu, Goldberg, Johnson

Problem Definition

DIMACS Implementation Challenges (http://dimacs.rutgers.edu/Challenges/) are scientific events devoted to assessing the practical performance of algorithms in experimental settings, fostering effective technology transfer and establishing common benchmarks for fundamental computing problems. They are organized by DIMACS, the Center for Discrete Mathematics and Theoretical Computer Science. One of the main goals of DIMACS Implementation Challenges is to address questions of determining realistic algorithm performance where worst case analysis is overly pessimistic and probabilistic models are too unrealistic: experimentation can provide guides to realistic algorithm performance where analysis fails. Experimentation also brings

algorithmic questions closer to the original problems that motivated theoretical work. It also tests many assumptions about implementation methods and data structures. It provides an opportunity to develop and test problem instances, instance generators, and other methods of testing and comparing performance of algorithms. And it is a step in technology transfer by providing leading edge implementations of algorithms for others to adapt.

The first Challenge was held in 1990–1991 and was devoted to *Network flows and Matching*. Other addressed problems included: *Maximum Clique, Graph Coloring, and Satisfiability* (1992–1993), *Parallel Algorithms for Combinatorial Problems* (1993–1994), *Fragment Assembly and Genome Rearrangements* (1994–1995), *Priority Queues, Dictionaries, and Multi-Dimensional Point Sets* (1995–1996), *Near Neighbor Searches* (1998–1999), *Semidefinite and Related Optimization Problems* (1999–2000), and *The Traveling Salesman Problem* (2000–2001).

This entry addresses the goals and the results of the *9th DIMACS Implementation Challenge*, held in 2005–2006 and focused on *Shortest Path* problems.

The 9th DIMACS Implementation Challenge: The Shortest Path Problem

Shortest path problems are among the most fundamental combinatorial optimization problems with many applications, both direct and as subroutines in other combinatorial optimization algorithms. Algorithms for these problems have been studied since the 1950s and still remain an active area of research.

One goal of this Challenge was to create a reproducible picture of the state of the art in the area of shortest path algorithms, identifying a standard set of benchmark instances and generators, as well as benchmark implementations of well-known shortest path algorithms. Another goal was to enable current researchers to compare their codes with each other, in hopes of identifying the more effective of the recent algorithmic innovations that have been proposed.

Challenge participants studied the following variants of the shortest paths problem:

- *Point to point shortest paths* [4, 5, 6, 9, 10, 11, 14]: the problem consists of answering multiple online queries about the shortest paths between pairs of vertices and/or their lengths. The most efficient solutions for this problem preprocess the graph to create a data structure that facilitates answering queries quickly.
- *External-memory shortest paths* [2]: the problem consists of finding shortest paths in a graph whose size is too large to fit in internal memory. The problem actually addressed in the Challenge was single-source shortest paths in undirected graphs with unit edge weights.
- *Parallel shortest paths* [8, 12]: the problem consists of computing shortest paths using multiple processors, with the goal of achieving good speedups over traditional sequential implementations. The problem actually addressed in the Challenge was single-source shortest paths.
- *K-shortest paths* [13, 15]: the problem consists of ranking paths between a pair of vertices by non decreasing order of their length.
- *Regular-language constrained shortest paths:* [3] the problem consists of a generalization of shortest path problems where paths must satisfy certain constraints specified by a regular language. The problems studied in the context of the Challenge were single-source and point-to-point shortest paths, with applications ranging from transportation science to databases.

The Challenge culminated in a Workshop held at the DIMACS Center at Rutgers University, Piscataway, New Jersey on November 13–14, 2006. Papers presented at the conference are available at the URL: http://www.dis.uniroma1.it/~challenge9/papers.shtml. Selected contributions are expected to appear in a book published by the American Mathematical Society in the DIMACS Book Series.

Key Results

The main results of the 9th DIMACS Implementation Challenge include:

- Definition of common file formats for several variants of the shortest path problem, both static and dynamic. These include an extension of the famous DIMACS graph file format used by several algorithmic software libraries. Formats are described at the URL: http://www.dis.uniroma1.it/~challenge9/formats.shtml.
- Definition of a common set of core input instances for evaluating shortest path algorithms.
- Definition of benchmark codes for shortest path problems.
- Experimental evaluation of state-of-the-art implementations of shortest path codes on the core input families.
- A discussion of directions for further research in the area of shortest paths, identifying problems critical in real-world applications for which efficient solutions still remain unknown.

The chief information venue about the 9th DIMACS Implementation Challenge is the website http://www.dis.uniroma1.it/~challenge9.

Applications

Shortest path problems arise naturally in a remarkable number of applications. A limited list includes transportation planning, network optimization, packet routing, image segmentation, speech recognition, document formatting, robotics, compilers, traffic information systems, and dataflow analysis. It also appears as a subproblem of several other combinatorial optimization problems such as network flows. A comprehensive discussion of applications of shortest path problems appears in [1].

Open Problems

There are several open questions related to shortest path problems, both theoretical and practical. One of the most prominent discussed at the 9th DIMACS Challenge Workshop is modeling traffic fluctuations in point-to-point shortest paths. The current fastest implementations preprocess the input graph to answer point-to-point queries efficiently, and this operation may take hours on graphs arising in large-scale road map navigation systems. A change in the traffic conditions may require rescanning the whole graph several times. Currently, no efficient technique is known for updating the preprocessing information without rebuilding it from scratch. This would have a major impact on the performance of routing software.

Data Sets

The collection of benchmark inputs of the 9th DIMACS Implementation Challenge includes both synthetic and real-world data. All graphs are strongly connected. Synthetic graphs include random graphs, grids, graphs embedded on a torus, and graphs with small-world properties. Real-world inputs consist of graphs representing the road networks of Europe and USA. Europe graphs are provided by courtesy of the PTV company, Karlsruhe, Germany, subject to signing a (no-cost) license agreement. They include the road networks of 17 European countries: AUT, BEL, CHE, CZE, DEU, DNK, ESP, FIN, FRA, GBR, IRL, ITA, LUX, NDL, NOR, PRT, SWE, with a total of about 19 million nodes and 23 million edges. USA graphs are derived from the *UA Census 2000 TIGER/Line Files* produced by the Geography Division of the US Census Bureau, Washington, DC. The TIGER/Line collection is available at: http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html. The Challenge USA core family contains a graph representing the full USA road system with about 24 million nodes and 58 million edges, plus 11 subgraphs obtained by cutting it along different bounding boxes as shown in Table 1. Graphs in the collection include also node coordinates and are given in DIMACS format.

The benchmark input package also features query generators for the single-source and point-to-point shortest path problems. For the single-source version, sources are randomly chosen. For the point-to-point problem, both random and

Implementation Challenge for Shortest Paths, Table 1 USA road networks derived from the TIGER/Line collection

Name	Description	Nodes	Arcs	Bounding box latitude (N)	Bounding box longitude (W)
USA	Full USA	23 947 347	58 333 344	–	–
CTR	Central USA	14 081 816	34 292 496	[25.0; 50.0]	[79.0; 100.0]
W	Western USA	6 262 104	15 248 146	[27.0; 50.0]	[100.0; 130.0]
E	Eastern USA	3 598 623	8 778 114	[24.0; 50.0]	[-∞; 79.0]
LKS	Great Lakes	2 758 119	6 885 658	[41.0; 50.0]	[74.0; 93.0]
CAL	California and Nevada	1 890 815	4 657 742	[32.5; 42.0]	[114.0; 125.0]
NE	Northeast USA	1 524 453	3 897 636	[39.5, 43.0]	[-∞; 76.0]
NW	Northwest USA	1 207 945	2 840 208	[42.0; 50.0]	[116.0; 126.0]
FLA	Florida	1 070 376	2 712 798	[24.0; 31.0]	[79; 87.5]
COL	Colorado	435 666	1 057 066	[37.0; 41.0]	[102.0; 109.0]
BAY	Bay Area	321 270	800 172	[37.0; 39.0]	[121; 123]
NY	New York City	264 346	733 846	[40.3; 41.3]	[73.5; 74.5]

local queries are considered. Local queries of the form (s, t) are generated by randomly picking t among the nodes with rank in $[2^i, 2^{i+1})$ in the ordering in which nodes are scanned by Dijkstra's algorithm with source s, for any parameter i. Clearly, the smaller i is, the closer nodes s and t are in the graph. Local queries are important to test how the algorithms' performance is affected by the distance between query endpoints.

The core input families of the 9th DIMACS Implementation Challenge are available at the URL: http://www.dis.uniroma1.it/~challenge9/download.shtml.

Experimental Results

One of the main goals of the Challenge was to compare different techniques and algorithmic approaches. The most popular topic was the point-to-point shortest path problem, studied by six research groups in the context of the Challenge. For this problem, participants were additionally invited to join a competition aimed at assessing the performance and the robustness of different implementations. The competition consisted of preprocessing a version of the full USA graph of Table 1 with unit edge lengths and answering a sequence of 1,000 random distance queries. The details were announced on the first day of the workshop and the results were due on the second day. To compare experimental results by different participants on different platforms, each participant ran a Dijkstra benchmark code [7] on the USA graph to do machine calibration. The final ranking was made by considering each query time divided by the time required by the benchmark code on the same platform (benchmark ratio). Other performance measures taken into account were space usage and the average number of nodes scanned by query operations.

Six point-to-point implementations were run successfully on the USA graph defined for the competition. Among them, the fastest query time was achieved by the *HH-based transit* code [14]. Results are reported in Table 2. Codes *RE* and *REAL*(16, 1) [9] were not eligible for the competition, but used by the organizers as a proof that the problem is feasible. Some other codes were not able to deal with the size of the full USA graph, or incurred runtime errors.

Experimental results for other variants of the shortest paths problem are described in the papers presented at the Challenge Workshop.

URL to Code

Generators of problem families and benchmark solvers for shortest paths problems are avail-

Implementation Challenge for Shortest Paths, Table 2 Results of the Challenge competition on the USA graph (23.9 million nodes and 58.3 million arcs) with unit arc lengths. The benchmark ratio is the average query time divided by the time required to answer a query using the Challenge Dijkstra benchmark code on the same platform. Query times and node scans are average values per query over 1000 random queries

	Preprocessing		Query		
Code	Time (minutes)	Space (MB)	Node scans	Time (ms)	Benchmark ratio
HH-based transit [14]	104	3664	n.a.	0.019	$4.78 \cdot 10^{-6}$
TRANSIT [4]	720	n.a.	n.a.	0.052	$10.77 \cdot 10^{-6}$
HH Star [6]	32	2662	1082	1.14	$287.32 \cdot 10^{-6}$
REAL(16,1) [9]	107	2435	823	1.42	$296.30 \cdot 10^{-6}$
HH with DistTab [6]	29	2101	1671	1.61	$405.77 \cdot 10^{-6}$
RE [9]	88	861	3065	2.78	$580.08 \cdot 10^{-6}$

able at the URL: http://www.dis.uniroma1.it/~challenge9/download.shtml.

Cross-References

- Engineering Algorithms for Large Network Applications
- Experimental Methods for Algorithm Analysis
- High Performance Algorithm Engineering for Large-Scale Problems
- Implementation Challenge for TSP Heuristics
- LEDA: a Library of Efficient Algorithms

Recommended Reading

1. Ahuja R, Magnanti T, Orlin J (1993) Network flows: theory, algorithms and applications. Prentice Hall, Englewood Cliffs
2. Ajwani D, Dementiev U, Meyer R, Osipov V (2006) Breadth first search on massive graphs. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway
3. Barrett C, Bissett K, Holzer M, Konjevod G, Marathe M, Wagner D (2006) Implementations of routing algorithms for transportation networks. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway
4. Bast H, Funke S, Matijevic D (2006) Transit: ultrafast shortest-path queries with linear-time preprocessing. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway
5. Delling D, Holzer M, Muller K, Schulz F, Wagner D (2006) High performance multi-level graphs. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway
6. Delling D, Sanders P, Schultes D, Wagner D (2006) Highway hierarchies star. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway
7. Dijkstra E (1959) A note on two problems in connexion with graphs. Numer Math 1:269–271
8. Edmonds N, Breuer A, Gregor D, Lumsdaine A (2006) Single source shortest paths with the parallel boost graph library. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway
9. Goldberg A, Kaplan H, Werneck R (2006) Better landmarks within reach. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway
10. Köhler E, Möhring R, Schilling H (2006) Fast point-to-point shortest path computations with arc-flags. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway
11. Lauther U (2006) An experimental evaluation of point-to-point shortest path calculation on road networks with precalculated edge-flags. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway
12. Madduri K, Bader D, Berry J, Crobak J (2006) Parallel shortest path algorithms for solving large-scale instances. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway
13. Pascoal M (2006) Implementations and empirical comparison of k shortest loopless path algorithms. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway
14. Sanders P, Schultes D (2006) Robust, almost constant time shortest-path queries in road networks. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway
15. Santos J (2006) K shortest path algorithms. In: 9th DIMACS implementation challenge workshop: shortest paths. DIMACS Center, Piscataway

Implementation Challenge for TSP Heuristics

Lyle A. McGeoch
Department of Mathematics and Computer Science, Amherst College, Amherst, MA, USA

Keywords

Concorde; Held-Karp; Lin-Kernighan; Three-opt; TSPLIB; Two-opt

Years and Authors of Summarized Original Work

2002; Johnson, McGeoch

Problem Definition

The Eighth DIMACS Implementation Challenge, sponsored by DIMACS, the Center for Discrete Mathematics and Theoretical Computer Science, concerned heuristics for the symmetric Traveling Salesman Problem. The Challenge began in June 2000 and was organized by David S. Johnson, Lyle A. McGeoch, Fred Glover and César Rego. It explored the state-of-the-art in the area of TSP heuristics, with researchers testing a wide range of implementations on a common (and diverse) set of input instances. The Challenge remained ongoing in 2007, with new results still being accepted by the organizers and posted on the Challenge website: www.research.att.com/~dsj/chtsp. A summary of the submissions through 2002 appeared in a book chapter by Johnson and McGeoch [5].

Participants tested their heuristics on four types of instances, chosen to test the robustness and scalability of different approaches:

1. The 34 instances that have at least 1000 cities in TSPLIB, the instance library maintained by Gerd Reinelt.
2. A set of 26 instances consisting of points uniformly distributed in the unit square, with sizes ranging from 1000 to 10,000,000 cities.
3. A set of 23 randomly generated clustered instances, with sizes ranging from 1000 to 316,000 cities.
4. A set of 7 instances based on random distance matrices, with sizes ranging from 1000 to 10,000 cities.

The TSPLIB instances and generators for the random instances are available on the Challenge website. In addition, the website contains a collection of instances for the asymmetric TSP problem.

For each instance upon which a heuristic was tested, the implementers reported the machine used, the tour length produced, the user time, and (if possible) memory usage. Some heuristics could not be applied to all of the instances, either because the heuristics were inherently geometric or because the instances were too large. To help facilitate timing comparisons between heuristics tested on different machines, participants ran a benchmark heuristic (provided by the organizers) on instances of different sizes. The benchmark times could then be used to normalize, at least approximately, the observed running times of the participants' heuristics.

The quality of a tour was computed from a submitted tour length in two ways: as a ratio over the optimal tour length for the instance (if known), and as a ratio over the Held-Karp (HK) lower bound for the instance. The Concorde optimization package of Applegate et al. [1] was able to find the optimum for 58 of the instances in reasonable time. Concorde was used in a second way to compute the HK lower bound for all but the three largest instances. A third algorithm, based on Lagrangian relaxation, was used to compute an approximate HK bound, a lower bound on true HK bound, for the remaining instances. The Challenge website reports on each of these three algorithms, presenting running times and a comparison of the bounds obtained for each instance.

The Challenge website permits a variety of reports to be created:

1. For each heuristic, tables can be generated with results for each instance, including tour length, tour quality, and raw and normalized running times.

2. For each instance, a table can be produced showing the tour quality and normalized running time of each heuristic.
3. For each pair of heuristics, tables and graphs can be produced that compare tour quality and running time for instances of different type and size.

Heuristics for which results were submitted to the Challenge fell into several broad categories:

Heuristics designed for speed. These heuristics – all of which target geometric instances – have running times within a small multiple of the time needed to read the input instance. Examples include the strip and spacefilling-curve heuristics. The speed requirement affects tour quality dramatically. Two of these algorithms produced tours with 14 % of the HK lower bound for a particular TSPLIB instance, but none came within 25 % on the other 89 instances.

Tour construction heuristics. These heuristics construct tours in various ways, without seeking to find improvements once a single tour passing through all cities is found. Some are simple, such as the nearest-neighbor and greedy heuristics, while others are more complex, such as the famous Christofides heuristic. These heuristics offer a number of options in trading time for tour quality, and several produce tours within 15 % of the HK lower bound on most instances in reasonable time. The best of them, a variant of Christofides, produces tours within 8 % on uniform instances but is much more time-consuming than the other algorithms.

Simple local improvement heuristics. These include the well-known two-opt and three-opt heuristics and variants of them. These heuristics outperform tour construction heuristics in terms of tour quality on most types of instances. For example, 3-opt gets within about 3 % of the HK lower bound on most uniform instances. The submissions in this category explored various implementation choices that affect the time-quality tradeoff.

Lin-Kernighan and its variants. These heuristics extend the local search neighborhood used in 3-opt. Lin-Kernighan can produce high-quality tours (for example, within 2 % of the HK lower bound on uniform instances) in reasonable time. One variant, due to Helsgaun [3], obtains tours within 1 % on a wide variety of instances, although the running time can be substantial.

Repeated local search heuristics. These heuristics are based on repeated executions of a heuristic such as Lin-Kernighan, with random kicks applied to the tour after a local optimum is found. These algorithms can yield high-quality tours at increased running time.

Heuristics that begin with repeated local search. One example is the tour-merge heuristic [2], which runs repeated local search multiple times, builds a graph containing edges found in the best tours, and does exhaustive search within the resulting graph. This approach yields the best known tours for some of the instances in the Challenge.

The submissions to the Challenge demonstrated the remarkable effectiveness of heuristics for the traveling salesman problem. They also showed that implementation details, such a choice of data structure or whether to approximate aspects of the computation, can affect running time and/or solution quality greatly. Results for a given heuristic also varied enormously depending on the type of instance to which it is applied.

URL to Code

www.research.att.com/~dsj/chtsp

Cross-References

▸ TSP-Based Curve Reconstruction

Recommended Reading

1. Applegate D, Bixby R, Chvátal V, Cook W (1998) On the solution of traveling salesman problems. Documenta Mathematica, Extra Volume Proceedings ICM, vol III. Deutsche Mathematiker-Vereinigung, Berlin, pp 645–656

2. Applegate D, Bixby R, Chvátal V, Cook W (1999) Finding tours in the TSP. Technical report 99885. Research Institute for Discrete Mathematics, Universität Bonn
3. Helsgaun K (2000) An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur J Oper Res 126(1):106–130
4. Johnson DS, McGeoch LA (1997) The traveling salesman problem: a case study. In: Aarts E, Lenstra JK (eds) Local search in combinatorial optimization. Wiley, Chichester, pp 215–310
5. Johnson DS, McGeoch LA (2002) Experimental analysis of heuristics for the STSP. In: Gutin G, Punnen AP (eds) The traveling salesman problem and its variants. Kluwer, Dordrecht, pp 369–443

Implementing Shared Registers in Asynchronous Message-Passing Systems

Eric Ruppert
Department of Computer Science and Engineering, York University, Toronto, ON, Canada

Keywords

Emulation; Simulation

Years and Authors of Summarized Original Work

1995; Attiya, Bar-Noy, Dolev

Problem Definition

A distributed system is composed of a collection of n processes which communicate with one another. Two means of interprocess communication have been heavily studied. *Message-passing systems* model computer networks where each process can send information over message channels to other processes. In *shared-memory systems*, processes communicate less directly by accessing information in shared data structures. Distributed algorithms are often easier to design for shared-memory systems because of their similarity to single-process system architectures. However, many real distributed systems are constructed as message-passing systems. Thus, a key problem in distributed computing is the implementation of shared memory in message-passing systems. Such implementations are also called simulations or emulations of shared memory.

The most fundamental type of shared data structure to implement is a *(read-write) register*, which stores a value, taken from some domain D. It is initially assigned a value from D and can be accessed by two kinds of operations, read and write(v), where $v \in D$. A register may be either *single-writer*, meaning only one process is allowed to write it, or *multi-writer*, meaning any process may write to it. Similarly, it may be either *single-reader* or *multi-reader*. Attiya and Welch [4] give a survey of how to build multi-writer, multi-reader registers from single-writer, single-reader ones.

If reads and writes are performed one at a time, they have the following effects: a read returns the value stored in the register to the invoking process, and a write(v) changes the value stored in the register to v and returns an acknowledgment, indicating that the operation is complete. When many processes apply operations concurrently, there are several ways to specify a register's behavior [14]. A single-writer register is *regular* if each read returns either the argument of the write that completed most recently before the read began or the argument of some write operation that runs concurrently with the read. (If there is no write that completes before the read begins, the read may return either the initial value of the register or the value of a concurrent write operation.) A register is *atomic* (see ▸ Linearizability) if each operation appears to take place instantaneously. More precisely, for any concurrent execution, there is a total order of the operations such that each read returns the value written by the last write that precedes it in the order (or the initial value of the register, if there is no such write).

Moreover, this total order must be consistent with the temporal order of operations: if one operation finishes before another one begins, the former must precede the latter in the total order. Atomicity is a stronger condition than regularity, but it is possible to implement atomic registers from regular ones with some complexity overhead [12].

This article describes the problem of implementing registers in an asynchronous message-passing system in which processes may experience crash failures. Each process can send a message, containing a finite string, to any other process. To make the descriptions of algorithms more uniform, it is often assumed that processes can send messages to themselves. All messages are eventually delivered. In the algorithms described below, senders wait for an acknowledgment of each message before sending the next message, so it is not necessary to assume that the message channels are first-in-first-out. The system is totally asynchronous: there is no bound on the time required for a message to be delivered to its recipient or for a process to perform a step of local computation. A process that fails by crashing stops executing its code, but other processes cannot distinguish between a process that has crashed and one that is running very slowly. (Failures of message channels [3] and more malicious kinds of process failures [15] have also been studied.)

A *t-resilient* register implementation provides programmes to be executed by processes to simulate read and write operations. These programmes can include any standard control structures and accesses to a process's local memory, as well as instructions to send a message to another process and to read the process's buffer, where incoming messages are stored. The implementation should also specify how the processes' local variables are initialized to reflect any initial value of the implemented register. In the case of a single-writer register, only one process may execute the write programme. A process may invoke the read and write programmes repeatedly, but it must wait for one invocation to complete before starting the next one. In any such execution where at most t processes crash, each of a process's invocations of the read or write programme should eventually terminate. Each read operation returns a result from the set D, and these results should satisfy regularity or atomicity.

Relevant measures of algorithm complexity include the number of messages transmitted in the system to perform an operation, the number of bits per message, and the amount of local memory required at each process. One measure of time complexity is the time needed to perform an operation, under the optimistic assumption that the time to deliver messages is bounded by Δ and local computation is instantaneous (although algorithms must work correctly even without these assumptions).

Key Results

Implementing a Regular Register

One of the core ideas for implementing shared registers in message-passing systems is a construction that implements a regular single-writer multi-reader register. It was introduced by Attiya, Bar-Noy and Dolev [3] and made more explicit by Attiya [2]. A write(v) sends the value v to all processes and waits until a majority of the processes ($\lfloor \frac{n}{2} \rfloor + 1$, including the writer itself) return an acknowledgment. A reader sends a request to all processes for their latest values. When it has received responses from a majority of processes, it picks the most recently written value among them. If a write completes before a read begins, at least one process that answers the reader has received the write's value prior to sending its response to the reader. This is because any two sets that each contain a majority of the processes must overlap. The time required by operations when delivery times are bounded is 2Δ.

This algorithm requires the reader to determine which of the values it receives is most recent. It does this using *timestamps* attached to the values. If the writer uses increasing integers as timestamps, the messages grow without bound as

the algorithm runs. Using the bounded timestamp scheme of Israeli and Li [13] instead yields the following theorem.

Theorem 1 (Attiya [2]) *There is an* $\lceil \frac{n-2}{2} \rceil$*-resilient implementation of a regular single-writer, multi-reader register in a message-passing system of n processes. The implementation uses* $\Theta(n)$ *messages per operation, with* $\Theta(n^3)$ *bits per message. The writer uses* $\Theta(n^4)$ *bits of local memory and each reader uses* $\Theta(n^3)$ *bits.*

Theorem 1 is optimal in terms of fault-tolerance. If $\lceil \frac{n}{2} \rceil$ processes can crash, the network can be partitioned into two halves of size $\lfloor \frac{n}{2} \rfloor$, with messages between the two halves delayed indefinitely. A write must terminate before any evidence of the write is propagated to the half not containing the writer, and then a read performed by a process in that half cannot return an up-to-date value. For $t \geq \lceil \frac{n}{2} \rceil$, registers can be implemented in a message-passing system only if some degree of synchrony is present in the system. The exact amount of synchrony required was studied by Delporte-Gallet et al. [6].

Theorem 1 is within a constant factor of the optimal number of messages per operation. Evidence of each write must be transmitted to at least $\lceil \frac{n}{2} \rceil - 1$ processes, requiring $\Omega(n)$ messages; otherwise this evidence could be obliterated by crashes. A write must terminate even if only $\lfloor \frac{n}{2} \rfloor + 1$ processes (including the writer) have received information about the value written, since the rest of the processes could have crashed. Thus, a read must receive information from at least $\lceil \frac{n}{2} \rceil$ processes (including itself) to ensure that it is aware of the most recent write operation.

A t-resilient implementation, for $t < \lceil \frac{n}{2} \rceil$, that uses $\Theta(t)$ messages per operation is obtained by the following adaptation. A set of $2t + 1$ processes is preselected to be data storage servers. Writes send information to the servers, and wait for $t + 1$ acknowledgments. Reads wait for responses from $t + 1$ of the servers and choose the one with the latest timestamp.

Implementing an Atomic Register

Attiya, Bar-Noy and Dolev [3] gave a construction of an atomic register in which readers forward the value they return to all processes and wait for an acknowledgment from a majority. This is done to ensure that a read does not return an older value than another read that precedes it. Using unbounded integer timestamps, this algorithm uses $\Theta(n)$ messages per operation. The time needed per operation when delivery times are bounded is 2Δ for writes and 4Δ for reads. However, their technique of bounding the timestamps increases the number of messages per operation to $\Theta(n^2)$ (and the time per operation to 12Δ). A better implementation of atomic registers with bounded message size is given by Attiya [2]. It uses the regular registers of Theorem 1 to implement atomic registers using the "handshaking" construction of Haldar and Vidyasankar [12], yielding the following result.

Theorem 2 (Attiya [2]) *There is an* $\lceil \frac{n-2}{2} \rceil$*-resilient implementation of an atomic single-writer, multi-reader register in a message-passing system of n processes. The implementation uses* $\Theta(n)$ *messages per operation, with* $\Theta(n^3)$ *bits per message. The writer uses* $\Theta(n^5)$ *bits of local memory and each reader uses* $\Theta(n^4)$ *bits.*

Since atomic registers are regular, this algorithm is optimal in terms of fault-tolerance and within a constant factor of optimal in terms of the number of messages. The time used when delivery times are bounded is at most 14Δ for writes and 18Δ for reads.

Applications

Any distributed algorithm that uses shared registers can be adapted to run in a message-passing system using the implementations described above. This approach yielded new or improved message-passing solutions for a number of problems, including randomized consensus [1], multi-writer registers [4], and snapshot objects ▸ Distributed Snapshots. The

reverse simulation is also possible, using a straightforward implementation of message channels by single-writer, single-reader registers. Thus, the two asynchronous models are equivalent, in terms of the set of problems that they can solve, assuming only a minority of processes crash. However there is some complexity overhead in using the simulations.

If a shared-memory algorithm is implemented in a message-passing system using the algorithms described here, processes must continue to operate even when the algorithm terminates, to help other processes execute their reads and writes. This cannot be avoided: if each process must stop taking steps when its algorithm terminates, there are some problems solvable with shared registers that are not solvable in the message-passing model [5].

Using a majority of processes to "validate" each read and write operation is an example of a quorum system, originally introduced for replicated data by Gifford [10]. In general, a quorum system is a collection of sets of processes, called quorums, such that every two quorums intersect. Quorum systems can also be designed to implement shared registers in other models of message-passing systems, including dynamic networks and systems with malicious failures. For examples, see [7, 9, 11, 15].

Open Problems

Although the algorithms described here are optimal in terms of fault-tolerance and message complexity, it is not known if the number of bits used in messages and local memory is optimal. The exact time needed to do reads and writes when messages are delivered within time Δ is also a topic of ongoing research. (See, for example, [8].) As mentioned above, the simulation of shared registers can be used to implement shared-memory algorithms in message-passing systems. However, because the simulation introduces considerable overhead, it is possible that some of those problems could be solved more efficiently by algorithms designed specifically for message-passing systems.

Cross-References

- ▸ Linearizability
- ▸ Quorums
- ▸ Registers

Recommended Reading

1. Aspnes J (2003) Randomized protocols for asynchronous consensus. Distrib Comput 16(2–3): 165–175
2. Attiya H (2000) Efficient and robust sharing of memory in message-passing systems. J Algorithms 34(1):109–127
3. Attiya H, Bar-Noy A, Dolev D (1995) Sharing memory robustly in message-passing systems. J ACM 42(1):124–142
4. Attiya H, Welch J (2004) Distributed computing: fundamentals, simulations and advanced topics, 2nd edn. Wiley-Interscience, Hoboken
5. Chor B, Moscovici L (1989) Solvability in asynchronous environments. In: Proceedings of the 30th symposium on foundations of computer science, pp 422–427
6. Delporte-Gallet C, Fauconnier H, Guerraoui R, Hadzilacos V, Kouznetsov P, Toueg S (2004) The weakest failure detectors to solve certain fundamental problems in distributed computing. In: Proceedings of the 23rd ACM symposium on principles of distributed computing, St. John's, 25–28 Jul 2004, pp 338–346
7. Dolev S, Gilbert S, Lynch NA, Shvartsman AA, Welch JL (2005) GeoQuorums: implementing atomic memory in mobile ad hoc networks. Distrib Comput 18(2):125–155
8. Dutta P, Guerraoui R, Levy RR, Chakraborty A (2004) How fast can a distributed atomic read be? In: Proceedings of the 23rd ACM symposium on principles of distributed computing, St. John's, 25–28 Jul 2004, pp 236–245
9. Englert B, Shvartsman AA (2000) Graceful quorum reconfiguration in a robust emulation of shared memory. In: Proceedings of the 20th IEEE international conference on distributed computing systems, Taipei, 10–13 Apr 2000, pp 454–463
10. Gifford DK (1979) Weighted voting for replicated data. In: Proceedings of the 7th ACM symposium on operating systems principles, Pacific Grove, 10–12 Dec 1979, pp 150–162
11. Gilbert S, Lynch N, Shvartsman A (2003) Rambo II: rapidly reconfigurable atomic memory for dynamic

networks. In: Proceedings of the international conference on dependable systems and networks, San Francisco, 22–25 Jun 2003, pp 259–268
12. Haldar S, Vidyasankar K (1995) Constructing 1-writer multireader multivalued atomic variables from regular variables. J ACM 42(1):186–203
13. Israeli A, Li M (1993) Bounded time-stamps. Distrib Comput 6(4):205–209
14. Lamport L (1986) On interprocess communication: part II: algorithms. Distrib Comput 1(2):86–101
15. Malkhi D, Reiter M (1998) Byzantine quorum systems. Distrib Comput 11(4):203–213

Incentive Compatible Selection

Xi Chen
Computer Science Department, Columbia University, New York, NY, USA
Computer Science and Technology, Tsinghua University, Beijing, China

Keywords

Algorithmic mechanism design; Incentive compatible ranking; Incentive compatible selection

Years and Authors of Summarized Original Work

2006; Chen, Deng, Liu

Problem Definition

Ensuring truthful evaluation of alternatives in human activities has always been an important issue throughout history. In sports, in particular, such an issue is vital and practice of the fair-play principle has been consistently put forward as a matter of foremost priority. In addition to relying on the code of ethics and professional responsibility of players and coaches, the design of game rules is an important measure in enforcing fair play.

Ranking alternatives through pairwise comparisons (or competitions) is the most common approach in sports tournaments. Its goal is to find out the "true" ordering among alternatives through complete or partial pairwise competitions [1, 3–7]. Such studies have been mainly based on the assumption that all the players play truthfully, i.e., with their maximal effort. It is, however, possible that some players form a coalition and cheat for group benefit. An interesting example can be found in [2].

Problem Description

The work of Chen, Deng, and Liu [2] considers the problem of choosing m winners out of n candidates.

Suppose a tournament is held among n players $P_n = \{p_1, \ldots p_n\}$ and m winners are expected to be selected by a selection protocol. Here a protocol $f_{n,m}$ is a predefined function (which will become clear later) to choose winners through pairwise competitions, with the intention of finding m players of highest capacity. When the tournament starts, a distinct ID in $N_n = \{1, 2, \ldots n\}$ is assigned to each player in P_n by a randomly picked indexing function $I : P_n \to N_n$. Then a match is played between each pair of players. The competition outcomes will form a graph G, whose vertex set is N_n and edges represent the results of all the matches. Finally, the graph will be treated as the input to $f_{n,m}$, and it will output a set of m winners. Now it should be clear that $f_{n,m}$ maps every possible tournament graph G to a subset (of cardinality m) of N_n.

Suppose there exists a group of bad players who play dishonestly, i.e., they might lose a match on purpose to gain overall benefit for the whole group, while the rest of the players always play truthfully, i.e., they try their best to win matches. The group of bad players gains benefit if they are able to have more winning positions than that according to the true ranking. Given knowledge of the selection protocol $f_{n,m}$, the indexing function I, and the true ranking of all players, the bad players try to find a cheating strategy that can fool the protocol and gain benefit.

The problem is discussed under two models in which the characterizations of bad players are

different. Under the *collective incentive compatible model*, bad players are willing to sacrifice themselves to win group benefit, while the ones under the *alliance incentive compatible model* only cooperate if their individual interests are well maintained in the cheating strategy.

The goal is to find an "ideal" protocol, under which players or groups of players maximize their benefits only by strictly following the fair-play principle, i.e., always play with maximal effort.

Formal Definitions

When the tournament begins, an indexing function I is randomly picked, which assigns ID $I(p) \in N_n$ to each player $p \in P_n$. Then a match is played between each pair of players, and the results are represented as a directed graph G. Finally, G is fed into the predefined selection protocol $f_{n,m}$, to produce a set of m winners $I^{-1}(W)$, where $W = f_{n,m}(G) \subset N_n$.

Notations

An indexing function I for a tournament attended by n players $P_n = \{p_1, p_2, \ldots p_n\}$ is a one-to-one correspondence from P_n to the set of IDs: $N_n = \{1, 2, \ldots n\}$. A ranking function R is a one-to-one correspondence from P_n to $\{1, 2, \ldots n\}$. $R(p)$ represents the underlying true ranking of player p among the n players. The smaller, the stronger.

A tournament graph of size n is a directed graph $G = (N_n, E)$ such that for all $i \neq j \in N_n$, either $ij \in E$ (player with ID i beats player with ID jn) or $ji \in E_n$. Let K_n denote the set of all such graphs. A selection protocol $f_{n,m}$, which chooses m winners out of n candidates, is a function from K_n to $\{S \subset N_n \text{ and } |S| = m\}$.

A tournament T_n among players P_n is a pair $T_n = (R, B)$ where R is a ranking function from P_n to N_n and $B \subset P_n$ is the group of bad players.

Definition 1 (Benefit) Given a protocol $f_{n,m}$, a tournament $T_n = (R, B)$, an indexing function I, and a tournament graph $G \in K_n$, the benefit of the group of bad players is

$$\text{Ben}(f_{n,m}, T_n, I, G) = |\{i \in f_{n,m}(G),\ I^{-1}(i) \in B\}| - |\{p \in B, R(p) \leq m\}|.$$

Given knowledge of $f_{n,m}$, T_n, and I, not every $G \in K_n$ is a feasible strategy for B: the group of bad players. First, it depends on the tournament $T_n = (R, B)$, e.g., a player $p_b \in B$ cannot win a player $p_g \notin B$ if $R(p_b) > R(p_g)$. Second, it depends on the property of bad players which is specified by the model considered. Tournament graphs, which are recognized as feasible strategies, are characterized below, for each model. The key difference is that a bad player in the alliance incentive compatible model is not willing to sacrifice his/her own winning position, while a player in the other model fights for group benefit at all costs.

Definition 2 (Feasible Strategy) Given $f_{n,m}$, $T_n = (R, B)$, and I, graph $G \in K_n$ is *c-feasible* if

1. For every two players $p_i, p_j \notin B$, if $R(p_i) < R(p_j)$, then $I(p_i)I(p_j) \in E$;
2. For all $p_g \notin B$ and $p_b \in B$, if $R(p_g) < R(p_b)$, then edge $I(p_g)I(p_b) \in E$.

Graph $G \in K_n$ is *a-feasible* if it is c-feasible and also satisfies

1. For every bad player $p \in B$, if $R(p) \leq m$, then $I(p) \in f_{n,m}(G)$.

A cheating strategy is then a feasible tournament graph G that can be employed by the group of bad players to gain positive benefit.

Definition 3 (Cheating Strategy) Given $f_{n,m}$, $T_n = (R, B)$, and I, a cheating strategy for the group of bad players under the collective incentive compatible (alliance incentive compatible) model is a graph $G \in K_n$ which is c-feasible (a-feasible) and satisfies $\text{Ben}(f_{n,m}, T_n, I, G) > 0$.

The following two problems are studied in [2]: (**1**) Is there a protocol $f_{n,m}$ such that for

all T_n and I no cheating strategy exists under the collective incentive compatible model? (**2**) Is there a protocol $f_{n,m}$ such that for all T_n and I, no cheating strategy exists under the alliance incentive compatible model?

Key Results

Definition 4 For all integers n and m such that $2 \leq m \leq n-2$, a tournament graph $G_{n,m} = (N_n, E) \in K_n$, which consists of three parts T_1, T_2, and T_3, is defined as follows:

1. $T_1 = \{1, 2, \ldots .m-2\}$.

For all $i < j \in T_1$, edge $ij \in E$;

2. $T_2 = \{m-1, m, m+1\}$.

$(m-1)m, m(m+1), (m+1)(m-1) \in E$;

3. $T_3 = \{m+2, m+3, \ldots n\}$.

For all $i < j \in T_3$, edge $ij \in E$;

4. For all $i' \in T_i$ and $j' \in T_j$ such that $i < j$, edge $i'j' \in E$.

Theorem 1 *Under the collective incentive compatible model , for every selection protocol $f_{n,m}$ with $2 \leq m \leq n-2$, if $T_n = (R, B)$ satisfies (**1**) at least one bad player ranks as high as $m-1$, (2) the ones ranked $m+1$ and $m+2$ are both bad players, and (**3**) the one ranked m is a good player, then there always exists an indexing function I such that $G_{n,m}$ is a cheating strategy.*

Theorem 2 *Under the alliance incentive compatible model , if $n - m \geq 3$, then there exists a selection protocol $f_{n,m}$ [2] such that for every tournament T_n, indexing function I, and a-feasible strategy $G \in K_n$, $\text{Ben}(f_{n,m}, T_n, I, G) \leq 0$.*

Applications

The result shows that if players are willing to sacrifice themselves, no protocol is able to prevent malicious coalitions from obtaining undeserved benefits.

The result may have potential applications in the design of output truthful mechanisms.

Open Problems

Under the collective incentive compatible model, the work of Chen, Deng, and Liu indicates that cheating strategies are available in at least 1/8 of tournaments, assuming the probability for each player to be in the bad group is 1/2. Could this bound be improved? Or could one find a good selection protocol in the sense that the number of tournaments with cheating strategies is close to this bound? On the other hand, although no ideal protocol exists in this model, does there exist any randomized protocol, under which the probability of having cheating strategies is negligible?

Cross-References

▸ Parity Games

Recommended Reading

1. Chang P, Mendonca D, Yao X, Raghavachari M (2004) An evaluation of ranking methods for multiple incomplete round-robin tournaments. In: Proceedings of the 35th annual meeting of decision sciences institute, Boston, 20–23 Nov 2004
2. Chen X, Deng X, Liu BJ (2006) On incentive compatible competitive selection protocol. In: Proceedings of the 12th annual international computing and combinatorics conference (COCOON'06), Taipei, 15–18 Aug 2006, pp 13–22
3. Harary F, Moser L (1966) The theory of round robin tournaments. Am Math Mon 73(3):231–246
4. Jech T (1983) The ranking of incomplete tournaments: a mathematician's guide to popular sports. Am Math Mon 90(4):246–266
5. Mendonca D, Raghavachari M (1999) Comparing the efficacy of ranking methods for multiple round-robin tournaments. Eur J Oper Res 123:593–605
6. Rubinstein A (1980) Ranking the participants in a tournament. SIAM J Appl Math 38(1):108–111
7. Steinhaus H (1950) Mathematical snapshots. Oxford University Press, New York

Independent Sets in Random Intersection Graphs

Sotiris Nikoletseas[1,3], Christoforos L. Raptopoulos[2,3,4], and Paul (Pavlos) Spirakis[5,6,7]
[1]Computer Engineering and Informatics Department, University of Patras, Patras, Greece
[2]Computer Science Department, University of Geneva, Geneva, Switzerland
[3]Computer Technology Institute and Press "Diophantus", Patras, Greece
[4]Research Academic Computer Technology Institute, Greece and Computer Engineering and Informatics Department, University of Patras, Patras, Greece
[5]Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece
[6]Computer Science, University of Liverpool, Liverpool, UK
[7]Computer Technology Institute (CTI), Patras, Greece

Keywords

Existence and efficient construction of independent sets of vertices in general random intersection graphs

Years and Authors of Summarized Original Work

2004; Nikoletseas, Raptopoulos, Spirakis

Problem Definition

This problem is concerned with the efficient construction of an independent set of vertices (i.e., a set of vertices with no edges between them) with maximum cardinality, when the input is an instance of the uniform random intersection graphs model. This model was introduced by Karoński, Sheinerman, and Singer-Cohen in [4] and Singer-Cohen in [10] and it is defined as follows

Definition 1 (Uniform random intersection graph) Consider a universe $M=\{1,2,\ldots,m\}$ of elements and a set of vertices $V=\{v_1,v_2,\ldots,v_n\}$. If one assigns independently to each vertex v_j, $j=1,2,\ldots,n$, a subset S_{v_j} of M by choosing each element independently with probability p and puts an edge between two vertices v_{j_1}, v_{j_2} if and only if $S_{v_{j_1}} \cap S_{v_{j_2}} \neq \emptyset$, then the resulting graph is an instance of the uniform random intersection graph $G_{n,m,p}$.

The universe M is sometimes called *label set* and its elements *labels*. Also, denote by L_l, for $l \in M$, the set of vertices that have chosen label l.

Because of the dependence of edges, this model can abstract more accurately (than the Bernoulli random graphs model $G_{n,p}$ that assumes independence of edges) many real-life applications. Furthermore, Fill, Sheinerman, and Singer-Cohen show in [3] that for some ranges of the parameters n, m, p $(m=n^{\alpha}, \alpha>6)$, the spaces $G_{n,m,p}$ and $G_{n,\hat{p}}$ are equivalent in the sense that the total variation distance between the graph random variables has limit 0. The work of Nikoletseas, Raptopoulos, and Spirakis [7] introduces two new models, namely the *general random intersection graphs model* $G_{n,m,\vec{p}}$, $\vec{p}=[p_1,p_2,\ldots,p_m]$ and the *regular random intersection graphs model* $G_{n,m,\lambda}$, $\lambda>0$ that use a different way to randomly assign labels to vertices, but the edge appearance rule remains the same. The $G_{n,m,\vec{p}}$ model is a generalization of the uniform model where each label $i \in M$ is chosen independently with probability p_i, whereas in the $G_{n,m,\lambda}$ model each vertex chooses a random subset of M with exactly λ labels.

The authors in [7] first consider the existence of independent sets of vertices of a given cardinality in general random intersection graphs and provide exact formulae for the mean and variance of the number of independent sets of vertices of cardinality k. Furthermore, they present and analyze three polynomial time (on the number of labels m and the number of vertices n) algorithms for constructing large independent sets of vertices when the input is an instance of the $G_{n,m,p}$ model. To the best knowledge of the

entry authors, this work is the first to consider algorithmic issues for these models of random graphs.

Key Results

The following theorems concern the existence of independent sets of vertices of cardinality k in general random intersection graphs. The proof of Theorem 1 uses the linearity of expectation of sums of random variables.

Theorem 1 *Let $X^{(k)}$ denote the number of independent sets of size k in a random intersection graph $G(n, m, \vec{p})$, where $\vec{p} = [p_1, p_2, \ldots, p_m]$. Then*

$$E\left[X^{(k)}\right] = \binom{n}{k} \prod_{i=1}^{m} \left((1-p_i)^k + k p_i (1-p_i)^{k-1}\right).$$

Theorem 2 *Let $X^{(k)}$ denote the number of independent sets of size k in a random intersection graph $G(n, m, \vec{p})$, where $\vec{p} = [p_1, p_2, \ldots, p_m]$. Then*

$$\mathrm{Var}\left(X^{(k)}\right) = \sum_{s=1}^{k} \binom{n}{2k-s} \binom{2k-s}{s} \left(\gamma(k,s) \frac{E\left[X^{(k)}\right]}{\binom{n}{k}} - \frac{E^2\left[X^{(k)}\right]}{\binom{n}{k}^2}\right)$$

where $E\left[X^{(k)}\right]$ is the mean number of independent sets of size k and

$$\gamma(k,s) = \prod_{i=1}^{m} \left((1-p_i)^{k-s} + (k-s) p_i (1-p_i)^{k-s-1} \left(1 - \frac{s p_i}{1+(k-1)p_i}\right)\right).$$

Theorem 2 is proved by first writing the variance as the sum of covariances and then applying a vertex contraction technique that merges several vertices into one supervertex with similar probabilistic behavior in order to compute the covariances. By using the second moment method (see [1]) one can derive thresholds for the existence of independent sets of size k.

One of the three algorithms that were proposed in [7] is presented below. The algorithm starts with V (i.e., the set of vertices of the graph) as its "candidate" independent set. In every subsequent step it chooses a label and removes from the current candidate independent set all vertices having that label in their assigned label set except for one. Because of the edge appearance rule, this ensures that after doing this for every label in M, the final candidate independent set will contain only vertices that do not have edges between them and so it will be indeed an independent set.

Algorithm:
Input: A random intersection graph $G_{n,m,p}$.
Output: An independent set of vertices A_m.

1. set $A_0 := V$; set $L := M$;
2. **for** $i = 1$ **to** m **do**
3. **begin**
4. select a random label $l_i \in L$; set $L := L - \{l_i\}$;
5. set $D_i := \{v \in A_{i-1} : l_i \in S_v\}$;
6. **if** $(|D_i| \geq 1)$ **then** select a random vertex $u \in D_i$ and set $D_i := D_i - \{u\}$;
7. set $A_i := A_{i-1} - D_i$;
8. **end**
9. **output** A_m;

The following theorem concerns the cardinality of the independent set produced by the algorithm. The analysis of the algorithm uses Wald's equation (see [9]) for sums of a random number of random variables to calculate the mean value of $|A_m|$, and also Chernoff bounds (see e.g., [6]) for concentration around the mean.

Theorem 3 *For the case $mp = \alpha \log n$, for some constant $\alpha > 1$ and $m \geq n$, and for some constant $\beta > 0$, the following hold with high probability:*

1. *If $np \to \infty$ then $|A_m| \geq (1-\beta)\frac{n}{\log n}$.*
2. *If $np \to b$ where $b > 0$ is a constant then $|A_m| \geq (1-\beta)n(1-\mathrm{e}^{-b})$.*
3. *If $np \to 0$ then $|A_m| \geq (1-\beta)n$.*

The above theorem shows that the algorithm manages to construct a quite large independent set with high probability.

Applications

First of all, note that (as proved in [5]) any graph can be transformed into an intersection graph. Thus, the random intersection graphs models can be very general. Furthermore, for some ranges of the parameters n, m, p ($m = n^{\alpha}, \alpha > 6$) the spaces $G_{n,m,p}$ and $G_{n,p}$ are equivalent (as proved by Fill, Sheinerman, and Singer-Cohen in [3], showing that in this range the total variation distance between the graph random variables has limit 0).

Second, random intersection graphs (and in particular the general intersection graphs model of [7]) may model real-life applications more accurately (compared to the $G_{n,p}$ case). In particular, such graphs can model resource allocation in networks, e.g., when network nodes (abstracted by vertices) access shared resources (abstracted by labels): the intersection graph is in fact the conflict graph of such resource allocation problems.

Other Related Work

In their work [4] Karoński et al. consider the problem of the emergence of graphs with a constant number of vertices as induced subgraphs of $G_{n,m,p}$ graphs. By observing that the $G_{n,m,p}$ model generates graphs via clique covers (for example the sets $L_l, l \in M$ constitute an obvious clique cover) they devise a natural way to use them together with the first and second moment methods in order to find thresholds for the appearance of any fixed graph H as an induced subgraph of $G_{n,m,p}$ for various values of the parameters n, m and p.

The connectivity threshold for $G_{n,m,p}$ was considered by Singer-Cohen in [10]. She studies the case $m = n^{\alpha}, \alpha > 0$ and distinguishes two cases according to the value of α. For the case $\alpha > 1$, the results look similar to the $G_{n,p}$ graphs, as the mean number of edges at the connectivity thresholds are (roughly) the same. On the other hand, for $\alpha \leq 1$ we get denser graphs in the $G_{n,m,p}$ model. Besides connectivity, [10] examines also the size of the largest clique in uniform random intersection graphs for certain values of n, m and p.

The existence of Hamilton cycles in $G_{n,m,p}$ graphs was considered by Efthymiou and Spirakis in [2]. The authors use coupling arguments to show that the threshold of appearance of Hamilton cycles is quite close to the connectivity threshold of $G_{n,m,p}$. Efficient probabilistic algorithms for finding Hamilton cycles in uniform random intersection graphs were presented by Raptopoulos and Spirakis in [8]. The analysis of those algorithms verify that they perform well w.h.p. even for values of p that are close to the connectivity threshold of $G_{n,m,p}$. Furthermore, in the same work, an expected polynomial algorithm for finding Hamilton cycles in $G_{n,m,p}$ graphs with constant p is given.

In [11] Stark gives approximations of the distribution of the degree of a fixed vertex in the $G_{n,m,p}$ model. More specifically, by applying a sieve method, the author provides an exact formula for the probability generating function of the degree of some fixed vertex and then analyzes this formula for different values of the parameters n, m and p.

Open Problems

A number of problems related to random intersection graphs remain open. Nearly all the algorithms proposed so far concerning constructing large independent sets and finding Hamilton cycles in random intersection graphs are greedy. An interesting and important line of research would be to find more sophisticated algorithms for these problems that outperform the greedy ones. Also, all these algorithms were presented and analyzed in the uniform random intersection graphs model. Very little is known about how the same algorithms would perform when their input was an instance of the general or even the regular random intersection graph models.

Of course, many classical problems concerning random graphs have not yet been studied.

One such example is the size of the minimum dominating set (i.e., a set of vertices that has the property that all vertices of the graph either belong to this set or are connected to it) in a random intersection graph. Also, what is the degree sequence of $G_{n,m,p}$ graphs? Note that this is very different from the problem addressed in [11].

Finally, notice that none of the results presented in the bibliography for general or uniform random intersection graphs carries over immediately to regular random intersection graphs. Of course, for some values of n, m, p and λ, certain graph properties shown for $G_{n,m,p}$ could also be proved for $G_{n,m,\lambda}$ by showing concentration of the number of labels chosen by any vertex via Chernoff bounds. Other than that, the fixed sizes of the sets assigned to each vertex impose more dependencies to the model.

Cross-References

▸ Hamilton Cycles in Random Intersection Graphs

Recommended Reading

1. Alon N, Spencer H (2000) The probabilistic method. Wiley
2. Efthymiou C, Spirakis P (2005) On the existence of Hamiltonian cycles in random intersection graphs. In: Proceedings of 32nd international colloquium on automata, languages and programming (ICALP). Springer, Berlin/Heidelberg, pp 690–701
3. Fill JA, Sheinerman ER, Singer-Cohen KB (2000) Random intersection graphs when m = ω(n): an equivalence theorem relating the evolution of the g(n, m, p) and g(n, p)models. Random Struct Algorithms 16(2):156–176
4. Karoński M, Scheinerman ER, Singer-Cohen KB (1999) On random intersection graphs: the subgraph problem. Adv Appl Math 8:131–159
5. Marczewski E (1945) Sur deux propriétés des classes d'ensembles. Fundam Math 33:303–307
6. Motwani R, Raghavan P (1995) Randomized algorithms. Cambridge University Press
7. Nikoletseas S, Raptopoulos C, Spirakis P (2004) The existence and efficient construction of large independent sets in general random intersection graphs. In: Proceedings of 31st international colloquium on Automata, Languages and Programming (ICALP). Springer, Berlin/Heidelberg, pp 1029–1040. Also in the Theoretical Computer Science (TCS) Journal, accepted, to appear in 2008
8. Raptopoulos C, Spirakis P (2005) Simple and efficient greedy algorithms for Hamiltonian cycles in random intersection graphs. In: Proceedings of the 16th international symposium on algorithms and computation (ISAAC). Springer, Berlin/Heidelberg, pp 493–504
9. Ross S (1995) Stochastic processes. Wiley
10. Singer-Cohen KB (1995) Random intersection graphs. Ph.D. thesis, John Hopkins University, Baltimore
11. Stark D (2004) The vertex degree distribution of random intersection graphs. Random Struct Algorithms 24:249–258

Indexed Approximate String Matching

Wing-Kin Sung
Department of Computer Science, National University of Singapore, Singapore, Singapore

Keywords

Edit distance; Hamming distance; Indexed inexact pattern matching; Indexed k-difference problem; Indexed k-mismatch problem; Suffix tree

Years and Authors of Summarized Original Work

1994; Myers
2007; Mass, Nowak
2008; Lam, Sung, Wong
2010; Chan, Lam, Sung, Tam, Wong
2010; Tsur
2011; Chan, Lam, Sung, Tam, Wong
2014; Belazzougui

Problem Definition

Consider a text $S[1 \ldots n]$ over a finite alphabet Σ. The problem is to build an index for S such that for any query pattern $P[1 \ldots m]$ and any integer $k \geq 0$, all locations in S that match P with at most k errors can be reported efficiently. If the error is measured in terms of the Hamming distance

(number of character substitutions), the problem is called k-mismatch problem. If the error is measured in terms of the edit distance (number of character substitutions, insertions, or deletions), the problem is called k-difference problem. The two problems are formally defined as follows.

Problem 1 (k-mismatch problem) Consider a text $S[1 \ldots n]$ over a finite alphabet Σ. For any pattern P and threshold k, position i is an occurrence of P if the Hamming distance between P and $S[i \ldots i']$ is less than k for some i'. The k-mismatch problem asks for an index I for S such that, for any pattern P, all occurrences of P in S can be reported efficiently.

Problem 2 (k-difference problem) Consider a text $S[1 \ldots n]$ over a finite alphabet Σ. For any pattern P and threshold k, position i is an occurrence of P if the edit distance between P and $S[i \ldots i']$ is less than k for some i'. The k-difference problem asks for an index I for S such that, for any pattern P, all occurrences of P in S can be reported efficiently.

These two problems are also called indexed inexact pattern matching problem or indexed pattern searching problem based on Hamming distance or edit distance.

The major concern of these two problems is how to achieve efficient pattern searching without using a large amount of space for storing the index.

Key Results

For indexed k-mismatch or k-difference string matching, a naive solution either requires an index of size $\Omega(n^k)$ or supports the query using $\Omega(m^k)$ time. The first non-trival solution is by Cole et al. [10]. They modify suffix tree to give an $O(n \log^k n)$-word index that supports k-difference query using $O(m + \text{occ} + \frac{1}{k!}(c \log n)^k \log\log n)$ time. After that, a number of indexes are proposed that support k-mismatch/k-difference pattern query for any $k > 0$. All these indexes are created by augmenting the suffix tree and its variants.

Tables 1 and 2 summarize the related results in the literature for $k = 1$ and $k \geq 2$. Below, the current best results are briefly summarized.

Indexed Approximate String Matching, Table 1 Known results for 1-difference matching. ϵ is some positive constant smaller than 1 and occ is the number of 1-difference occurrences in the text

Space	Running time	
$O(\lvert\Sigma\rvert n \log n)$ words in avg	$O(m + \text{occ})$	[15]
$O(\lvert\Sigma\rvert n \log n)$ words	$O(m + \text{occ})$ in avg	[15]
$O(n \log^2 n)$ words	$O(m \log n \log\log n + \text{occ})$	[1]
$O(n \log n)$ words	$O(m \log\log n + \text{occ})$	[4]
	$O(m + \text{occ} + \log n \log\log n)$	[10]
$O(n)$ words	$O(\min\{n, \lvert\Sigma\rvert m^2\} + \text{occ})$	[8]
	$O(\lvert\Sigma\rvert m \log n + \text{occ})$	[13]
	$O(n^\epsilon \log n)$	[16]
	$O(n^\epsilon)$	[17]
	$O(m + \text{occ} + \lvert\Sigma\rvert \log^3 n \log\log n)$	[5]
	$O(m + \text{occ} + \log n \log\log n)$	[6]
$O(n(\log n \log\log n)^2 \log\lvert\Sigma\rvert)$ bits	$O(m + \text{occ})$	[2]
$O(n\sqrt{\log n} \log\lvert\Sigma\rvert)$ bits	$O(\lvert\Sigma\rvert m \log\log n + \text{occ})$	[14]
$O(n \log^\epsilon n \log\lvert\Sigma\rvert)$ bits	$O(\lvert\Sigma\rvert m + \text{occ})$	[3]
$O(n \log\log n \log\lvert\Sigma\rvert)$ bits	$O((\lvert\Sigma\rvert m + \text{occ}) \log\log n)$	[3]
$O(n \log\lvert\Sigma\rvert)$ bits	$O(\lvert\Sigma\rvert m \log^2 n + \text{occ} \log n)$	[13]
	$O((\lvert\Sigma\rvert m \log\log n + \text{occ}) \log^\epsilon n)$	[14]
	$O(m + (\text{occ} + \lvert\Sigma\rvert \log^4 n \log\log n) \log^\epsilon n)$	[5]

Indexed Approximate String Matching, Table 2 Known results for k-difference matching for $k \geq 2$. c and d are some positive constants and ϵ is some positive constant smaller than 1. occ is the number of k-difference occurrences in the text

Space	Running time	
$O(n^{1+\epsilon})$ words	$O(m + \log\log n + \text{occ})$	[19]
$O(\lvert\Sigma\rvert^k n \log^k n)$ words in avg	$O(m + \text{occ})$	[15]
$O(\lvert\Sigma\rvert^k n \log^k n)$ words	$O(m + \text{occ})$ in avg	[15]
$O(n \log^k n)$ words in avg	$O(3^k m^{k+1} + \text{occ})$	[9]
$O(\frac{d^k}{k!} n \log^k n)$ words	$O(m + 3^k \text{occ} + \frac{1}{k!}(c \log n)^k \log\log n)$	[10]
$O(n \log^{k-1} n)$ words	$O(m + k^3 3^k \text{occ} + \frac{1}{k!}(c \log n)^k \log\log n)$	[5]
$O(n)$ words	$O(\min\{n, \lvert\Sigma\rvert^k m^{k+2}\} + \text{occ})$	[8]
	$O((\lvert\Sigma\rvert m)^k \max(k, \log n) + \text{occ})$	[13]
	$O(m + k^3 3^k \text{occ} + (c \log n)^{k(k+1)} \log\log n)$	[5]
	$O((2\lvert\Sigma\rvert)^{k-1} m^{k-1} \log n \log\log n + \text{occ})$	[6]
$O(n \sqrt{\log n} \log \lvert\Sigma\rvert)$ bits	$O((\lvert\Sigma\rvert m)^k (k + \log\log n) + \text{occ})$	[14]
$O(n \log \lvert\Sigma\rvert)$ bits	$O((\lvert\Sigma\rvert m)^k \max(k, \log^2 n) + \text{occ} \log n)$	[13]
	$O(((\lvert\Sigma\rvert m)^k (k + \log\log n) + \text{occ}) \log^\epsilon n)$	[14]
	$O(m + (k^3 3^k \text{occ} + (c \log n)^{k^2+2k} \log\log n) \log^\epsilon n)$	[5]

Inexact Matching When $k = 1$

For 1-mismatch and 1-difference approximate matching problem, the theorems below give the current best solutions. Both algorithms try to handle long and short patterns separately. Short patterns of size polylog(n) can be handled using index of size O(polylog(n)) space by brute force. Long patterns can be handled with the help of some augmented suffix tree.

When the index is of size $O(n \log \lvert\Sigma\rvert)$ bits, the next theorem is the current best result.

Theorem 1 (Chan, Lam, Sung, Tam, and Wong [5]) *Given an index of size $O(n \log \lvert\Sigma\rvert)$ bits, 1-mismatch or 1-difference query can be supported in $O(m + (\text{occ} + \lvert\Sigma\rvert \log^4 n \log\log n) \log^\epsilon n)$ time where ϵ is any positive constant smaller than or equal to 1.*

When we allow a bit more space, Belazzougui can further reduce the query time, as shown in the following theorem.

Theorem 2 (Belazzougui [3]) *Given an index of size $O(n \log^\epsilon n \log \lvert\Sigma\rvert)$ bits (or $O(n \log\log n \log \lvert\Sigma\rvert)$ bits, respectively), 1-mismatch/1-difference lookup can be supported in $O(\lvert\Sigma\rvert m + \text{occ})$ (or $O((\lvert\Sigma\rvert m + \text{occ}) \log\log n)$, respectively) time.*

Inexact Matching When $k \geq 2$

For k-mismatch and k-difference approximate matching problem where $k \geq 2$, existing solutions are all based on the so-called k-error suffix trees and its variants (following the idea of Cole et al.).

Some current solutions create indexes whose sizes depend on k. Theorems 3–6 summarize the current best results in this direction.

Theorem 3 (Maas and Nowak [15]) *Given an index of size $O(\lvert\Sigma\rvert^k n \log^k n)$ words, k-mismatch/k-difference lookup can be supported in $O(m + \text{occ})$ expected time.*

Theorem 4 (Maas and Nowak [15]) *Consider a uniformly and independently generated text of length n. There exists an index of size $O(\lvert\Sigma\rvert^k n \log^k n)$ words on average such that an k-mismatch/k-difference lookup query can be supported in $O(m + \text{occ})$ worst-case time.*

Theorem 5 (Chan, Lam, Sung, Tam, and Wong [5]) *Given an index of size* $O(n \log^{k-h+1} n)$ *words where* $h \leq k$, *k-mismatch lookup can be supported in* $O(m + \text{occ} + c^{k^2} \log^{\max\{kh,k+h\}} n \log\log n)$ *time where c is a positive constant. For k-difference lookup, the term* occ *becomes* $k^3 3^k \text{occ}$.

Theorem 6 (Chan, Lam, Sung, Tam, and Wong [6]) *Given an index of size* $O(n \log^{k-1} n)$ *words, k-mismatch/k-difference lookup can be supported in* $O(m + \text{occ} + \log^k n \log\log n)$ *time.*

Theorems 7–12 summarize the current best results when the index size is independent of k.

Theorem 7 (Chan, Lam, Sung, Tam, and Wong [5]) *Given an index of size* $O(n)$ *words, k-mismatch lookup can be supported in* $O(m + \text{occ} + (c \log n)^{k(k+1)} \log\log n)$ *time where c is a positive constant. For k-difference lookup, the term* occ *becomes* $k^3 3^k \text{occ}$.

Theorem 8 (Chan, Lam, Sung, Tam, and Wong [5]) *Given an index of size* $O(n \log|\Sigma|)$ *bits, k-mismatch lookup can be supported in* $O(m + (\text{occ} + (c \log n)^{k(k+2)} \log\log n) \log^{\epsilon} n)$ *time where c is a positive constant and* ϵ *is any positive constant smaller than or equal to* 1. *For k-difference lookup, the term* occ *becomes* $k^3 3^k \text{occ}$.

Theorem 9 (Lam, Sung, and Wong [14]) *Given an index of size* $O(n \sqrt{\log n} \log|\Sigma|)$ *bits, k-mismatch/k-difference lookup can be supported in* $O((|\Sigma|m)^k (k + \log\log n) + \text{occ})$ *time.*

Theorem 10 (Lam, Sung, and Wong [14]) *Given an index of size* $O(n \log|\Sigma|)$ *bits, k-mismatch/k-difference lookup can be supported in* $O(((|\Sigma|m)^k (k + \log\log n) + \text{occ}) \log^{\epsilon} n)$ *time where* ϵ *is any positive constant smaller than or equal to* 1.

Theorem 11 (Chan, Lam, Sung, Tam, and Wong [6]) *Given an index of size* $O(n)$ *words, k-mismatch/k-difference lookup can be supported in* $O((2|\Sigma|)^{k-1} m^{k-1} \log n \log\log n + \text{occ})$ *time.*

Theorem 12 (Tsur [19]) *Given an index of size* $O(n^{1+\epsilon})$ *words, k-mismatch/k-difference lookup can be supported in* $O(m + \text{occ} + \log\log n)$ *time.*

Practically Fast Inexact Matching

In addition, there are indexes which are efficient in practice for small k/m but give no worst-case complexity guarantees. Those methods are based on filtration. The basic idea is to partition the pattern into short segments and locate those short segments in the text allowing zero or a small number of errors. Those short segments help to identify candidate regions for the occurrences of the pattern. Finally, by verifying those candidate regions, all occurrences of the pattern are recovered. See [18] for a summary of those results. One of the best results based on filtration is stated in the following theorem.

Theorem 13 (Myers [16] and Navarro and Baeza-Yates [17]) *If* $k/m < 1 - O(1/\sqrt{|\Sigma|})$, *k-mismatch/k-difference search can be supported in* $O(n^{\epsilon})$ *expected time, where* ϵ *is a positive constant smaller than* 1, *with an index of size* $O(n)$ *words.*

Other methods with good performance on average include [11] and [12].

All the above approaches either try to index the strings with errors or are based on filtering. There are also solutions which use radically different approaches. For instance, there are solutions which transform approximate string searching into range queries in metric space [7].

Applications

Due to the advance in both the Internet and biological technologies, enormous text data is accumulated. For example, 60G genomic sequence data are currently available in GenBank. The data size is expected to grow exponentially.

To handle the huge data size, indexing techniques are vital to speed up the pattern matching queries. Moreover, exact pattern matching is no longer sufficient for both the Internet and bio-

logical data. For example, biological data usually contains a lot of differences due to experimental errors and due to mutation and evolution. Therefore, approximate pattern matching becomes more appropriate. This gives the motivation for developing indexing techniques that allow pattern matching with errors.

Open Problems

The complexity for indexed approximate matching is still not fully understood. A number of questions are still open. For instance, there are two open questions: (1) Given a fixed index size of $O(n)$ words, what is the best time complexity of a k-mismatch/k-difference query? (2) Fixed the k-mismatch/k-difference query time to be $O(m + \text{occ})$, what is the best space complexity of the index?

Cross-References

- ▸ Approximate String Matching
- ▸ Suffix Trees and Arrays

Recommended Reading

1. Amir A, Keselman D, Landau GM, Lewenstein M, Lewenstein N, Rodeh M (2000) Indexing and dictionary matching with one error. J Algorithms, 37(2):309–325
2. Belazzougui D (2009) Faster and space-optimal edit distance "1" dictionary. In: Proceedings of the 20th annual symposium on combinatorial pattern matching (CPM), Lille, pp 154–167
3. Belazzougui D (2014) Improved space-time tradeoffs for approximate full-text indexing with one edit error. Algorithmica. doi:10.1007/s00453-014-9873-9
4. Buchsbaum AL, Goodrich MT, Westbrook JR (2000) Range searching over tree cross products. In: Proceedings of European symposium on algorithms, Saarbrücken, pp 120–131
5. Chan H-L, Lam T-W, Sung W-K, Tam S-L, Wong S-S (2011) A linear size index for approximate pattern matching. J Discr Algorithms 9(4):358–364
6. Chan H-L, Lam T-W, Sung W-K, Tam S-L, Wong S-S (2010) Compressed indexes for approximate string matching. Algorithmica 58(2):263–281
7. Navarro G, Chávez E (2006) A metric index for approximate string matching. Theor Comput Sci 352(1–3):266–279
8. Cobbs A (1995) Fast approximate matching using suffix trees. In: Proceedings of symposium on combinatorial pattern matching, Espoo, pp 41–54
9. Coelho LP, Oliveira AL (2006) Dotted suffix trees: a structure for approximate text indexing. In: SPIRE, Glasgow, pp 329–336
10. Cole R, Gottlieb LA, Lewenstein M (2004) Dictionary matching and indexing with errors and don't cares. In: Proceedings of symposium on theory of computing, Chicago, pp 91–100
11. Epifanio C, Gabriele A, Mignosi F, Restivo A, Sciortino M (2007) Languages with mismatches. Theor Comput Sci 385(1–3):152–166
12. Gabriele A, Mignosi F, Restivo A, Sciortino M (2003) Indexing structures for approximate string matching. In: Proceedings of the 5th Italian conference on algorithms and complexity (CIAC), Rome, pp 140–151
13. Huynh TND, Hon WK, Lam TW, Sung WK (2006) Approximate string matching using compressed suffix arrays. Theor Comput Sci 352(1–3):240–249
14. Lam TW, Sung WK, Wong SS (2008) Improved approximate string matching using compressed suffix data structures. Algorithmica 51(3): 298–314
15. Maaß MG, Nowak J (2007) Text indexing with errors. J Discr Algorithms 5(4):662–681
16. Myers EG (1994) A sublinear algorithm for approximate keyword searching. Algorithmica 12: 345–374
17. Navarro G, Baeza-Yates R (2000) A hybrid indexing method for approximate string matching. J Discr Algorithms 1(1):205–209
18. Navarro G, Baeza-Yates RA, Sutinen E, Tarhio J (2001) Indexing methods for approximate string matching. IEEE Data Eng Bull 24(4):19–27
19. Tsur D (2010) Fast index for approximate string matching. J Discr Algorithms 8(4):339–345

Indexed Regular Expression Matching

Chee Yong Chan[1], Minos Garofalakis[2], and Rajeev Rastogi[3]
[1]National University of Singapore, Singapore, Singapore
[2]Technical University of Crete, Chania, Greece
[3]Amazon, Seattle, WA, USA

Keywords

Regular expression indexing; Regular expression retrieval

Years and Authors of Summarized Work

2003; Chan, Garofalakis, Rastogi

Problem Definition

Regular expressions (REs) provide an expressive and powerful formalism for capturing the structure of messages, events, and documents. Consequently, they have been used extensively in the specification of a number of languages for important application domains, including the XPath pattern language for XML documents [6] and the policy language of the *Border Gateway Protocol* (BGP) for propagating routing information between autonomous systems in the Internet [12]. Many of these applications have to manage large databases of RE specifications and need to provide an effective matching mechanism that, given an input string, quickly identifies all the REs in the database that match it. This RE retrieval problem is therefore important for a variety of software components in the middleware and networking infrastructure of the Internet.

The RE retrieval problem can be stated as follows: Given a large set S of REs over an alphabet Σ, where each RE $r \in S$ defines a regular language $L(r)$, construct a data structure on S that efficiently answers the following query: given an arbitrary input string $w \in \Sigma^*$, find the subset S_w of REs in S whose defined regular languages include the string w. More precisely, $r \in S_w$ iff $w \in L(r)$. Since S is a large, dynamic, disk-resident collection of REs, the data structure should be dynamic and provide efficient support of updates (insertions and deletions) to S. Note that this problem is the opposite of the more traditional RE search problem where $S \subseteq \Sigma^*$ is a collection of strings and the task is to efficiently find all strings in S that match an input regular expression.

Notations

An RE r over an alphabet Σ represents a subset of strings in σ^* (denoted by $L(r)$) that can be defined recursively as follows [9]: (1) the constants ϵ and $\emptyset$ are REs, where $L(\epsilon) = \{\epsilon\}$ and $L(\emptyset) = \emptyset$; (2) for any letter $a \in \sigma$, a is an RE where $L(a) = \{a\}$; (3) if r_1 and r_2 are REs, then their union, denoted by $r_1 + r_2$, is an RE where $L(r_1 + r_2) = L(r_1) \cup L(r_2)$; (4) if r_1 and r_2 are REs, then their concatenation, denoted by $r_1.r_2$, is an RE where $L(r_1.r_2) = \{s_1 s_2 \mid s_1 \in L(r_1), s_2 \in L(r_2)\}$; (5) if r is an RE, then its closure, denoted by r^*, is an RE where $L(r^*) = L(\epsilon) \cup L(r) \cup L(rr) \cup L(rrr) \cup \cdots$; and (6) if r is an RE, then a parenthesized r, denoted by (r), is an RE where $L((r)) = L(r)$. For example, if $\sigma = \{a, b, c\}$, then $(a + b).(a + b + c)^*.c$ is an RE representing the set of strings that begins with either a "a" or a "b" and ends with a "c." A string $s \in \sigma^*$ is said to match an RE r if $s \in L(r)$.

The language $L(r)$ defined by an RE r can be recognized by a *finite automaton (FA)* M that decides if an input string w is in $L(r)$ by reading each letter in w sequentially and updating its current state such that the outcome is determined by the final state reached by M after w has been processed [9]. Thus, M is an FA for r if the language accepted by M, denoted by $L(M)$, is equal to $L(r)$. An FA is classified as a *deterministic finite automaton* (DFA) if its current state is always updated to a single state; otherwise, it is a *nondeterministic finite automaton* (NFA) if its current state could refer to multiple possible states. The trade-off between a DFA and an NFA representations for an RE is that the latter is more space efficient, while the former is more time efficient for recognizing a matching string by checking a single path of state transitions. Let $|L(M)|$ denote the size of $L(M)$ and $|L_n(M)|$ denote the number of length-n strings in $L(M)$. Given a set $\mathcal{M}$ of finite automata, let $L(\mathcal{M})$ denote the language recognized by the automata in $\mathcal{M}$; i.e., $L(\mathcal{M}) = \bigcup_{M_i \in \mathcal{M}} L(M_i)$.

Key Results

The RE retrieval problem was first studied for a restricted class of REs in the context of content-based dissemination of XML documents using

XPath-based subscriptions (e.g., [1, 3, 7]), where each XPath expression is processed in terms of a collection of path expressions. While the XPath language [6] allows rich patterns with tree structure to be specified, the path expressions that it supports lack the full expressive power of REs (e.g., XPath does not permit the RE operators *, + and · to be arbitrarily nested in path expressions), and thus extending these XML-filtering techniques to handle general REs may not be straightforward. Further, all of the XPath-based methods are designed for indexing main-memory resident data. Another possible approach would be to coalesce the automata for all the REs into a single NFA and then use this structure to determine the collection of matching REs. It is unclear, however, if the performance of such an approach would be superior to a simple sequential scan over the database of REs; furthermore, it is not easy to see how such a scheme could be adapted for disk-resident RE data sets.

The first disk-based data structure that can handle the storage and retrieval of REs in their full generality is the *RE-tree* [4, 5]. Similar to the R-tree [8], an RE-tree is a dynamic, height-balanced, hierarchical index structure, where the leaf nodes contain data entries corresponding to the indexed REs, and the internal nodes contain "directory" entries that point to nodes at the next level of the index. Each leaf node entry is of the form (id, M), where id is the unique identifier of an RE r and M is a finite automaton representing r. Each internal node stores a collection of finite automata, and each node entry is of the form (M, ptr), where M is a finite automaton and ptr is a pointer to some node N (at the next level) such that the following *containment property* is satisfied: If $\mathcal{M}_N$ is the collection of automata contained in node N, then $L(\mathcal{M}_N) \subseteq L(M)$. The automaton M is referred to as the *bounding automaton* for $\mathcal{M}_N$. The containment property is key to improving the search performance of hierarchical index structures like RE-trees: if a query string w is not contained in $L(M)$, then it follows that $w \notin L(M_i)$ for all $M_i \in \mathcal{M}_N$. As a result, the entire subtree rooted at N can be pruned from the search space. Clearly, the closer $L(M)$ is to $L(\mathcal{M}_N)$, the more effective this search-space pruning will be.

In general, there are an infinite number of bounding automata for $\mathcal{M}_N$ with different degrees of precision from the least precise bounding automaton with $L(M) = \sigma^*$ to the most precise bounding automaton, referred to as the *minimal bounding automaton*, with $L(M) = L(\mathcal{M}_N)$. Since the storage space for an automaton is dependent on its complexity (in terms of the number of its states and transitions), there is a space-precision trade-off involved in the choice of a bounding automaton for each internal node entry. Thus, even though minimal bounding automata result in the best pruning due to their tightness, it may not be desirable (or even feasible) to always store minimal bounding automata in RE-trees since their space requirement can be too large (possibly exceeding the size of an index node), thus resulting in an index structure with a low fan-out. Therefore, to maintain a reasonable fan-out for RE-trees, a space constraint is imposed on the maximum number of states (denoted by α) permitted for each bounding automaton in internal RE-tree nodes. The automata stored in RE-tree nodes are, in general, NFAs with a minimum number of states. Also, for better space utilization, each individual RE-tree node is required to contain at least m entries. Thus, the RE-tree height is $O(\log_m(|S|))$.

RE-trees are conceptually similar to other hierarchical, spatial index structures, like the R-tree [8] that is designed for indexing a collection of multidimensional rectangles, where each internal entry is represented by a minimal bounding rectangle (MBR) that contains all the rectangles in the node pointed to by the entry. RE-tree search simply proceeds top-down along (possibly) multiple paths whose bounding automaton accepts the input string; RE-tree updates try to identify a "good" leaf node for insertion and can lead to node splits (or, node merges for deletions) that can propagate all the way up to the root. There is, however, a fundamental difference between the RE-tree and the R-tree in the indexed data types: regular languages typically represent *infinite* sets with no well-defined notion of spatial locality. This difference mandates the development of

novel algorithmic solutions for the core RE-tree operations. To optimize for search performance, the core RE-tree operations are designed to keep each bounding automaton M in every internal node to be as "tight" as possible. Thus, if M is the bounding automaton for $\mathcal{M}_N$, then $L(M)$ should be as close to $L(\mathcal{M}_N)$ as possible.

There are three core operations that need to be addressed in the RE-tree context: (P1) selection of an optimal insertion node, (P2) computing an optimal node split, and (P3) computing an optimal bounding automaton. The goal of (P1) is to choose an insertion path for a new RE that leads to "minimal expansion" in the bounding automaton of each internal node of the insertion path. Thus, given the collection of automata $\mathcal{M}(N)$ in an internal index node N and a new automaton M, an optimal $M_i \in \mathcal{M}(N)$ needs to be chosen to insert M such that $|L(M_i) \cap L(M)|$ is maximum. The goal of (P2), which arises when splitting a set of REs during an RE-tree node-split, is to identify a partitioning that results in the minimal amount of "covered area" in terms of the languages of the resulting partitions. More formally, given the collection of automata $\mathcal{M} = \{M_1, M_2, \ldots, M_k\}$ in an overflowed index node, find the optimal partition of $\mathcal{M}$ into two disjoint subsets $\mathcal{M}_1$ and $\mathcal{M}_2$ such that $|\mathcal{M}_1| \geq m$, $|\mathcal{M}_2| \geq m$, and $|L(\mathcal{M}_1)| + |L(\mathcal{M}_2)|$ is minimum. The goal of (P3), which arises during insertions, node-splits, or node-merges, is to identify a bounding automaton for a set of REs that does not cover too much "dead space." Thus, given a collection of automata $\mathcal{M}$, the goal is to find the optimal bounding automaton M such that the number of states of M is no more than α, $L(\mathcal{M}) \subseteq L(M)$ and $|L(M)|$ is minimum.

The objective of the above three operations is to maximize the pruning during search by keeping bounding automata tight. In (P1), the optimal automaton M_i selected (within an internal node) to accommodate a newly inserted automaton M is to maximize $|L(M_i) \cap L(M)|$. The set of automata $\mathcal{M}$ are split into two tight clusters in (P2), while in (P3), the most precise automaton (with no more than α states) is computed to cover the set of automata in $\mathcal{M}$. Note that (P3) is unique to RE-trees, while both (P1) and (P2) have their equivalents in R-trees. The heuristics solutions [2, 8] proposed for (P1) and (P2) in R-trees aim to minimize the number of visits to nodes that do not lead to any qualifying data entries. Although the minimal bounding automata in RE-trees (which correspond to regular languages) are very different from the MBRs in R-trees, the intuition behind minimizing the area of MBRs (total area or overlapping area) in R-trees should be effective for RE-trees as well. The counterpart for area in an RE-tree is $|L(M)|$, the size of the regular language for M. However, since a regular language is generally an infinite set, new measures need to be developed for the size of a regular language or for comparing the sizes of two regular languages.

One approach to compare the relative sizes of two regular languages is based on the following definition: for a pair of automata M_i and M_j, $L(M_i)$ is said to be larger than $L(M_j)$ if there exists a positive integer N such that for all $k \geq N$, $\sum_{l=1}^{k} |L_l(M_i)| \geq \sum_{l=1}^{k} |L_l(M_j)|$. Based on the above intuition, three increasingly sophisticated measures are proposed to capture the size of an infinite regular language. The *max-count measure* simply counts the number of strings in the language up to a certain size λ; i.e., $|L(M)| = \sum_{i=1}^{\lambda} |L_i(M)|$. This measure is useful for applications where the maximum length of all the REs to be indexed is known and is not too large so that λ can be set to some value slightly larger than the maximum length of the REs. A second more robust measure that is less sensitive to the λ parameter value is the *rate-of-growth measure* which is based on the intuition that a larger language grows at a faster rate than a smaller language. The size of a language is approximated by computing the rate of change of its size from one "window" of lengths to the next consecutive "window" of lengths: if λ is a length parameter that denote the start of the first window and θ is a window-size parameter, then $|L(M)| = \sum_{\lambda+\theta}^{\lambda+2\theta-1} |L_i(M)| / \sum_{\lambda}^{\lambda+\theta-1} |L_i(M)|$. As in the max-count measure, the parameters λ

and θ should be chosen to be slightly greater than the number of states of M to ensure that strings involving a substantial portion of paths, cycles, and accepting states are counted in each window. However, there are cases where the rate-of-growth measure also fails to capture the "larger than" relationship between regular languages [4]. To address some of the shortcomings of the first two metrics, a third information-theoretic measure is proposed that is based on Rissanen's minimum description length (MDL) principle [11]. The intuition is that if $L(M_i)$ is larger than $L(M_j)$, then the per-symbol cost of an MDL-based encoding of a random string in $L(M_i)$ using M_i is very likely to be higher than that of a string in $L(M_j)$ using M_j, where the per-symbol cost of encoding a string $w \in L(M)$ is the ratio of the cost of an MDL-based encoding of w using M to the length of w. More specifically, if $w = w_1.w_2.\ldots.w_n \in L(M)$ and $s_0, s_1, \ldots, s_n$ is the unique sequence of states visited by w in M, then the MDL-based encoding cost of w using M is given by $\sum_{i=0}^{n-1} \lceil \log_2(n_i) \rceil$, where each n_i denotes the number of transitions out of state s_i, and $\log_2(n_i)$ is the number of bits required to specify the transition out of state s_i. Thus, a reasonable measure for the size of a regular language $L(M)$ is the expected per-symbol cost of an MDL-based encoding for a random sample of strings in $L(M)$.

To utilize the above metrics for measuring $L(M)$, one common operation needed is the computation of $|L_n(M)|$, the number of length-n strings in $L(M)$. While $|L_n(M)|$ can be efficiently computed when M is a DFA, the problem becomes #P-complete when M is an NFA [10]. Two approaches were proposed to approximate $|L_n(M)|$ when N is an NFA [10]. The first approach is an unbiased estimator for $|L_n(M)|$, which can be efficiently computed but can have a very large standard deviation. The second approach is a more accurate randomized algorithm for approximating $|L_n(M)|$ but it is not very useful in practice due to its high time complexity of $O(n^{\log(n)})$. A more practical approximation algorithm with a time complexity of $O(n^2|M|^2 \min\{|\sigma|, |M|\})$ was proposed in [4].

The RE-tree operations (P1) and (P2) require frequent computations of $|L(M_i \cap M_j)|$ and $|L(M_i \cup M_j)|$ to be performed for pairs of automata M_i, M_j. These computations can adversely affect RE-tree performance since construction of the intersection and union automaton M can be expensive. Furthermore, since the final automaton M may have many more states than the two initial automata M_i and M_j, the cost of measuring $|L(M)|$ can be high. The performance of these computations can, however, be optimized by using sampling. Specifically, if the counts and samples for each $L(M_i)$ are available, then this information can be utilized to derive approximate counts and samples for $L(M_i \cap M_j)$ and $L(M_i \cup M_j)$ without incurring the overhead of constructing the automata $M_i \cap M_j$ and $M_i \cup M_j$ and counting their sizes. The sampling techniques used are based on the following results for approximating the sizes of and generating uniform samples of unions and intersections of arbitrary sets:

Theorem 1 (Chan, Garofalakis, Rastogi [4]) *Let r_1 and r_2 be uniform random samples of sets S_1 and S_2, respectively.*

1. $(|r_1 \cap S_2||S_1|)/|r_1|$ *is an unbiased estimator of the size of* $S_1 \cap S_2$.
2. $r_1 \cap S_2$ *is a uniform random sample of* $S_1 \cap S_2$ *with size* $|r_1 \cap S_2|$.
3. *If the sets S_1 and S_2 are disjoint, then a uniform random sample of $S_1 \cup S_2$ can be computed in $O(|r_1| + |r_2|)$ time. If S_1 and S_2 are not disjoint, then an approximate uniform random sample of $S_1 \cup S_2$ can be computed with the same time complexity.*

Applications

The RE retrieval problem also arises in the context of both XML document classification, which identifies matching DTDs for XML documents, as well as BGP routing, which assigns appropriate priorities to BGP advertisements based on their matching routing-system sequences.

Experimental Results

Experimental results with synthetic data sets [5] clearly demonstrate that the RE-tree index is significantly more effective than performing a sequential search for matching REs and, in a number of cases, outperforms sequential search by up to an order of magnitude.

Cross-References

▸ Regular Expression Matching

Recommended Reading

1. Altinel M, Franklin M (2000) Efficient filtering of XML documents for selective dissemination of information. In: Proceedings of 26th international conference on very large data bases, Cairo. Morgan Kaufmann, Missouri, pp 53–64
2. Beckmann N, Kriegel H-P, Schneider R, Seeger B (1990) The R*-tree: an efficient and robust access method for points and rectangles. In: Proceedings of the ACM international conference on management of data, Atlantic City. ACM, New York, pp 322–331
3. Chan C-Y, Felber P, Garofalakis M, Rastogi R (2002) Efficient filtering of XML documents with XPath expressions. In: Proceedings of the 18th international conference on data engineering, San Jose. IEEE Computer Society, Piscataway, pp 235–244
4. Chan C-Y, Garofalakis M, Rastogi R (2002) RE-tree: an efficient index structure for regular expressions. In: Proceedings of 28th international conference on very large data bases, Hong Kong. Morgan Kaufmann, Missouri, pp 251–262
5. Chan C-Y, Garofalakis M, Rastogi R (2003) RE-tree: an efficient index structure for regular expressions. VLDB J 12(2):102–119
6. Clark J, DeRose S (1999) XML Path Language (XPath) Version 1.0. W3C Recommendation. http://www.w3.org./TR/xpath. Accessed Nov 1999
7. Diao Y, Fischer P, Franklin M, To R (2002) YFilter: efficient and scalable filtering of XML documents. In: Proceedings of the 18th international conference on data engineering, San Jose. IEEE Computer Society, Piscataway, pp 341–342
8. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proceedings of the ACM international conference on management of data, Boston. ACM, New York, pp 47–57
9. Hopcroft J, Ullman J (1979) Introduction to automata theory, languages, and computation. Addison-Wesley, Reading
10. Kannan S, Sweedyk Z, Mahaney S (1995) Counting and random generation of strings in regular languages. In: Proceedings of the 6th ACM-SIAM symposium on discrete algorithms, San Francisco. ACM, New York, pp 551–557
11. Rissanen J (1978) Modeling by shortest data description. Automatica 14:465–471
12. Stewart JW (1998) BGP4, inter-domain routing in the Internet. Addison Wesley, Reading

Indexed Two-Dimensional String Matching

Joong Chae Na[1], Paolo Ferragina[2], Raffaele Giancarlo[3], and Kunsoo Park[4]
[1]Department of Computer Science and Engineering, Sejong University, Seoul, Korea
[2]Department of Computer Science, University of Pisa, Pisa, Italy
[3]Department of Mathematics and Applications, University of Palermo, Palermo, Italy
[4]School of Computer Science and Engineering, Seoul National University, Seoul, Korea

Keywords

Index data structures for matrices or images; Indexing for matrices or images; Two-dimensional indexing for pattern matching; Two-dimensional index data structures

Years and Authors of Summarized Original Work

2007; Na, Giancarlo, Park
2011; Kim, Na, Sim, Park

Problem Definition

This entry is concerned with designing and building indexes of a two-dimensional matrix, which is basically the generalization of indexes of a string, the *suffix tree* [12] and the *suffix array* [11], to a two-dimensional matrix. This problem was first introduced by Gonnet [7]. Informally, a two-dimensional analog of the suffix tree is a

tree data structure storing all submatrices of an $n \times m$ matrix, $n \geq m$. The *submatrix tree* [2] is an incarnation of such indexes. Unfortunately, building such indexes requires $\Omega(nm^2)$ time [2]. Therefore, much of the attention paid has been restricted to square matrices and submatrices, the important special case in which much better results are available.

For square matrices, the *Lsuffix tree* and its array form, storing all *square submatrices* of an $n \times n$ matrix, have been proposed [3, 9, 10]. Moreover, the general framework for these index families is also introduced [4, 5]. Motivated by LZ1-type image compression [14], the online case, i.e., the matrix is given one row or column at a time, has been also considered. These data structures can be built in time close to n^2. Building these data structures is a nontrivial extension of the algorithms for the standard suffix tree and suffix array. Generally, a tree data structure and its array form of this type for square matrices are referred to as the *two-dimensional suffix tree* and the *two-dimensional suffix array*, which are the main concerns of this entry.

Notations

Let A be an $n \times n$ matrix with entries defined over a finite alphabet Σ. $A[i \ldots k, j \ldots l]$ denotes the submatrix of A with corners (i, j), (k, j), (i, l), and (k, l). When $i = k$ or $j = l$, one of the repeated indexes is omitted. For $1 \leq i, j \leq n$, the *suffix* $A(i, j)$ of A is the largest square submatrix of A that starts at position (i, j) in A. That is, $A(i, j) = A[i \ldots i + k, j \ldots j + k]$, where $k = n - \max(i, j)$. Let $\$_i$ be a special symbol not in Σ such that $\$_i$ is lexicographically smaller than any other character in Σ. Assume that $\$_i$ is lexicographically smaller than $\$_j$ for $i < j$. For notational convenience, assume that the last entries of the ith row and column are $\$_i$. It makes all suffixes distinct. See Fig. 1a, b for an example.

Let $L\sum = \bigcup_{i=1}^{\infty} \sum^{2i-1}$. The strings of $L\Sigma$ are referred to as *Lcharacters*, and each of them is considered as an atomic item. $L\Sigma$ is called the *alphabet of Lcharacters*. Two Lcharacters are equal if and only if they are equal as strings over Σ. Moreover, given two Lcharacters *La* and *Lb* of equal length, *La* is *lexicographically smaller than or equal to Lb* if and only if the string corresponding to *La* is lexicographically smaller than or equal to that corresponding to *Lb*. A *chunk* is the concatenation of Lcharacters with the following restriction: an Lcharacter in Σ^{2i-1} can precede only one in $\Sigma^{2(i+1)-1}$ and succeed only one in $\Sigma^{2(i-1)-1}$. An *Lstring* is a chunk such that the first Lcharacter is in Σ.

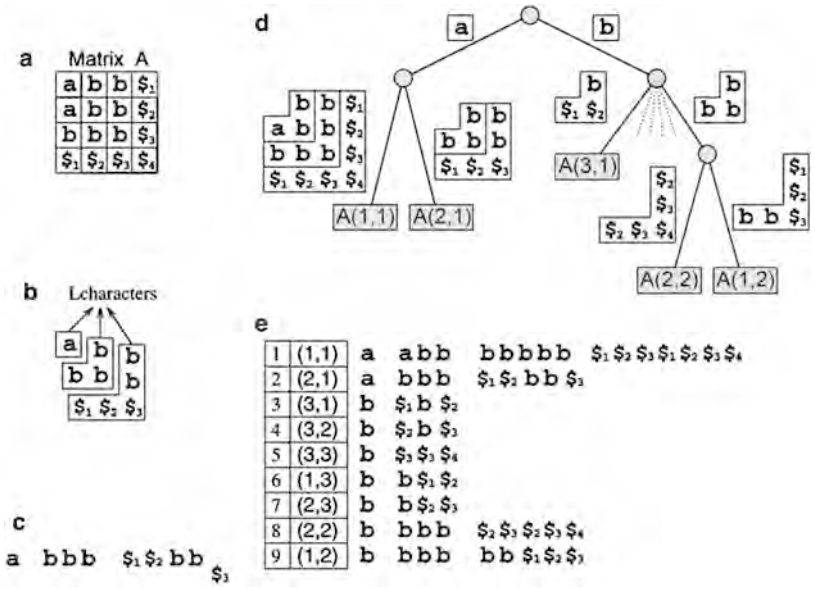
For dealing with matrices as strings, a linear representation of square matrices is needed. Given $A[1 \ldots n, 1 \ldots n]$, divide A into n Lshaped characters. Let $a(i)$ be the concatenation of row $A[i, 1 \ldots i - 1]$ and column $A[1 \ldots i, i]$. Then, $a(i)$ can be regarded as an Lcharacter. The linearized string of matrix A, called the Lstring of matrix A, is the concatenation of Lcharacters $a(1), \ldots, a(n)$. See Fig. 1c for an example. Slightly different linearizations have been used [9, 10, 13], but they are essentially the same in the aspect of two-dimensional functionality.

Two-Dimensional Suffix Trees

The suffix tree of matrix A is a compacted trie over the alphabet $L\Sigma$ that represents Lstrings corresponding to all suffixes of A. Formally, the *two-dimensional suffix tree* of matrix A is a rooted tree that satisfies the following conditions (see Fig. 1d for an example):

1. Each edge is labeled with a chunk.
2. There is no internal node of outdegree one.
3. Chunks assigned to sibling edges start with different Lcharacters, which are of the same length as strings in Σ^*.
4. The concatenation of the chunks labeling the edges on the path from the root to a leaf gives the Lstring of exactly one suffix of A, say $A(i, j)$. It is said that this leaf is associated with $A(i, j)$.
5. There is exactly one leaf associated with each suffix.

Conditions 4 and 5 mean that there is a one-to-one correspondence between the leaves of the tree and the suffixes of A (which are all distinct because $\$_i$ is unique).



Indexed Two-Dimensional String Matching, Fig. 1 (**a**) A matrix A, (**b**) the suffix $A(2,1)$ and Lcharacters composing $A(2,1)$, (**c**) the Lstring of $A(2,1)$, (**d**) the suffix tree of A, and (**e**) the suffix array of A (omitting the suffixes started with $\$_i$)

Problem 1 (Construction of 2D suffix tree)

INPUT: An $n \times n$ matrix A.

OUTPUT: A two-dimensional suffix tree storing all square submatrices of A.

Online Suffix Trees

Assume that A is read *online* in row major order (column major order can be considered similarly). Let $A_t = A[1 \ldots t, 1 \ldots n]$ and $row_t = A[t, 1 \ldots n]$. At time $t-1$, nothing but A_{t-1} is known about A. At time t, row_t is read and so A_t is known. After time t, the online suffix tree of A is storing all suffixes of A_t . Note that Condition 4 may not be satisfied during the online construction of the suffix tree. A leaf may be associated with more than one suffix, because the suffixes of A_t are not all distinct.

Problem 2 (Online construction of 2D suffix tree)

INPUT: A sequence of rows of $n \times n$ matrix A, $row_1, row_2, \ldots, row_n$.

OUTPUT: A two-dimensional suffix tree storing all square submatrices of A_t after reading row_t .

Two-Dimensional Suffix Arrays

The *two-dimensional suffix array* of matrix A is basically a sorted list of all Lstrings corresponding to suffixes of A. Formally, the kth element of the array has the start position (i, j) if and only if the Lstring of $A(i, j)$ is the kth smallest one among the Lstrings of all suffixes of A. See Fig. 1e for an example. The two-dimensional suffix array is also coupled with additional information tables, called *Llcp* and *Rlcp*, to enhance its performance like the standard suffix array. The two-dimensional suffix array can be constructed from the two-dimensional suffix tree in linear time.

Problem 3 (Construction of 2D suffix array)

INPUT: An $n \times n$ matrix A.

OUTPUT: The two-dimensional suffix array storing all square submatrices of A.

Submatrix Trees

The *submatrix tree* is a tree data structure storing all submatrices. This entry just gives a result on submatrix trees. See [2] for details.

Problem 4 (Construction of a submatrix tree)

INPUT: An $n \times m$ matrix B, $n \geq m$.
OUTPUT: The submatrix tree and its array form storing all submatrices of B.

Key Results

Theorem 1 (Kim et al. 2011 [10], Cole and Hariharan 2003 [1]) *Given an $n \times n$ matrix A over an integer alphabet, one can construct the two-dimensional suffix tree in $O(n^2)$ time.*

Kim and Park's result is a deterministic algorithm, while Cole and Hariharan's result is a randomized one. For an arbitrary alphabet, one needs first to sort it and then to apply the theorem above.

Theorem 2 (Na et al. 2007 [13]) *Given an $n \times n$ matrix A, one can construct online the two-dimensional suffix tree of A in $O(n^2 \log n)$ time.*

Theorem 3 (Kim et al. 2003 [9]) *Given an $n \times n$ matrix A, one can construct the two-dimensional suffix array of A in $O(n^2 \log n)$ time without constructing the two-dimensional suffix tree.*

Theorem 4 (Giancarlo 1993 [2]) *Given an $n \times m$ matrix B, one can construct the submatrix tree of B in $O(nm^2 log(nm))$ time.*

Applications

Two-dimensional indexes can be used for many pattern-matching problems of two-dimensional applications such as low-level image processing, image compression, visual data bases, and so on [3, 6]. Given an $n \times n$ text matrix and an $m \times m$ pattern matrix over an alphabet Σ, the *two-dimensional pattern retrieval problem*, which is a basic pattern-matching problem, is to find all occurrences of the pattern in the text. The two-dimensional suffix tree and array of the text can be queried in $O(m^2 \log |\Sigma| + occ)$ time and $O(m^2 + \log n + occ)$ time, respectively, where occ is the number of occurrences of the pattern in the text. This problem can be easily extended to a set of texts. These queries have the same procedure and performance as those of indexes for strings. Online construction of the two-dimensional suffix tree can be applied to LZ-1-type image compression [6].

Open Problems

The main open problems on two-dimensional indexes are to construct indexes in optimal time. The linear-time construction algorithm for two-dimensional suffix trees is already known [10]. The online construction algorithm due to [13] is optimal for unbounded alphabets, but not for integer or constant alphabets. Another open problem is to construct two-dimensional suffix arrays directly in linear time.

Experimental Results

An experiment that compares construction algorithms of two-dimensional suffix trees and suffix arrays was presented in [8]. Giancarlo's algorithm [2] and Kim et al.'s algorithm [8] were implemented for two-dimensional suffix trees and suffix arrays, respectively. Random matrices of sizes $200 \times 200 \sim 800 \times 800$ and alphabets of sizes 2, 4, 16 were used for input data. According to experimental results, the construction of two-dimensional suffix arrays is ten times faster and five times more space efficient than that of two-dimensional suffix trees.

Cross-References

- Multidimensional String Matching
- Suffix Array Construction
- Suffix Tree Construction

Recommended Reading

1. Cole R, Hariharan R (2003) Faster suffix tree construction with missing suffix links. SIAM J Comput 33:26–42

2. Giancarlo R (1993) An index data structure for matrices, with applications to fast two-dimensional pattern matching. In: Proceedings of workshop on algorithm and data structures, Montréal. Springer Lecture notes in computer science, vol 709, pp 337–348
3. Giancarlo R (1995) A generalization of the suffix tree to square matrices, with application. SIAM J Comput 24:520–562
4. Giancarlo R, Grossi R (1996) On the construction of classes of suffix trees for square matrices: algorithms and applications. Inf Comput 130:151–182
5. Giancarlo R, Grossi R (1997) Suffix tree data structures for matrices. In: Apostolico A, Galil, Z (eds) Pattern matching algorithms, ch. 11. Oxford University Press, Oxford, pp 293–340
6. Giancarlo R, Guaiana D (1999) On-line construction of two-dimensional suffix trees. J Complex 15:72–127
7. Gonnet GH (1988) Efficient searching of text and pictures. Technical report OED-88-02, University of Waterloo
8. Kim DK, Kim YA, Park K (1998) Constructing suffix arrays for multi-dimensional matrices. In: Proceedings of the 9th symposium on combinatorial pattern matching, Piscataway, pp 249–260
9. Kim DK, Kim YA, Park K (2003) Generalizations of suffix arrays to multi-dimensional matrices. Theor Comput Sci 302:401–416
10. Kim DK, Na JC, Sim JS, Park K (2011) Linear-time construction of two-dimensional suffix trees. Algorithmica 59:269–297
11. Manber U, Myers G (1993) Suffix arrays: a new method for on-line string searches. SIAM J Comput 22:935–948
12. McCreight EM (1976) A space-economical suffix tree construction algorithms. J ACM 23:262–272
13. Na JC, Giancarlo R, Park K (2007) On-line construction of two-dimensional suffix trees in $O(n^2 \log n)$ time. Algorithmica 48:173–186
14. Storer JA (1996) Lossless image compression using generalized LZ1-type methods. In: Proceedings of data compression conference, Snowbird, pp 290–299

Inductive Inference

Sandra Zilles
Department of Computer Science, University of Regina, Regina, SK, Canada

Keywords

Induction; Learning from examples; Recursion theory

Years and Authors of Summarized Original Work

1983; Case, Smith

Problem Definition

The theory of inductive inference is concerned with the capabilities and limitations of machine learning. Here the learning machine, the concepts to be learned, as well as the hypothesis space are modeled in recursion theoretic terms, based on the framework of identification in the limit [1, 9, 15].

Formally, considering recursive functions (mapping natural numbers to natural numbers) as target concepts, a learner (inductive inference machine) is supposed to process, step by step, gradually growing initial segments of the graph of a target function. In each step, the learner outputs a program in some fixed programming system, where successful learning means that the sequence of programs returned in this process eventually stabilizes on some program actually computing the target function.

Case and Smith [3, 4] proposed several variants of this model in order to study the influence that certain constraints or relaxations may have on the capabilities of learners. Their models restrict (i) the number of mind changes (i.e., changes of output programs) a learner is allowed to make during the learning process and (ii) the number of errors the program eventually hypothesized may have when compared to the target function.

One major result of studying the corresponding effects is a hierarchy of inference types culminating in a model general enough to allow for the identification of the whole class of recursive functions by a single inductive inference machine.

Notation

The target concepts for learning in the model discussed below are recursive functions [14] mapping natural numbers to natural numbers. Such functions, as well as partial recursive functions in general, are considered as computable

in an arbitrary, but fixed Gödel numbering $\varphi = (\varphi_i)_{i\in\mathbb{N}}$. Here $\mathbb{N} = \{0, 1, 2, \ldots\}$ denotes the set of all natural numbers. $\varphi = (\varphi_i)_{i\in\mathbb{N}}$ is interpreted as a programming system, where the number $i \in \mathbb{N}$ is called a program for the partial recursive function φ_i.

Suppose f and g are partial recursive functions and $n \in \mathbb{N}$. Below $f =^n g$ is written if the set $\{x \in \mathbb{N} | f(x) \neq g(x)\}$ is of cardinality at most n. If the set $\{x \in \mathbb{N} | f(x) \neq g(x)\}$ is finite, this is denoted by $f =^* g$. One considers $*$ as a special symbol for which the $<$-relation is extended by $n < *$ for all $n \in \mathbb{N}$. For any recursive f and any $z \in \mathbb{N}$, let $f[z]$ denote $(z, (f(0), \ldots, f(z)))$ for short.

For further basic recursion theoretic notions, the reader is referred to [14].

Learning Models

Case and Smith [4] build their theory upon the fundamental model of identification in the limit [1, 9]. There a learner can be understood as an algorithmic device, called an inductive inference machine, which, given any "graph segment" $f[z]$ as its input, returns a program $i \in \mathbb{N}$. Such a learner M identifies a recursive function f in the limit, if there is some $j \in \mathbb{N}$ such that

$$\varphi_j = f \text{ and } M(f[z]) = j \text{ for all but finitely many } z \in \mathbb{N}.$$

A class of recursive functions is learnable in the limit, if there is an inductive inference machine identifying each function in the class in the limit. Identification in the limit is called EX-identification, since a program for f is termed an *ex*planation for f.

For instance, the class of all primitive recursive functions is EX-identifiable, whereas the class of all recursive functions is not [9].

The central question discussed by Case and Smith [4] is how the limitations of EX-learners are affected by posing certain requirements on the success criterion, concerning:

- Convergence criteria:
  - e.g., when restricting the number of permitted mind changes
  - e.g., when relaxing the constraints on syntactical convergence of the sequence of programs returned in the learning process
- Accuracy:
  - e.g., when relaxing the number of permitted anomalies in the programs returned eventually

Problem 1 In which way do modifications of EX-identification in terms of accuracy and convergence criteria affect the capabilities of the corresponding learners?

Problem 2 In particular, if inaccuracies are permitted, can EX-learners always refute inaccurate hypotheses?

Problem 3 How much relaxation of the model of EX-identification is needed to achieve learnability of the full class of recursive functions?

Key Results

Accuracy and Convergence Constraints

In order to systematically address these problems, Case and Smith [4] defined inference types reflecting restrictions and relaxations of EX-identification as follows.

Definition 1 Suppose S is a class of recursive functions and $m, n \in \mathbb{N} \cup \{*\}$. S is $\mathrm{EX}_n{}^m$-identifiable if there is an inductive inference machine M, such that for any function $f \in S$, there is some $j \in \mathbb{N}$ satisfying:

- $M(f[z]) = j$ for all but finitely many $z \in \mathbb{N}$.
- $j = mf$.
- The cardinality of the set $\{z \in \mathbb{N} | M(f[z]) \neq M(f[z+1])\}$ is at most n.

For intuition one may view n as an upper bound on the allowed number of "mind changes" and m as an upper bound on the allowed number of "anomalies."

$\mathrm{EX}_n{}^m$ denotes the set of all classes of recursive functions which are $\mathrm{EX}_n{}^m$-identifiable.

Definition 2 Suppose S is a class of recursive functions and $m \in \mathbb{N} \cup \{*\}$. S is BC^m-identifiable if there is an inductive inference machine M, which, for any function $f \in S$, satisfies:

- $\varphi_{M(f[z])} =^m f$ for all but finitely many $z \in \mathbb{N}$.

BC^m denotes the set of all classes of recursive functions which are BC^m-identifiable. BC is short for behaviorally correct; the difference to EX-learning is that convergence of the sequence of programs returned by the learner is defined only in terms of semantics, no longer in terms of syntax.

The Impact of Accuracy and Convergence Constraints

In general, each permission of mind changes or anomalies increases the capabilities of learners; however, mind changes cannot be traded in for anomalies or vice versa.

Theorem 1 *Let* $a, b, c, d \in \mathbb{N} \cup \{*\}$. *Then* $\mathrm{EX}_b{}^a \subseteq \mathrm{EX}_d{}^c$ *if and only if* $a \leq c$ *and* $b \leq d$.

Corollary 1 *For any* $m, n \in \mathbb{N}$, *the following inclusions hold.*

1. $\mathrm{EX}_n^m \subset \mathrm{EX}_n^{m+1} \subset \mathrm{EX}_n^*$.
2. $\mathrm{EX}_n^m \subset \mathrm{EX}_{n+1}^m \subset \mathrm{EX}_*^m$.

Theorem 2 *Let* $n \in \mathbb{N}$. *Then* $\mathrm{EX}_*^* \subset \mathrm{BC}^n \subset \mathrm{BC}^{n+1} \subset \mathrm{BC}^*$.

These results provide a solution to Problem 1.

Refutability

In particular, refutability demands (in the sense that every incorrect hypothesis should be refutable; see [13]) are not applicable in the theory of inductive inference; see Problem 2.

Formally, Case and Smith [4] consider refutability as a property guaranteed by Popperian machines, the latter being defined as follows:

Definition 3 Suppose M is an inductive inference machine M. M is Popperian if, on any input, M returns a program of a recursive function.

Results thereon include the following:

Theorem 3 *There is an EX-identifiable class S of recursive functions for which there is no Popperian inductive inference machine witnessing its EX-identifiability.*

Corollary 2 *There is an* EX^1*-identifiable class S of recursive functions for which there is no Popperian inductive inference machine witnessing its* EX^1*-identifiability.*

Additionally, in EX^1-identification, Popper's refutability principle cannot be applied even if it concerns only those hypotheses returned in the limit.

Learning All Recursive Functions

Since the results above yield a hierarchy of inference types with strictly growing collections of learnable classes, there is also an implicit answer to Problem 3: the class of recursive functions is neither in $\mathrm{EX}_n{}^m$ for any $m, n \in \mathbb{N} \cup \{*\}$ nor in BC^m for any $m \in \mathbb{N}$. In contrast to that, Case and Smith [4] prove:

Theorem 4 *The class of all recursive functions is in* BC^*.

Applications

The work of Case and Smith [4] has been of high impact in learning theory.

A consequence of the discussion of anomalies is that refutability principles in general do not hold for identification in the limit. This result has given rise to later studies on methods and techniques inductive inference machines might apply in order to discover their errors [7] and thus to further insights into the nature of inductive inference.

Concerning the study of mind change hierarchies, among others, their lifting to transfinite ordinal numbers [8] is a notable extension.

Moreover, the theory of learning as proposed by Case and Smith [4] has been applied for the development of the theory of identifying recursive [11] or recursively enumerable [10] languages.

Open Problems

Among the currently open problems in inductive inference, one key challenge is to find a reasonable notion of the complexity of learning problems (i.e., of classes of recursive functions) involving the run-time complexity of learners as well as the number of mind changes required to learn the functions in a class. In particular, special natural classes of functions should be analyzed in terms of such a complexity notion.

Though of course the hierarchies $\mathrm{EX}_0{}^m \subset \mathrm{EX}_1{}^m \subset \mathrm{EX}_2{}^m \subset \ldots$ for any $m \in \mathbb{N}$ reflect some increase of complexity in that sense, a corresponding complexity notion would not address the aspect of run-time complexity of learners. Different complexity notions have been introduced, such as the so-called intrinsic complexity [2, 6] (neglecting run-time complexity) and the "measure under the curve" [5] (respecting the number of examples required, but neglecting the number of mind changes). In particular, for learning deterministic finite automata, different notions of run-time complexity have been discussed [12].

However, the definition of a more capacious complexity notion remains an open issue.

Cross-References

▸ PAC Learning

Recommended Reading

1. Blum L, Blum M (1975) Toward a mathematical theory of inductive inference. Inf Control 28(2):125–155
2. Case J, Kötzing T (2011) Measuring learning complexity with criteria epitomizers. In: Proceedings of the 28th international symposium on theoretical aspects of computer science, Dortmund. Leibniz international proceedings in informatics, pp 320–331
3. Case J, Smith CH (1978) Anomaly hierarchies of mechanized inductive inference. In: Proceedings of the 10th symposium on the theory of computing, San Diego. ACM, New York, pp 314–319
4. Case J, Smith CH (1983) Comparison of identification criteria for machine inductive inference. Theor Comput Sci 25(2):193–220
5. Daley RP, Smith CH (1986) On the complexity of inductive inference. Inf Control 69(1–3):12–40
6. Freivalds R, Kinber E, Smith CH (1995) On the intrinsic complexity of learning. Inf Comput 118(2):208–226
7. Freivalds R, Kinber E, Wiehagen R (1995) How inductive inference strategies discover their errors. Inf Comput 123(1):64–71
8. Freivalds R, Smith CH (1993) On the role of procrastination in machine learning. Inf Comput 107(2):237–271
9. Gold EM (1967) Language identification in the limit. Inf Control 10(5):447–474
10. Kinber EB, Stephan F (1995) Language learning from texts: mindchanges, limited memory, and monotonicity. Inf Comput 123(2):224–241
11. Lange S, Grieser G, Zeugmann T (2005) Inductive inference of approximations for recursive concepts. Theor Comput Sci 348(1):15–40
12. Pitt L (1989) Inductive inference, DFAs, and computational complexity. In: Analogical and inductive inference, 2nd international workshop, Reinhardsbrunn Castle, GDR. Lecture notes in computer science, vol 397. Springer, Berlin, pp 18–44
13. Popper K (1959) The logic of scientific discovery. Harper & Row, New York
14. Rogers H (1967) Theory of recursive functions and effective computability. McGraw-Hill, New York
15. Zeugmann T, Zilles S (2008) Learning recursive functions: a survey. Theor Comput Sci 397:4–56

Influence and Profit

Yuqing Zhu[2,3] and Weili Wu[1,2,3]
[1]College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi Province, China
[2]Department of Computer Science, California State University, Los Angeles, CA, USA
[3]Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Keywords

Data mining; Influence; Profit; Social networks; Viral marketing

Years and Authors of Summarized Original Work

2003; Kempe, Kleinberg, Tardos
2013; Zhu, Lu, Bi, Wu, Jiang, Li

Problem Definition

A social network is a graph of relationships and interactions within a set of individuals. Information can spread within a social network by "word-of-mouth" effects. In other words, information diffuses from individuals to individuals in a social network through the connections between them, and if some information is spread by some initial individuals, many individuals may believe in it due to information diffusion. A social network is denoted as G=(V, E, w), where V is a set of vertexes with size n, $E \subseteq V \times V$ is a set of edges with size m, and $w : E \to [0, 1]$ is the set of all $w(u, v)$ which is the weight of edge (u, v).

Independent Cascade (IC) and Linear Threshold (LT) Models

The IC and LT models [1] are two basic models of influence diffusion in social networks, and there are two vertex stages: *inactive* and *active*. The influence always starts from a set of a set S consists of seeds (initially active nodes). The time is divided into discrete steps $0, 1, 2, \ldots$ Denote S_i the set of active vertexes at step i ($S_0 = S$ and $S_{-1} = \emptyset$). In the IC model, influence propagates as follows: S_i is the union set of S_{i-1} and other vertexes activated by vertexes in $S_{i-1} \setminus S_{i-2}$ in step i. Each node u has only one chance to activate each of its neighbors v with probability $w(u, v)$ when u first becomes active. In the LT model, influence propagates as follows: at the beginning each vertex v picks a threshold θ_v uniformly at random from $[0, 1]$ which is the threshold of this vertex becoming active. In each step i, $S_i = S_{i-1} \cup \{v | \sum_{u \in S_{i-1}} w(u, v) \geq \theta_v\}$. Both IC and LT models stop at the step $t+1$ when the process reaches its maximum influence, i.e., $S_{t+1} = S_t$.

Problem 1: Influence Maximization Problem (InfMax) [1]

INPUT: *A social network $G = (V, E, w)$ and k, the number of seeds.*

OUTPUT: *The set S containing k seeds that maximizes the influence $\mathcal{I}(S)$.*

Price-Related Propagation (PR) Frame

Adding monetary factor into the propagation process of the IC and LT models makes this *price related (PR) propagation* frame. In the PR frame, only the individual who adopts a product propagates this product's influence, and the adoption depends on the relationship between the price offered and the individual's valuation about this product. In detail, every vertex u has three stages: *neutral*, *influenced*, and *active*. Vertex u being neutral means it has no idea or positive attitude about this product. When u becomes influenced, u holds a positive attitude to the product but u hasn't adopted this product yet. Only if u further turns into active stage, u adopts the product and propagates the influence by telling its network neighbors. The PR frame separates, holding a positive attitude and propagating influence, in which the two are the same in traditional IC and LT models. This separation comes from the fact that individuals in social networks are independent human beings who not only are influenced by the people around but also have their own judgements. If someone receives some information, surely he or she should first evaluate the information before spreading it.

The PR frame assumes that each individual u has a *valuation* for the product, which is the highest price this individual thinks the product is worth. The rule of judging whether an influenced individual turns into active is the following: only if u is *influenced* and its valuation is higher than the offered price, u will turn *active*, adopt this product, and propagate the influence. The PR frame is an extension to the IC and LT models; it contains the PR-I model based on the IC and the PR-L model based on the LT. The rules of an individual turning from *neutral* to *influenced* in PR-I and PR-L model are the same as the rules of an individual turning from *inactive* to *active* in IC and LT model, respectively. However in the PR frame the *influenced* individuals do not propagate influence, but only the *active* ones do, and an *influence* individual turns to *active* if and only if the offered price is lower than its valuation.

Pricing Strategies in the PR Frame

Since price is vitally significant in the PR frame, we design two strategies to determine the prices offered to the individuals. The first one is *binary pricing* (BYC), in which all chosen seeds are given free samples and all other individuals are charged the same price, and the second one is *panoramic pricing* (PAP), in which prices for individuals including seeds are unconstrained different values that can be any value if needed.

In the PR frame, choosing node u as a seed merely means turning u to *influenced*. However, in BYC, any seed u must further become *active* for each seed is offered a free sample, i.e., the offered price is 0 and no greater than the valuation. In PAP, on the other hand, a seed u may not be *active*.

Price plays a vital role on the influence and profit in the PR frame. High prices may bring high profit but it hinders the influence propagation, and to enlarge the influence, some sacrifice on profit is inevitable. Base on this observation, a parameter $\lambda \in [0, 1)$ is adopted to denote the decision maker's preference toward influence and profit, and the objective is the weighted sum of influence and profit, which we call balanced influence and profit (BIP).

Problem 2: Balanced Influence and Profit Maximization Problem (BIPMax) [2]

INPUT: *A social network $G = (V, E, w)$, the distribution of customer evaluation, and λ the decision maker's preference.*

OUTPUT: *The seed set S and the price $\boldsymbol{p}$ for all individuals that maximize the objective function $\mathcal{B}(S, \boldsymbol{p}) = \lambda \cdot \mathcal{I}(S, \boldsymbol{p}) + (1-\lambda) \cdot \mathcal{R}(S, \boldsymbol{p})$ where $\mathcal{I}(S, \boldsymbol{p})$ is the influence and $\mathcal{R}(S, \boldsymbol{p})$ is the profit.*

Key Results

Result 1: *InfMax under the IC and LT models is both NP-hard.* [1]

Result 2: *BIPMax under the PR-I and PR-L models is both NP-hard.* [2]

The above two results show the difficulties of solving InfMax and BIPMax, respectively. It can be seen that both of them are "hard" to solve. However, approximation algorithms may exist, and the following two properties are used to design and analyze algorithms. Suppose f is a set function on subsets of V.

Submodularity and Monotony

1. **Submodular function.** f is called *submodular* if for every $X \subseteq Y \subseteq V$ and $z \in V \setminus Y$, $f(X \cup \{z\}) - f(X) \geq f(Y \cup \{z\}) - f(Y)$.
2. **Monotone function.** f is called *monotone* if $f(X \cup \{z\}) \geq f(X)$ for any set $X \subset V$ and element $z \in V$.

Result 3: *Influence $\mathcal{I}(S)$ under both IC and LT models is submodular and monotone w.r.t. S.* [1]

Result 4: *BIP $\mathcal{B}(S, \boldsymbol{p})$ under both PR-I and PR-L models is submodular w.r.t. S, if the prices $\boldsymbol{p}$ are fixed and $p_i \geq c$, where p_i is the ith element of $\boldsymbol{p}$ and c is the manufacturing cost of the product.* [2]

Remark 1 $\mathcal{B}(S, \boldsymbol{p})$ under both PR-I and PR-L models is non-monotone w.r.t. S. [2]

Algorithm for InfMax

Nemhauser et al. in [3] showed that greedy hill-climbing algorithm has the approximation ratio with $1 - 1/e$ of maximizing a submodular and monotone set function f. The greedy algorithm of maximizing influence is presented in Algorithm 1: each time the vertex that brings the highest marginal influence will be picked as a new seed, until the desired number of seeds are picked. Hence, according to Result 3, Algorithm 1 has a constant performance ratio $1 - 1/e$ solving InfMax.

Note that computing actual influence as well as marginal influence is #P-hard [1]. To estimate the influence in a reasonable time, Monte Carlo simulation is usually adopted, generating a number of samples and calculating the average value of all samples.

Algorithm 1 Greedy algorithm

$S \leftarrow \emptyset$;
while $|S| < k$ **do**
  $u = \arg\max_{u \in V \setminus S} \{\mathcal{I}(S \cup \{u\}) - \mathcal{I}(S)\}$;
  $S \leftarrow S \cup u$;
end while
output S;

Algorithms for BIPMax

For a non-monotone submodular set function f, Feige et al. [4] devised a deterministic local-search $\frac{1}{3}$-approximation and a randomized $\frac{2}{5}$-approximation algorithm if f is nonnegative; therefore, if the prices $\boldsymbol{p}$ are preknown and fixed, the techniques in [4] may be ideal approximation algorithms. However, usually prices $\boldsymbol{p}$ are to be determined to achieve the maximum BIP, and general algorithms need careful consideration.

To give a better pricing method for BIPMax, both the manufacturing cost and local influence should be considered. The manufacturing cost is denoted by c. Individual v_i's evaluation is a random variable X_i whose cumulative distribution function (CDF) is denoted by F_i. (If v_i is *influenced* and being offered price q, then the probability that v_i turns *active* is $\text{Prob}(x_i \geq q) = 1 - F_i(q)$.) For v_i itself, if it is chosen as the only seed and offered price p, the expected profit solely from v_i is $(1-F_i(p))(p-c)$, what is more, the expected influence to other nodes solely from v_i is $(1-F_i(p))\cdot\mathcal{I}(v_i, \boldsymbol{p})$. However, $\mathcal{I}(v_i, \boldsymbol{p})$ depends on other nodes' prices; to ease the computation, the following simple one-hop estimation is adopted: $\sum_{\forall u|\text{ outneighbor of } v_i} w(v_i, u)/d^{\text{out}}(v_i)$, where $d^{\text{out}}(v_i)$ is the outdegree of v_i. Then the *optimal price* p_i' for v_i is calculated as follows:

$$\tilde{\mathcal{B}}_i(p) = \lambda(1 - F_i(p)) \sum \frac{w(v_i,u)}{d^{\text{out}}(v_i)} + (1-\lambda)(1-F_i(p))(p-c), \quad (1)$$

$$p_i' = \arg\min_{p \in [0,1]} \tilde{\mathcal{B}}_i(p). \quad (2)$$

Equation (1) considers both the manufacturing cost and the network structure; however, the price calculated by (2) is still myopic.

Determine the Seeds and Prices Under BYC

In BYC the prices can only be 0 or the full price; for a company this strategy takes the least implementation expense. *ABYC* is the algorithm for BYC. ABYC contains two stages: first offering every individual a same full price and second determining the seeds whom free samples are given to.

Equation (2) is not used in the first stage since the obtained p_i' may vary from v_i. Instead we calculate the *universal optimal price*: $p_U' = \arg\min_{p \in [0,1]} \sum_i^n \tilde{\mathcal{B}}_i(p)$.

Greedy is used in the second stage of determining seeds: every round for each non-seed vertex u we compute the marginal BIP of picking u as a seed, and choose the vertex that provides the highest marginal gain. When no marginal BIP gain can be bought by any vertex, ABYC stops.

Suppose the price vector $\boldsymbol{p} = (p_1, \ldots p_n)$; denote $(\boldsymbol{p}_{-i}, q)$ the vector obtained by altering p_i, the ith element of $\boldsymbol{p}$ to q, i.e., $(\boldsymbol{p}_{-i}, q) = (p_1, \ldots, p_{i-1}, q, p_{i+1}, \ldots, p_n)$.

Algorithm 2 ABYC: the algorithm for BYC

$S \leftarrow \emptyset$, $\boldsymbol{p} \leftarrow \boldsymbol{0}$;
for $\forall v_i \in V$ **do**
  $p_i \leftarrow p_U'$;
end for
while true **do**
  $u \leftarrow \arg\max_{v_i \in V \setminus S} \{\mathcal{B}(S \cup \{v_i\}, (\boldsymbol{p}_{-i}, 0)) - \mathcal{B}(S, \boldsymbol{p})\}$;
  if $\mathcal{B}(S \cup \{u\}, (\boldsymbol{p}_{-i}, 0)) - \mathcal{B}(S, \boldsymbol{p}) > 0$ **then**
    $S \leftarrow S \cup \{u\}$; $\boldsymbol{p} \leftarrow (\boldsymbol{p}_{-i}, 0)$;
  else *break*;
  end if
end while
output $(S, \boldsymbol{p})$;

Determine the Seeds and Prices Under PAP

BYC is easy to implement, however it is too simple and constrained, PAP is much freer where prices are assigned with no constraint. APAP is the algorithm for PAP, and like ABYC it also contains two stages. In the first stage, to obtain p_i' for every v_i (2) is adopted. In the second stage, the vertex with the maximum marginal BIP gain

is picked step by step until no positive gain is available.

The computation of the marginal BIP gain under PAP when adding v_i into the seed set S is much more complex comparing to BYC, since when choosing v_i as a new seed a new price may also be offered to it. Suppose the new price for v_i is q, then the marginal BIP gain of adding v_i is: $\mathcal{B}(S \cup \{v_i\}, (\boldsymbol{p}_{-i}, q)) - \mathcal{B}(S, \boldsymbol{p})$. Since $\mathcal{B}(S, \boldsymbol{p})$ is a constant w.r.t. q, $\mathcal{B}(S \cup \{v_i\}, (\boldsymbol{p}_{-i}, q))$ should be maximized. When offering price q to v_i, only two outcomes exist in the sample space, outcome ω_1 where v_i accepts the price and turns *active*, outcome ω_0 where v_i rejects the price, stays *influenced* and never spreads the influence. If ω_1 happens, the influence gain collected from v_i is 1 and the profit gain collected from v_i is $q - c$, suppose the influence from other nodes is I_1 and the profit from other nodes is R_1, then the BIP gain is $g_i(q) = \lambda(I_1 + 1) + (1-\lambda)(q - c + R_1)$, which is a linear function w.r.t. q. Else if ω_0 happens, the influence gain collected from v_i is 1 and the profit gain collected from v_i is 0, suppose the influence from other nodes is I_0 and the profit from other nodes is R_0, then the BIP gain is $h_i = \lambda(I_0 + 1) + (1 - \lambda)R_0$, a constant independent of q. $\text{Prob}(\omega_1) = 1 - F_i(q)$ and $\text{Prob}(\omega_0) = F_i(q)$. Hence the expected BIP is:

$$\delta_i(q) = g_i(q) \cdot (1 - F_i(q)) + h_i \cdot F_i(q) \quad (3)$$

Algorithm 3 APAP: the algorithm for PAP

$S \leftarrow \emptyset$, $\boldsymbol{p} \leftarrow \boldsymbol{0}$;
for $\forall v_i \in V$ **do**
  $p_i \leftarrow p_i' = \arg\min_{p \in [0,1]} \tilde{\mathcal{B}}_i(p)$;
end for
while true **do**
  for $\forall v_i \in V \setminus S$ **do**
    $p_i^* \leftarrow \arg\max_{q \in [0,1]} \delta_i(q)$;
  end for
  $u \leftarrow \arg\max_{v_i \in V \setminus S} \{\mathcal{B}(S \cup \{v_i\}, (\boldsymbol{p}_{-i}, p_i^*)) - \mathcal{B}(S, \boldsymbol{p})\}$;
  if $\mathcal{B}(S \cup \{u\}, (\boldsymbol{p}_{-i}, p_i^*)) - \mathcal{B}(S, \boldsymbol{p}) > 0$ **then**
    $S \leftarrow S \cup \{u\}$; $\boldsymbol{p} \leftarrow (\boldsymbol{p}_{-i}, p_i^*)$;
  else *break*;
  end if
end while
output $(S, \boldsymbol{p})$;

To calculate I_1 and R_1, set v_i turns active with probability 1 and run Monte Carlo simulations, and to calculate I_0 and R_0, set v_i turns active with probability 0 and run Monte Carlo simulations. After obtaining I_1, R_1, I_0, and R_0, p_i^* should be computed. If $\delta_i(q)$ is a closed form, then p_i^* is easy to calculate. However $\delta_i(q)$ may not be a close form. For example, if the valuation follows normal distribution, then F_i contains an integral term and $\delta_i(q)$ is not a closed form. In this case, *golden section search* [5] which works fast on finding the extremum of a strictly unimodal function can be used. This technique successively narrows the range inside which the extremum exists to find it. Even if $\delta_i(q)$ is not always unimodal, it is unimodel in subintervals of $[0, 1]$. To reduce error, divide the interval $[0, 1]$ into several small intervals with the same size and pick each small interval's midpoint as a sample q_t. The search starts with the interval that contains the sample $q_0 = \arg\max_t \delta_i(q_t)$ and stops when the interval that contains p^* is narrower than a predefined threshold.

Cross-References

- Greedy Approximation Algorithms
- Influence and Profit

Recommended Reading

1. Kempe D, Kleinberg J, Tardos É (2003) Maximizing the spread of influence through a social network. In: ACM KDD '03, Washington, DC, pp 137–146
2. Zhu Y, Lu Z, Bi Y, Wu W, Jiang Y, Li D (2013) Influene and profit: two sizes of the coin. In: IEEE ICDM '13, Dallas, 1301–1306
3. Nemhauser GL, Wolsey LA, Fisher ML (1978) An analysis of approximations for maximizing submodular set functions-i. Math Program 14(1):265–294
4. Feige U, Mirrokni VS, Vondrák J (2007) Maximizing non-monotone submodular functions. In: IEEE symposium on foundations of computer science (FOCS'07), Providence, pp 461–471

5. Kiefer L (1953) Sequential minimax search for a maximum. In: Proceedings of the American mathematical society, pp 502–206
6. Domingos P, Richardson M (2001) Mining the network value of customers. In: ACM KDD '01, San Francisco, pp 57–66
7. Richardson M, Domingos P (2002) Mining knowledge-sharing sites for viral marketing. In: ACM KDD '02, Edmonton, pp 61–70
8. Kleinberg R, Leighton T (2003) The value of knowing a demand curve: bounds on regret for online posted-price auctions. In: IEEE FOCS '03, Cambridge, pp 594–628
9. Hartline J, Mirrokni V, Sundararajan M (2008) Optimal marketing strategies over social networks. In: ACM WWW '08, Beijing, pp 189–198
10. Arthur D, Motwani R, Sharma A, Xu Y (2009) Pricing strategies for viral marketing on social networks. In: CoRR. abs/0902.3485
11. Lu W, Lakshmanan LVS (2012) Profit maximization over social networks. In: IEEE ICDM '12, Brussels

Influence Maximization

Zaixin Lu[1] and Weili Wu[2,3,4]
[1]Department of Mathematics and Computer Science, Marywood University, Scranton, PA, USA
[2]College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi Province, China
[3]Department of Computer Science, California State University, Los Angeles, CA, USA
[4]Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Keywords

Approximation algorithm; Influence maximization; NP-hard; Social network

Years and Authors of Summarized Original Work

2011; Lu, Zhang, Wu, Fu, Du
2012; Lu, Zhang, Wu, Kim, Fu

Problem Definition

One of the fundamental problems in social network is influence maximization. Informally, if we can convince a small number of individuals in a social network to adopt a new product or innovation, and the target is to trigger a maximum further adoptions, then which set of individuals should we convince? Consider a social network as a graph $G(V, E)$ consisting of individuals (node set V) and relationships (edge set E); essentially influence maximization comes down to the problem of finding important nodes or structures in graphs.

Influence Diffusion

In order to address the influence maximization problem, first it is needed to understand the influence diffusion process in social networks. In other words, how does the influence propagate over time through a social network? Assume time is partitioned into discrete time slots, and then influence diffusion can be modeled as the process by which activations occur from neighbor to neighbor. In each time slot, all previously activated nodes remain active and others either remain inactive or be activated by their neighbors according to the activation constraints. The whole process runs in a finite number of time slots and stops at a time slot when no more activation occurs. Let S denote the set of initially activated nodes; we denote by $f(S)$ eventually the number of activations, and the target is to maximize $f(S)$ with a limited budget.

Problem (Influence Maximization)

INPUT: A graph $G(V, E)$ where V is the set of individuals and E is the set of relationships, an activation model f, and a limited budget number K.

OUTPUT: A set S of nodes where $S \subseteq V$ such that the final activations $f(S)$ is maximized and $|S| \leq K$.

Activation Models

The influence maximization problem was first proposed by Domingos et al. and Richardson

Influence Maximization, Fig. 1 Pseudo-code: Greedy algorithm

Greedy Algorithm

1: let $S \leftarrow \emptyset$ (S holds the selected nodes);
2: **while** $|S| \leq K$ **do**
3: find $v \in (V \setminus S)$ such that $f(S \cup \{v\})$ is maximized;
4: let $S \leftarrow S \cup \{v\}$;
5: **end while**

et al. in [4] and [8], respectively, in which the social networks are modeled as Markov random field. After that, Kempe et al. ([6] and [7]) further investigated this problem in two models: *Independent Cascade* proposed by Goldenberg et al. ([5] and [11]) and *Linear Threshold* proposed by Granovetter et al. and Schelling et al., respectively, in [9] and [10].

In the *Independent Cascade* model, the activations are independent among different individuals, i.e., each newly activated individual u will have a chance, in the next time slot, to activate his or her neighbors v with certain probability $p(u, v)$ which is independent with other activations. In the *Linear Threshold* model, the activation is based on a threshold manner; the influence from an individual u to another individual v is presented by a weight $w(i, j)$ and the individual v will be activated at the moment when the sum of weights he or she receives from previous activated neighbors exceeds the threshold $t(v)$. It is worthy to note that there are two ways to assign the thresholds to individuals: *random* and *deterministic*. In the random model, the thresholds are randomly selected at uniform during the time, while in the deterministic model, the thresholds are assigned to individuals at the beginning and fixed for all time slots. For the sake of simplicity, they are called *Random Linear Threshold* and *Deterministic Linear Threshold*, respectively.

Key Results

Greedy Algorithm

In [6], it has been found that the activation function f under the *Independent Cascade* model and the *Random Linear Threshold* model is sub-modular. Therefore, the natural greedy algorithm (Fig. 1), which selects the node with the maximum marginal gain repeatedly, achieves a $\left(1 - \frac{1}{e}\right)$-approximation solution. However, the problem of exactly calculating the activation function f in a general graph G under the *Independent Cascade* model or the *Random Linear Threshold* model, respectively, is #P-hard [1, 2], which indicates that the greedy algorithm is not a polynomial time algorithm for the two models. The time complexity directly follows the pseudo-code (Fig. 1). Assume there exists an oracle that can compute the activation function f in τ time, and then the greedy algorithm runs in $O(K|V|\tau)$ time.

In [13], it has been found that the problem of exactly calculating the activation function f given an arbitrary set S under the *Deterministic Linear Threshold* model can be solved in linear time in terms of the number of edges. Therefore, the greedy algorithm runs in $O(K|V||E|)$ time. However, it has no approximation guarantee under this model.

Inapproximation Results

Under the *Independent Cascade* model or the *Random Linear Threshold* model, it can be shown by doing a gap-preserving reduction from the Set Cover problem [3] that $\left(1 - \frac{1}{e}\right)$ is the best possible polynomial time approximation ratio for the influence maximization problem; assume NP $\not\subset$ DTIME $\left(n^{\log \log n}\right)$. Under the *Deterministic Linear Threshold* model, it has been shown that there is no polynomial time $n^{1-\epsilon}$-approximation algorithm for the influence maximization problem unless P = NP where n is the number of nodes and $0 < \epsilon < 1$ [12].

Actually in the case that an individual can be activated after one of his or her neighbors

becomes active, the greedy algorithm achieves a polynomial time $\left(1-\frac{1}{e}\right)$-approximation solution, and even in the simple case that an individual can be activated when one or two of his or her neighbors become active, the influence maximization problem under the *Deterministic Linear Threshold* model is NP-hard to approximate.

Degree-Bounded Graphs

A graph $G(V, E)$ is a (d_1, d_2)-degree-bounded graph if every node in V has at most d_1 incoming edges and at most d_2 outgoing edges.

For the sake of simplicity, the influence maximization problem over such a degree-bounded graph is called (d_1, d_2)-influence maximization. In [13], it has been found that for any constant $\epsilon \in (0, 1)$, there is no polynomial time $n^{1-\epsilon}$-approximation algorithm for the $(2, 2)$-influence maximization problem under the *Deterministic Linear Threshold* model unless P = NP where n is the number of nodes, which indicates that the influence maximization problem under *Deterministic Linear Threshold* model is NP-hard to approximate to within any nontrivial factor, even if an individual can be activated when at least two of his or her neighbors become active.

Applications

Influence maximization would be of great interest for corporations, such as Facebook, LinkedIn, and Twitter, as well as individuals who desire to spread their products, ideas, etc. The solutions have a wide range of applications in various fields, such as product promotions where corporations want to distribute sample products among customers, political elections where candidates want to spread their popularity or political ideas among voters, and emergency situations where emergency news like sudden earthquake needs to spread to every resident in the community. In addition, the solutions may be also applicable in military defense where malicious information which has already propagated dynamically needs to be blocked.

Cross-Reference

▸ Influence and Profit

Recommended Reading

1. Chen W, Yuan Y, Zhang L (2010) Scalable influence maximization in social networks under the linear threshold model. In: The 2010 international conference on data mining. Sydney, Australia
2. Chen W, Wang C, Wang Y (2009) Scalable influence maximization for prevalent viral marketing in large-scale social networks. In: The 2010 international conference on knowledge discovery and data mining. Washington DC, USA
3. Feige U (1998) A threshold of ln n for approximating set cover. J ACM 45:314–318
4. Domingos P, Richardson M (2001) Mining the network value of customers. In: The 2001 international conference on knowledge discovery and data mining. San Francisco, CA, USA
5. Goldenberg J, Libai B, Muller E (2001) Using complex systems analysis to advance marketing theory development. Acad Mark Sci Rev 9(3):1–18
6. Kempe D, Kleinberg J, Tardos É (2003) Maximizing the spread of influence through a social network. In: The 2003 international conference on knowledge discovery and data mining. Washington DC, USA
7. Kempe D, Kleinberg J, Tardos É (2005) Influential nodes in a diffusion model for social networks. In: The 2005 international colloquium on automata, languages and programming. Lisbon, Portugal
8. Richardson M, Domingos P (2002) Mining knowledge-sharing sites for viral marketing. In: The 2002 international conference on knowledge discovery and data mining. Edmonton, AB, Canada
9. Granovetter M (1978) Threshold models of collective behavior. Am J Sociol 83(6):1420–1443
10. Schelling T (1978) Micromotives and macrobehavior. Norton, New York
11. Goldenberg J, Libai B, Muller E (2001) Talk of the network: a complex systems look at the underlying process of word-of-mouth. Mark Lett 12(3): 211–223
12. Lu Z, Zhang W, Wu W, Kim J, Fu B (2012) The complexity of influence maximization problem in deterministic threshold mode. J Comb Optim 24(3):374–378
13. Lu Z, Zhang W, Wu W, Fu B, Du D (2011) Approximation and inapproximation for the influence maximization problem in social networks under deterministic linear threshold model. In: The 2011 international conference on distributed computing systems workshops. Minneapolis, USA

I

Intersections of Inverted Lists

Andrew Kane and Alejandro López-Ortiz
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Keywords

Algorithms; Bitvectors; Compression; Efficiency; Information retrieval; Intersection; Optimization; Performance; Query processing; Reordering

Years and Authors of Summarized Original Work

2014; Kane, Tompa

Problem Definition

This problem is concerned with efficiently finding a set of documents that closely match a query within a large corpus of documents (i.e., a search engine). This is accomplished by producing an index offline to the query processing and then using the index to quickly answer the queries. The indexing stage involves splitting the dataset into tokens and then constructing an *inverted index* which maps from each token to the list of document identifiers of the documents that contain that token (a *postings list*). The query can then be executed by converting it to a set of query tokens, using the inverted index to find the corresponding postings lists, and *intersecting* the lists to find the documents contained in all the lists (conjunctive intersection or boolean AND). A subsequent ranking step is used to restrict the conjunctive intersection to a list of top-k results that best answer the query.

Objective

Produce an efficient system to answer queries, where efficiency is a space-time trade-off involving the storage of the inverted index (space) and the intersection of the lists (time). If the inverted index is stored on a slow medium, then efficiency might also include the size of the lists required to answer the query (transfer).

Design Choices

There are many degrees of freedom in designing such a system:

1. Creating a document to internal identifier mapping
2. Encoding of the inverted index mapping
3. Encoding of the postings lists
4. Using auxiliary structures in postings lists
5. Ordering of the internal identifiers in the postings lists
6. Order and method of executing the list intersection

Variants

For queries where the conjunctive intersection is too small, the intersection can be relaxed to find documents containing a weighted portion of the query tokens (see t-threashold or Weak-AND). The query lists can be intersected using any boolean operators, though the most commonly added is the boolean NOT operator which can be used to quickly reduce the number of conjunctive results. The query results can also be reduced by including token offset restrictions, such as ensuring tokens appear as a phrase or within some proximity. While many implementations interleave the conjunctive intersection with the calculation of the ranking, some also use ranking information to prune documents from the intersection or to terminate the query early when the correct results (or good enough results) have been found.

Key Results

Traditionally, inverted indexes were stored on disk, causing the reduction of transfer costs to be the dominant objective. Modern systems often store their inverted indexes in memory meaning that reducing overall index size is important and that implementation details of the intersection algorithms can produce significant performance

differences, thus leading to a more subtle space-time trade-off. In either case, the mapping portion of the inverted index (the dictionary or lexicon) can be implemented using a data structure such as a B-tree which is both fast and compact, so we do not examine dictionary implementations in the remainder of this article.

Multi-list Processing

Intersecting multiple lists can be implemented by intersecting the two smallest lists and then intersecting the result with the next smallest list iteratively, thus producing a *set versus set* (svs) or *term-at-a-time* (TAAT) approach. If the lists are in sorted order, then each step of the svs approach uses the *merge* algorithm, which takes each element in the smaller list M and finds it in the larger list N by executing a forward search and then reports any elements that are found. The M list could be encoded differently than the N list, and indeed, after the first svs step, it is the uncompressed result list of the previous step. The sequential processing and memory accesses of the svs approach allows the compiler and CPU to optimize the execution, making this approach extremely fast, even though temporary memory is required for intermediate result accumulators. If the lists are not sorted, then additional temporary memory must be used to intersect the lists using some equality-join algorithm.

Intersecting multiple lists can also be implemented by intersecting all the lists at the same time. If the lists are not in sorted order, using this approach may require a large amount of temporary space for the join structures and the accumulators. If the lists are sorted, then we call this a *document-at-a-time* (DAAT) approach, and it requires very little temporary space: just one pointer per list to keep track of its processing location. The order that the lists are intersected could be static, such as ascending term frequency order (as done with svs), or it could adapt to the processing. All of these non-svs approaches jump among the lists that are stored at different memory locations, so it is more difficult for the compiler and CPU to optimize the execution. In addition, the loop iterating over the lists for each result item and the complications when using different list encodings will slow down non-svs implementations. Despite these limitations, many of the optimizations for svs list intersection can be applied to implementations using non-svs approaches. For systems that return the top-ranked results, a small amount of additional memory is needed for a top-k heap to keep track of the best results. Instead of adding results that match all the terms into a simple array of results, they are added to the heap. At the end of query processing, the content of the heap is output in rank order to form the final top-k query results.

Uncompressed Lists

Storing the lists of document identifiers in an *uncompressed* format simply means using a sequential array of integers. For fast intersection, the integers in these lists are stored in order, thus avoiding join structures and allowing many methods of searching for a particular value in a list. As a result, there are many fast algorithms available for intersecting uncompressed integer lists, but the memory used to store the uncompressed lists is very large and probes into the list can produce wasted or inefficient memory access. All of these uncompressed intersection algorithms rely on random access into the lists, so they are inappropriate for compressed lists. We present only three of the best performing algorithms [2]:

Galloping svs (g-svs): Galloping forward search probes into the list to find a point past the desired value, where the probe distance doubles each time, then the desired location is found using binary search within the last two probe points.

Galloping swapping svs (g-swsvs): In the previous galloping svs algorithm, values from the smaller list are found in the larger list. Galloping swapping svs, however, finds values from the list with the smaller number of remaining integers in the other list, thus potentially swapping the roles of the lists.

Sorted Baeza-Yates using adaptive binary forward search (ab-sBY): The Baeza-Yates algorithm is a divide and conquer approach that finds the median value of the smaller list in the

larger list, splits the lists, and recurses. Adding matching values at the end of the recursion produces a sorted result list. The adaptive binary forward search variant uses binary search within the recursed list boundaries, rather than using the original list boundaries.

Compressed Lists

There are a large variety of compression algorithms available for sorted integer lists. The lists are first converted into differences minus one (i.e., deltas or d-gaps) to get smaller values, but this removes the ability to randomly access elements in the list. Next, a variable length encoding is used to reduce the number of bits needed to store the values, often grouping multiple values together to allow word or byte alignment of the groups and faster bulk decoding. The most common list compression algorithms are *Variable byte (vbyte)*, *PForDelta (PFD)*, and *Simple9 (S9)*. Recent work has improved decoding, and delta restore speeds for many list compression algorithms using vectorization [7]. Additional gains are possible by changing delta encoding to act on groups of values, thus improving runtime at the expense of using more space. Another recent approach called quasi-succinct indexing [10] first acts on the values as monotone sequences and then incorporates some delta encoding more deeply in the compression algorithm.

List Indexes

List indexes, also known as skip structures or auxiliary indexes, can be included to jump over values in the postings lists and thus avoid decoding, or even accessing, portions of the lists. The desired jump points can be encoded inline with the lists, but they are better stored in a separate contiguous memory location without compression, allowing fast scanning through the jump points. These list index algorithms can be used with compressed lists by storing the deltas of the jump points, but the block-based structure causes complications if the jump point is not byte or word aligned, as well as block aligned. The actual list values that are found in the skip structures can either be maintained within the original compressed list (overlaid) preserving fast iteration through the list, or the values could be extracted (i.e., removed) from the original compressed lists, giving a space reduction but slower iteration through the list.

A simple list index algorithm ("skipper" [9]) groups by a fixed number of elements storing every X^{th} element in a separate array structure, where X is a constant, so we refer to it as skips(X). When intersecting lists, the skip structure is scanned linearly to find the appropriate jump point into the compressed structure, where the decoding can commence. Using variable length skips is possible, such as tuning the number of skipped values relative to the list size n, perhaps using a multiple of $\sqrt{n}$ or $\log(n)$.

Another type of list index algorithm ("lookup" [9]) groups by a fixed size document identifier range using the top-level bits of the value to index into an array storing the desired location in the encoded list, similar to a segment/offset scheme. Each list can pick the number of bits in order to produce reasonable jump sizes. We use D as the domain size and n as the list size, giving a list's density as $y = \frac{n}{D}$. If we assume randomized data and use the parameter B to tune the system, then by using $\left\lceil \log_2 \left(\frac{B}{y} \right) \right\rceil$ bottom level bits will leave between $\frac{B}{2}$ and B entries per segment in expectation. As a result, we call this algorithm segment(B).

Bitvectors

When using a compact domain of integers, as we are, the lists can instead be stored as bitvectors, where the bit number is the integer value and the bit is set if the integer is in the list. For runtime performance benefits, this mapping from the identifier to the bit location can be changed if it remains a one-to-one mapping and is applied to all bitvectors.

If all the lists are stored as bitvectors, conjunctive list intersection can be easily implemented by combining the bitvectors of the query terms using bitwise AND (bvand), with the final step converting the result to a list of integers (bvconvert). Note, except for the last step, the result of each step is a bitvector rather than an uncompressed result list. The bvconvert algorithm can

be implemented as a linear scan of the bits of each word, but using a logarithmic check is faster since the bitvectors being converted are typically sparse. Encoding all the lists as bitvectors gives good query runtime, but the space usage is very large since there are many tokens.

To alleviate the space costs of using bitvectors, the lists with density less than a parameter value F can be stored using normal delta compression, resulting in a hybrid bitvector algorithm [4]. This hybrid algorithm intersects the delta-compressed lists using merge and then intersects the remainder with the bitvectors by checking if the elements are contained in the first bitvector (bv-contains), repeating this for each bitvector in the query, with the final remaining values being the query result. Bitvectors are faster than other approaches for dense lists, so this hybrid algorithm is faster than non-bitvector algorithms. It can also be more compact than other compression schemes, because dense lists can be compactly stored as bitvectors. In addition, large overlaid skips can be used in the delta-compressed lists to improve query runtime.

In order to store more postings in the faster bitvector form, a semi-bitvector structure [5] encodes the front portion of a list as a bitvector and the rest using skips over delta compression. By skewing lists to have dense front portions, this approach can improve both space and runtime.

Other Approaches

Quasi-succinct indices [10] store list values in blocks with the lower bits of the values in an array and the higher bits as deltas using unary encoding combined with skips for fast access. This structure produces a good space-time trade-off when the number of higher-level bits is limited. The list intersection implementation can exploit the unary encoding of the higher-level bits by counting the number of ones in machine words to find values quickly. The resultant space-time performance is comparable to various skip-type implementations for conjunctive list intersection, though indexing speed may be slower.

The treap data structure combines the functionality of a binary tree and a binary heap. This treap data structure can implement list intersection [6] by storing each list as a treap where the list values are used as the tree order and the frequencies are used as the heap order. During list intersection, subtrees can be pruned from the processing if the frequency is too low to produce highly ranked results. In order to make this approach viable, low-frequency values are stored in separate lists using delta-compression with skips. This treap and frequency separated index structure can produce some space-time performance improvements compared to existing ranking-based search systems.

Wavelet trees can also be used to implement list intersection [8]. The postings lists are ordered by frequency, and each value is assigned a global location, so that each list can be represented by a range of global locations. A series of bit sequences represents a tree which starts with the frequency-ordered lists of the global locations and translates them into the actual document identifiers. Each level of the tree splits the document identifier domain ranges in half and encodes the edges of the tree using the bit sequences (left as 0 and right as 1). Multiple lists can be intersected by following the translation of their document ranges in this tree of bit sequences, only following the branch if it occurs in all of the list translations. After some careful optimization of the translation code, the wavelet tree data structure results in similar space usage, but faster query runtimes than some existing methods for conjunctive queries.

Ranking

Ranking algorithms are closely guarded trade secrets for large web search companies, so their details are not generally known. Many approaches, however, add term frequencies and/or postings offsets into the postings lists and combine this with corpus statistics to produce good results, as done in the standard BM25 approach. Unfortunately, information such as frequencies cannot be easily added to bitvector structures, thus limiting their use. Data and link analysis can also help by producing a global order, such as PageRank, which can be factored into the ranking function to improve results.

Reordering

Document identifiers in postings lists can be assigned to make the identifier deltas smaller and more compressible. As a first stage, they are assigned to form a compact domain of values, while a second stage renumbers these identifiers to optimize the system performance in a process referred to as document *reordering* [3]. Reordering can improve space usage by placing documents with similar terms close together in the ordering, thus reducing the size of the deltas, which can then be stored more compactly, among other benefits.

Applications

Intersecting inverted lists is at the heart of search engine query processing and top-k operators in databases.

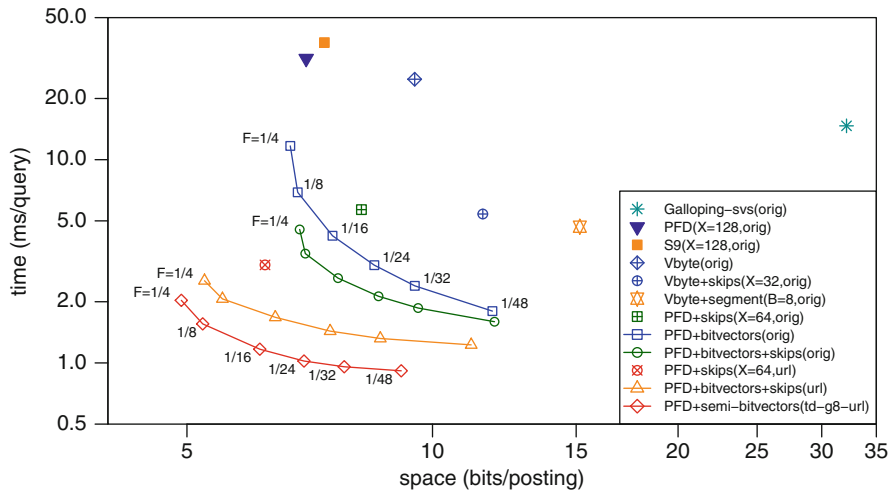
Open Problems

Going forward, main goal is to design list representations which are provably optimal in terms of space usage (entropy) together with algorithms that achieve the optimal trade-off between time and the space used by the given representation.

Experimental Results

Most published works involving intersections of inverted lists prove their results through experimentation. Following this approach, we show the relative performance of the described approaches using an in-memory conjunctive list intersection system that has document ordered postings lists without any ranking-based structures or processing. We run this system on the TREC GOV2 corpus and execute 5,000 of the associated queries. These experiments were executed on an AMD Phenom II X6 1090T 3.6 GHz Processor with 6 GB of memory running Ubuntu Linux 2.6.32-43-server. The code was compiled using the gcc 4.4.3 compiler with the -O3 command line parameter. Our results are presented in Fig. 1 using a space-time (log-log) plot. For the configurations considered, the block size of the encoding always equals the skip size, X, and the bitvector configurations all use $X = 256$.

With compressed lists alone, intersection is slow. On the other hand, uncompressed lists are much larger than compressed ones, but random access allows them to be fast. List indexes when combined with the compressed lists add some space, but their targeted access into the lists allows them to be even faster than the uncompressed algorithms. (For the list indexes, we present the fastest configurations we tested over the parameter ranges of X and B.)



Intersections of Inverted Lists, Fig. 1 Space vs. time (log-log) plot for various intersection algorithms

The performance of list indexes suggests that the benefits of knowing where to probe into the list (i.e., using skips rather than a probing search routine) outweigh the cost of decoding the data at that probe location. Using the hybrid bitvector approach is much faster and somewhat smaller than the other techniques. Adding large overlaid skips to the delta-compressed lists allows the bitvectors + skips algorithm to improve performance.

Reordering the documents to be in URL order gives significant space improvements. This ordering also improves query runtimes significantly, for all combinations of skips and/or bitvectors. Splitting the documents into eight groups by descending number of terms in document, reordering within the groups by URL ordering (td-g8-url), and using semi-bitvectors produce additional improvements in both space and runtime. This demonstration of the superior performance of bitvectors suggests that integrating them into ranking-based systems warrants closer examination.

URLs to Code and Datasets

Several standard datasets and query workloads are available from the Text REtreival Conference (TREC at http://trec.nist.gov). Many implementations of search engines are available in the open source community, including Wumpus (http://www.wumpus-search.org), Zettair (http://www.seg.rmit.edu.au/zettair/), and Lucene (http://lucene.apache.org).

Cross-References

▸ Compressing Integer Sequences

Recommended Reading

1. Anh VN, Moffat A (2005) Inverted index compression using word-aligned binary codes. Inf Retr 8(1):151–166
2. Barbay J, López-Ortiz A, Lu T, Salinger A (2009) An experimental investigation of set intersection algorithms for text searching. J Exp Algorithmics 14:3.7, 1–24
3. Blandford D, Blelloch G (2002) Index compression through document reordering. In: Proceedings of the data compression conference (DCC), Snowbird. IEEE, pp 342–351
4. Culpepper JS, Moffat A (2010) Efficient set intersection for inverted indexing. ACM Trans Inf Syst 29(1):1, 1–25
5. Kane A, Tompa FW (2014) Skewed partial bitvectors for list intersection. In: Proceedings of the 37th ACM international conference on research and development in information retrieval (SIGIR), Gold Coast. ACM, pp 263–272
6. Konow R, Navarro G, Clarke CLA, López-Ortiz A (2013) Faster and smaller inverted indices with treaps. In: Proceedings of the 36th ACM international conference on research and development in information retrieval (SIGIR), Dubin. ACM, pp 193–202
7. Lemire D, Boytsov L (2013) Decoding billions of integers per second through vectorization. Softw Pract Exp. doi: 10.1002/spe.2203. To appear
8. Navarro G, Puglisi SJ (2010) Dual-sorted inverted lists. In: String processing and information retrieval (SPIRE), Los Cabos. Springer, pp 309–321
9. Sanders P, Transier F (2007) Intersection in integer inverted indices. In: Proceedings of the 9th workshop on algorithm engineering and experiments (ALENEX), New Orlean. SIAM, pp 71–83
10. Vigna S (2013) Quasi-succinct indices. In: Proceedings of the 6th international conference on web search and data mining (WSDM), Rome. ACM, pp 83–92
11. Zukowski M, Heman S, Nes N, Boncz P (2006) Super-scalar RAM-CPU cache compression. In: Proceedings of the 22nd international conference on data engineering (ICDE), Atlanta. IEEE, pp 59.1–59.12

Intrinsic Universality in Self-Assembly

Damien Woods
Computer Science, California Institute of Technology, Pasadena, CA, USA

Keywords

Abstract Tile Assembly Model; Intrinsic universality; Self-assembly; Simulation

Years and Authors of Summarized Original Work

2012; Doty, Lutz, Patitz, Schweller, Summers, Woods
2013; Demaine, Patitz, Rogers, Schweller, Summers, Woods

2014; Meunier, Patitz, Summers, Theyssier, Winslow, Woods

2014; Demaine, Demaine, Fekete, Patitz, Schweller, Winslow, Woods

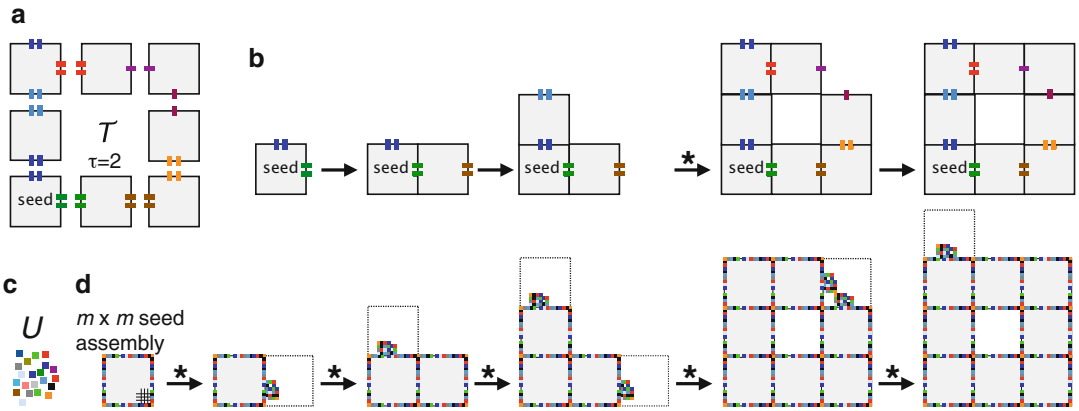
Problem Definition

Algorithmic self-assembly [11] is the idea that small self-assembling molecules can compute as they grow structures. It gives programmers a set of theoretical models in which to specify and design target structures while trying to optimize resources such as number of molecule types or even construction time. The abstract Tile Assembly Model [11] is one such model. An instance of the model is called a tile assembly system and is a triple $\mathcal{T} = (T, \sigma, \tau)$ consisting of a finite set T of square tiles, a seed assembly σ (one or more tiles stuck together), and a temperature $\tau \in \{1, 2, 3, \ldots\}$, as shown in Fig. 1a. Each side of a square tile has a glue (or color) g which in turn has a strength $s \in \{0, 1, 2, \ldots\}$. Growth occurs on the integer plane and begins from a seed assembly (or a seed tile) placed at the origin, as shown in Fig. 1b. A tile sticks to a partially formed assembly if it can be placed next to the assembly in such a way that enough of its glues match the glues of the adjacent tiles on the assembly and the sum of the matching glue strengths is at least the temperature. Growth proceeds one tile at a time, asynchronously and nondeterministically.

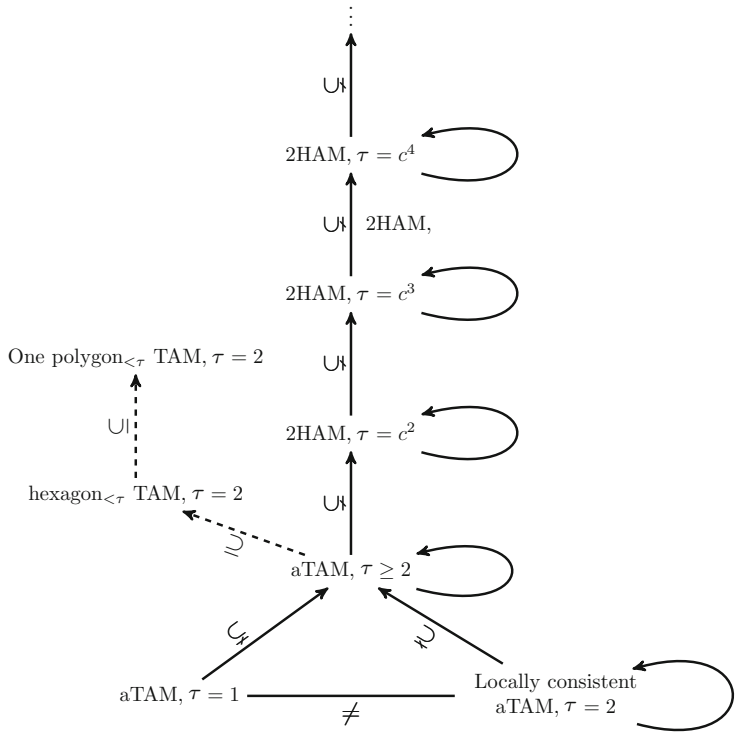
Here we discuss recent results and suggest open questions on intrinsic universality and simulation as a method to compare self-assembly models. Figure 2 gives an overview of these and other results. For more details, see [12].

Simulation and Intrinsic Universality

Intuitively, one self-assembly model simulates another if they grow the same structures, via the same dynamical growth processes, possibly with some spatial scaling. Let $\mathcal{S}$ and $\mathcal{T}$ be tile assembly systems of the abstract Tile Assembly Model described above. $\mathcal{S}$ is said to *simulate* $\mathcal{T}$ if the following conditions hold: (1) each tile of $\mathcal{T}$ is represented by one or more $m \times m$ blocks of tiles in $\mathcal{S}$ called *supertiles*, (2) the seed assembly



Intrinsic Universality in Self-Assembly, Fig. 1 An instance of the abstract Tile Assembly Model and an example showing simulation and intrinsic universality. (**a**) A tile assembly system $\mathcal{T}$ consists of a tile set, seed tile, and a temperature $\tau \in \mathbb{N}$. Colored glues on the tiles' sides have a natural number strength (shown here as 0, 1, or 2 colored tabs). (**b**) Growth begins from the seed with tiles sticking to the growing assembly if the sum of the strengths of the matching glues is at least τ. (**c**) An intrinsically universal tile set U. (**d**) When initialized with a seed assembly (which encodes $\mathcal{T}$) and at temperature 2, the intrinsically universal tile set simulates the dynamics of $\mathcal{T}$ with each tile placement in $\mathcal{T}$ being simulated by the growth of an $m \times m$ block of tiles. Single tile attachment is denoted by $\rightarrow$, and $\overset{*}{\rightarrow}$ denotes multiple tile attachments. Note that both systems have many other growth dynamics that are not shown



Intrinsic Universality in Self-Assembly, Fig. 2 Classes of tile assembly systems and their relationship with respect to simulation. There is an arrow from $\boldsymbol{B}$ to $\boldsymbol{A}$ if $\boldsymbol{A}$ contains $\boldsymbol{B}$ with respect to simulation: that is, for each tile assembly system $\mathcal{B} \in \boldsymbol{B}$, there is a tile assembly system $\mathcal{A}_{\mathcal{B}} \in \boldsymbol{A}$ that simulates $\mathcal{B}$. *Dashed arrows* denote containment, *solid arrows* denote strict containment, a self-loop denotes the existence of an intrinsically universal tile set for a class and its omission implies that the existence of such a tile set is an open problem. aTAM: abstract Tile Assembly Model (growth from a seed assembly by single tile addition in 2D), τ denotes "temperature." 2HAM: Two-Handed Tile Assembly Model (assemblies of tiles stick together in 2D). A 2HAM temperature hierarchy is shown for some $c \in \{2, 3, 4, \ldots\}$ and, in fact, for each such c the set of temperatures $\{c^i | i \in \{2, 3, \ldots\}\}$ gives an infinite hierarchy of classes of strictly increasing simulation power in the 2HAM

of $\mathcal{T}$ is represented by the *seed assembly* of $\mathcal{S}$ (one or more connected $m \times m$ supertiles), and (3) via supertile representation every sequence of tile placements in the simulated system $\mathcal{T}$ has a corresponding sequence of supertile placements in the simulator system $\mathcal{S}$, and vice versa. It is worth pointing out that although the intuitive idea of one assembly system simulating another is fairly simple, the formal definition of simulation [10] gets a little technical as the filling out of supertiles in the simulator is an asynchronous and nondeterministic distributed process with many supertiles growing independently and in parallel in the simulator system.

Key Results

The Abstract Tile Assembly Model Is Intrinsically Universal

A class of tile assembly systems C is said to be *intrinsically universal* if there exists a single set of tiles U that simulates any instance of C. For each such simulation, U should be appropriately initialized as an instance (i.e., a tile assembly system) of C itself. Figure 1d illustrates the concept. For example, the abstract Tile Assembly Model has been shown to be intrinsically universal [5]. Specifically, this means that there is a single set of tiles U that when appropriately initialized is

capable of simulating an arbitrary tile assembly system $\mathcal{T}$. To program such a simulation, tiles from $\mathcal{T}$ are represented as $m \times m$ supertiles (built from tiles in U), and the seed assembly of $\mathcal{T}$ is represented as a connected assembly $\sigma_{\mathcal{T}}$ of such supertiles. Furthermore, the entire tile assembly system $\mathcal{T}$ (a finite object) is itself encoded in the supertiles of $\sigma_{\mathcal{T}}$ of $\mathcal{U}$. Then if we watch all possible growth dynamics in both $\mathcal{T} = (T, \sigma, \tau)$ and $\mathcal{U} = (U, \sigma_{\mathcal{T}}, 2)$, we get that both systems produce the same set of assemblies via the same dynamics where we use a supertile representation function to map from supertiles over U to tiles from T. It is worth pointing out that in this particular construction [5], the simulating system is always (merely) at temperature $\tau = 2$ no matter how large the temperature ($\tau \geq 1$) of the simulated system.

This intrinsically universal tile set U has the ability to simulate both the geometry and growth order of any tile assembly system. Modulo spatial rescaling U represents the full power and expressivity of the entire abstract Tile Assembly Model.

Noncooperative Assembly Is Weaker than Cooperative Assembly

The temperature 1, or noncooperative, model is a restriction of the abstract Tile Assembly Model. Despite its esoteric name, it models a fundamental and ubiquitous form of growth: asynchronous growing and branching tips in Euclidian space where each new tile is added if it matches on at *least one side*. Separating the power of the noncooperative and cooperative models has presented significant challenge to the community.

Recently it has been shown that the noncooperative model is provably weaker than the full model [10] in that sense that it is not capable of *simulating* arbitrary tile assembly systems. This is the first fully general negative result about temperature 1 that does not assume restrictions on the model nor unproven hypotheses.

An interesting aspect of this result is that it holds for 3D noncooperative systems; they too cannot simulate arbitrary tile assembly systems. This seems quite shocking, given that 3D noncooperative systems are Turing-universal [1]! So in particular, 3D noncooperative systems can simulate 2D (or 3D) cooperative systems by simulating a Turing machine that in turn simulates the cooperative system, but this loose style of simulation ends up destroying the geometry and dynamics of tile assembly by encoding everything as "geometry-less" strings. Hence, Turing-universal algorithmic behavior in self-assembly does not imply the ability to simulate, in a direct geometric fashion, arbitrary algorithmic self-assembly processes.

One Tile to Rule Them All

As an example of a simulation result on a very different model of self-assembly, Demaine, Demaine, Fekete, Patitz, Schweller, Winslow, and Woods [4] describe a sequence of simulations that route from square tiles, to the intrinsically universal tile set, to hexagons (with strength $< \tau$, or weak, glues), to a *single* polygon that is translatable, rotatable, and flipable. Their fixed-sized polygon, when appropriately seeded, simulates any tile assembly system from the abstract Tile Assembly Model. They also show that with translation only (i.e., no rotation), such results are not possible with a small (size ≤ 3) seed (although with larger seeds, a single translation-only polyomino simulates the space-time diagram of a 1D cellular automaton). In the simpler setting of Wang plane tiling, they give an easy method to "compile" any tile set T (on the square or hexagonal lattice) to a single regular polygon that simulates exactly the tilings of T, except with tiny gaps between the polygons.

Two Hands

It has been shown that the two-handed, or hierarchical, model of self-assembly (where large assemblies of tiles may come together in a single step) is not intrinsically universal [3]. Specifically there is no tile set that, in the two-handed model, can simulate all two-handed systems for all temperatures. However, for each $\tau \in \{2, 3, 4, \ldots\}$, there is a tile set U_τ that is intrinsically universal for the class of two-handed systems that work at temperature τ. Also, there is an infinite hierarchy of classes of such systems with each level strictly more powerful than the one below. In fact there are an infinite set of such hierarchies, as

described in the caption of Fig. 2. These results give a formalization of the intuition that multiple long-range interactions are more powerful than fewer long-range interactions in the two-handed model.

Open Problems

Gaps in Fig. 2 (i.e., missing solid arrows and missing models) suggest a variety of open questions. Also, it remains as future work to further tease apart the power of restrictions of the abstract Tile Assembly Model, for example, it remains open whether 2D noncooperative systems are intrinsically universal for themselves.

It is an open question whether or not the hexagonal Tile Assembly Model [4], various polygonal Tile Assembly Models [4, 7], the Nubot model [13], and Signal-Passing Tile Assembly Model [6, 9] are intrinsically universal. Furthermore, simulation could be used to tease apart the power of subclasses of these models.

Gilbert et al. [7] investigate the computational power of various kinds of polygonal tile assembly systems, showing that regular polygon tiles with >6 sides simulate Turing machines. What is the relationship between tile geometry and simulation power? Do more sides give strictly more simulation power?

A desirable feature of a simulator is not only that it simulates all possible dynamics of some simulated system, but that the probability of a given dynamics is roughly equal in both the simulated system and the simulator. Is there an intrinsically universal tile set with that property? Here, the probability of seeing a given dynamics or assembly in a simulator should be close to that of the simulated system, where "close" means, say, within a factor proportional to the spatial scaling.

Does there exist a tile set U for the abstract Tile Assembly Model, such that for any (adversarially chosen) seed assembly σ, at temperature 2, this tile assembly system simulates some tile assembly system $\mathcal{T}$? Moreover, U should be able to simulate all such members $\mathcal{T}$ of some nontrivial class S. U is a tile set that can do one thing and nothing else: simulate tile assembly systems from the class S. This question about U is inspired by the factor simulation question in CA [2].

Many algorithmic tile assembly systems use cooperative self-assembly to simulate Turing machines in a "zig-zag" fashion, as do a number of experimentally implemented systems. Can the negative result of [10] be extended to show 2D temperature 1 abstract Tile Assembly Model systems do not simulate zig-zag tile assembly systems?

There are a number of future research directions for the two-handed, or hierarchical, self-assembly model. One open question [3, 8] asks whether or not temperature τ two-handed systems can simulate temperature $\tau - 1$ two-handed systems. Another direction involves finding which aspects of the model (e.g., mismatches, excess binding strength, geometric blocking) are required for intrinsic universality at a given temperature, to better understand the intricacies of this very powerful, but natural, model.

Of course, there are many other ways to compare the power of self-assembly models: shape and pattern building, tile complexity, time complexity, determinism versus nondeterminism, and randomized (coin-flipping) algorithms in self-assembly. It remains as important future work to find relationships between these notions on the one hand and intrinsic universality and simulation on the other hand. Can ideas from intrinsic universality be used to answer questions about these notions?

Cross-References

- Active Self-Assembly and Molecular Robotics With Nubots
- Combinatorial Optimization and Verification in Self-Assembly
- Patterned Self-Assembly Tile Set Synthesis
- Randomized Self-Assembly
- Robustness in Self-Assembly
- Self-Assembly at Temperature 1
- Self-Assembly of Fractals

▶ Self-Assembly of Squares and Scaled Shapes
▶ Self-Assembly with General Shaped Tiles
▶ Temperature Programming in Self-Assembly

Acknowledgments A warm thanks to all of my coauthors on this topic. The author is supported by NSF grants 0832824, 1317694, CCF-1219274, and CCF-1162589.

Recommended Reading

1. Cook M, Fu Y, Schweller RT (2011) Temperature 1 self-assembly: deterministic assembly in 3D and probabilistic assembly in 2D. In: SODA 2011: Proceedings of the 22nd annual ACM-SIAM symposium on discrete algorithms, San Francisco. SIAM, pp 570–589
2. Delorme M, Mazoyer J, Ollinger N, Theyssier G (2011) Bulking II: classifications of cellular automata. Theor Comput Sci 412(30):3881–3905. doi: 10.1016/j.tcs.2011.02.024
3. Demaine ED, Patitz MJ, Rogers TA, Schweller RT, Summers SM, Woods D (2013) The two-handed tile assembly model is not intrinsically universal. In: ICALP: Proceedings of the 40th international colloquium on automata, languages and programming, Part 1, Riga. LNCS, vol 7965. Springer, pp 400–412. arxiv preprint arXiv:1306.6710 [cs.CG]
4. Demaine ED, Demaine ML, Fekete SP, Patitz MJ, Schweller RT, Winslow A, Woods D (2014) One tile to rule them all: simulating any tile assembly system with a single universal tile. In: ICALP: Proceedings of the 41st international colloquium on automata, languages, and programming, Copenhagen. LNCS, vol 8572. Springer, pp 368–379. arxiv preprint arXiv:1212.4756 [cs.DS]
5. Doty D, Lutz JH, Patitz MJ, Schweller RT, Summers SM, Woods D (2012) The tile assembly model is intrinsically universal. In: FOCS: Proceedings of the 53rd annual IEEE symposium on foundations of computer science, New Brunswick, pp 439–446. doi: 10.1109/FOCS.2012.76
6. Fochtman T, Hendricks J, Padilla JE, Patitz MJ, Rogers TA (2014) Signal transmission across tile assemblies: 3D static tiles simulate active self-assembly by 2D signal-passing tiles. Nat Comput 14(2):251–264
7. Gilbert O, Hendricks J, Patitz MJ, Rogers TA (2015) Computing in continuous space with self-assembling polygonal tiles. Tech. rep., arxiv preprint arXiv:1503.00327 [cs.CG]
8. Hendricks J, Patitz MJ, Rogers TA (2015) The simulation powers and limitations of higher temperature hierarchical self-assembly systems. In: MCU: Proceedings of the 7th international conference on machines, computations and universality, North Cyprus, to appear. Tech. rep., arXiv arXiv:1503.04502
9. Jonoska N, Karpenko D (2014) Active tile self-assembly, part 1: universality at temperature 1. Int J Found Comput Sci 25:141–163. doi:10.1142/S0129054114500087
10. Meunier PE, Patitz MJ, Summers SM, Theyssier G, Winslow A, Woods D (2014) Intrinsic universality in tile self-assembly requires cooperation. In: SODA: Proceedings of the ACM-SIAM symposium on discrete algorithms, Portland, pp 752–771. arxiv preprint arXiv:1304.1679 [cs.CC]
11. Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology
12. Woods D (2015) Intrinsic universality and the computational power of self-assembly. Philos Trans R Soc A Math Phys Eng Sci 373(2046):20140214
13. Woods D, Chen HL, Goodfriend S, Dabby N, Winfree E, Yin P (2013) Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In: ITCS: Proceedings of the 4th conference on innovations in theoretical computer science, Berkeley. ACM, pp 353–354. arxiv preprint arXiv:1301.2626 [cs.DS]

J

Jamming-Resistant MAC Protocols for Wireless Networks

Andréa W. Richa[1] and Christian Scheideler[2]
[1]School of Computing, Informatics, and Decision Systems Engineering, Ira A. Fulton Schools of Engineering, Arizona State University, Tempe, AZ, USA
[2]Department of Computer Science, University of Paderborn, Paderborn, Germany

Keywords

Adversarial models; Competitive analysis; Jamming; MAC protocol; Wireless communication

Years and Authors of Summarized Original Work

2008; Awerbuch, Richa, Scheideler
2010; Richa, Scheideler, Schmid, Zhang
2011; Richa, Scheideler, Schmid, Zhang
2012; Richa, Scheideler, Schmid, Zhang
2014; Ogierman, Richa, Scheideler, Schmid, Zhang

Motivation

The problem of coordinating the access to a shared medium is a central challenge in wireless networks. In order to solve this problem, a proper medium access control (MAC) protocol is needed. Ideally, such a protocol should not only be able to use the wireless medium as effectively as possible, but it should also be robust against a wide range of interference problems including jamming attacks. Interference problems from outside sources are usually ignored in theory but in practice it is important to take these into account, particularly because the ISM frequency band, which is the standard band used for wireless communication, is one of the most dirty frequency bands as it is affected by many devices like microwaves.

Problem Definition

We model inference from outside sources with the help of an adversary. In the most general model that we have published so far [9], our adversarial model is based on the most widely used model to capture interference problems, which is known as the SINR (signal-to-interference-and-noise ratio) model. In the SINR model, a message sent by node u is correctly received by node v if and only if $P_v(u)/(\mathcal{N} + \sum_{w \in S} P_v(w)) \geq \beta$ where $P_x(y)$ is the received power at node x of the signal transmitted by node y, $\mathcal{N}$ is the background noise, and S is the set of nodes $w \neq u$ that are transmitting at the same time as u. The threshold $\beta > 1$ depends on the desired rate, the modulation scheme, etc. When using the standard model for signal propagation, then this expression results in $(P(u)/d(u,v)^\alpha)/(\mathcal{N} + \sum_{w \in S}$

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

$P(w)/d(w,v)^\alpha) \geq \beta$ where $P(x)$ is the strength of the signal transmitted by x, $d(x,y)$ is the Euclidean distance between x and y, and α is the path-loss exponent. We assume that all nodes transmit with some fixed signal strength P and that $\alpha > 2 + \epsilon$ for some constant $\epsilon > 0$, which is usually the case in an outdoors environment.

In most theory papers on MAC protocols, the background noise $\mathcal{N}$ is either ignored (i.e., $\mathcal{N} = 0$) or assumed to behave like a Gaussian variable. This, however, is an oversimplification of the real world. There are many sources of interference producing a non-Gaussian noise such as electrical devices, temporary obstacles, coexisting networks, or jamming attacks. In order to capture a very broad range of noise phenomena, we model the background noise $\mathcal{N}$ (due to jamming or to environmental noise) with the aid of an adversary $\mathcal{ADV}$ that has a fixed energy budget within a certain time frame for each node v. More precisely, in our case, a message transmitted by a node u will be successfully received by node v if and only if

$$\frac{P/d(u,v)^\alpha}{\mathcal{ADV}(v) + \sum_{w \in S} P/d(w,v)^\alpha} \geq \beta, \quad (1)$$

where $\mathcal{ADV}(v)$ is the current noise level created by the adversary at node v. The goal is to design a MAC protocol that allows the nodes to successfully transmit messages under this model as long as this is in principle possible.

For the formal description and analysis, we assume a synchronized setting where time proceeds in synchronized time steps called *rounds*. In each round, a node u may either transmit a message or sense the channel, but it cannot do both. A node which is sensing the channel may either (i) sense an *idle* channel, (ii) sense a *busy* channel, or (iii) *receive* a packet. In order to distinguish between an idle and a busy channel, the nodes use a fixed noise threshold ϑ: if the measured signal power exceeds ϑ, the channel is considered busy, otherwise idle. Whether a message is successfully received is determined by the SINR rule described above. To leave some chance for the nodes to communicate, we restrict the adversary to be (B,T)-*bounded*: for each node v and time interval I of length T, a (B,T)-*bounded adversary* has an overall noise budget of $B \cdot T$ that it can use to increase the noise level at node v and that it can distribute among the time steps of I as it likes, depending on the current state of the nodes. This adversarial noise model is very general, since in addition to being adaptive, the adversary is allowed to make independent decisions on which nodes to jam at any point in time (provided that the adversary does not exceed its noise budget over a time window of size T).

Our goal is to design a *symmetric local-control* MAC protocol (i.e., there is no central authority controlling the nodes, and all the nodes are executing the same protocol) that has a constant competitive throughput against any (B,T)-bounded adversary as long as certain conditions (that are as general as possible) are met. In order to define what we mean by "competitive," we need some notation. The *transmission range* of a node v is defined as the disk with center v and radius r with $P/r^\alpha \geq \beta\vartheta$. Given a constant $\epsilon > 0$, a time step is called *potentially busy* at some node v if $\mathcal{ADV}(v) \geq (1-\epsilon)\vartheta$ (i.e., only a little bit of additional interference by the other nodes is needed so that v sees a busy channel). For a not potentially busy time step, it is still possible that a message sent by a node u within v's transmission range is successfully received by v. Therefore, as long as the adversary is forced to offer not potentially busy time steps due to its limited budget and every node has a least one other node in its transmission range, it is in principle possible for the nodes to successfully transmit messages. To investigate that formally, we use the following notation. For any time frame F and node v let $f_v(F)$ be the number of time steps in F that are not potentially busy at v and let $s_v(F)$ be the number of time steps in which v successfully receives a message. We call a protocol c-*competitive* for some time frame F if $\sum_{v \in V} s_v(F) \geq c \sum_{v \in V} f_v(F)$. An adversary is *uniform* if at any time step, $\mathcal{ADV}(v) = \mathcal{ADV}(w)$ for all nodes $v, w \in V$, which implies that $f_v(F) = f_w(F)$ for all nodes.

Key Results

We presented a MAC protocol called SADE which can achieve a c-competitive throughput where c only depends on ϵ and the path loss exponent α but not on the size of the network or other network parameters [9]. The intuition behind SADE is simple: each node v maintains a parameter p_v which specifies v's probability of accessing the channel at a given moment of time. That is, in each round, each node u decides to broadcast a message with probability p_v. (This is similar to classical random backoff mechanisms where the next transmission time t is chosen uniformly at random from an interval of size $1/p_v$.) The nodes adapt their p_v values over time in a multiplicative-increase multiplicative-decrease manner, i.e., the value is lowered in times when the channel is utilized (more specifically, we decrease p_v whenever a successful transmission occurs) or increased during times when the channel is idling. However, p_v will never exceed $\hat{p}$, for some sufficiently small constant $\hat{p} > 0$.

In addition to the probability value p_v, each node v maintains a time window estimate T_v and a counter c_v for T_v. The variable T_v is used to estimate the adversary's time window T: a good estimation of T can help the nodes recover from a situation where they experience high interference in the network. In times of high interference, T_v will be increased and the sending probability p_v will be decreased. Now we are ready to describe SADE in full detail.

Initially, every node v sets $T_v := 1$, $c_v := 1$, and $p_v := \hat{p}$. In order to distinguish between idle and busy rounds, each node uses a fixed noise threshold of ϑ.

The SADE protocol works in synchronized rounds. In every round, each node v decides with probability p_v to send a message. If it decides not to send a message, it checks the following two conditions:

- If v successfully receives a message, then $p_v := (1+\gamma)^{-1} p_v$.
- If v senses an idle channel (i.e., the total noise created by transmissions of other nodes and the adversary is less than ϑ), then $p_v := \min\{(1+\gamma)p_v, \hat{p}\}$, $T_v := \max\{1, T_v - 1\}$.

Afterward, v sets $c_v := c_v + 1$. If $c_v > T_v$ then it does the following – v sets $c_v := 1$ – and if there was no idle step among the past T_v rounds, then $p_v := (1+\gamma)^{-1} p_v$ and $T_v := T_v + 2$.

Given that $\gamma \in O(1/(\log T + \log\log n))$, one can show the following theorem, where n is the number of nodes and $N = \max\{n, T\}$.

Theorem 1 *When running* SADE *for at least* $\Omega((T \log N)/\epsilon + (\log N)^4/(\gamma\epsilon)^2)$ *time steps,* SADE *has a* $2^{-O((1/\epsilon)^{2/(\alpha-2)})}$*-competitive throughput for any* $((1-\epsilon)\vartheta, T)$*-bounded adversary as long as (a) the adversary is uniform and the transmission range of every node contains at least one node or (b) there are at least* $2/\epsilon$ *nodes within the transmission range of every node.*

SADE is an adaption of the MAC protocol described in [6] for Unit Disk Graphs that works in more realistic network scenarios considering physical interference. Variants of SADE have also been shown to be successful in other scenarios:

In [7] a variant called ANTIJAM is presented for a simpler wireless model but a more severe adversary called *reactive* adversary, which is an adversary that can base the jamming decision on the actions of the nodes in the current time step and not just the initial state of the system at the current time step. However, the adversary can only distinguish between the cases that at least one node is transmitting or no node is transmitting, i.e., it cannot determine whether a transmitted message is successfully received.

In [8] another variant called COMAC is presented for a simpler wireless model that can handle coexisting networks. Even if these networks cannot exchange any information and the number

of these networks is unknown, the protocol is shown to be competitive.

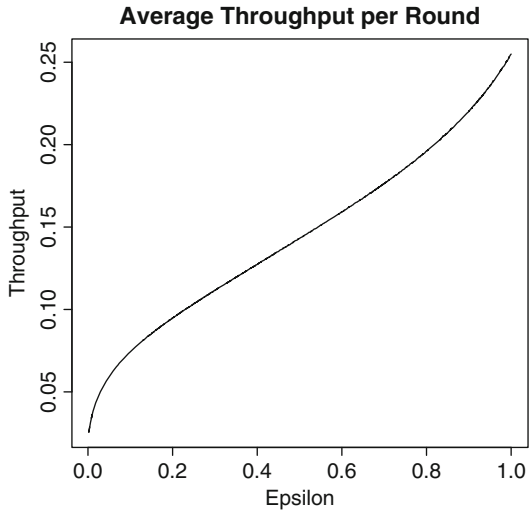
All of these results trace back to a first result in [1] for a very simple wireless model and the case of a single-hop wireless network.

Applications

Practical applications of our results are MAC protocols that are much more robust to outside interference and jamming than the existing protocols like 802.11. In fact, it is known that a much weaker jammer than the ones considered by us already suffices to dramatically reduce the throughput of the standard 802.11 MAC protocol [2].

Open Problems

So far, we have not considered the case of power control and multiple communication channels. Multiple communication channels have been covered in several other works (e.g., [3–5]) but under an adversary that is not as powerful as our adversary. Also, several of our bounds are not tight yet, so it remains to determine tight upper and lower bounds on the competitiveness of MAC protocols within our models.



Jamming-Resistant MAC Protocols for Wireless Networks, Fig. 1 The throughput with respect to varying ϵ

Experimental Results

We conducted various simulations to study the robustness of SADE. When varying ϵ, we found out that the worst-case bound of Theorem 1 may be too pessimistic in many scenarios, and the throughput depends to a lesser extent on the constant ϵ. To be more specific, our results suggest that the throughput depends only polynomially on ϵ (cf. the left-most image of Fig. 1), so more work is needed here.

Recommended Reading

1. Awerbuch B, Richa A, Scheideler C (2008) A jamming-resistant MAC protocol for single-hop wireless networks. In: Proc. of the 27th ACM Symp. on Principles of Distributed Computing (PODC), pp. 45–54, Toronto, Canada, 2008.
2. Bayraktaroglu E, King C, Liu X, Noubir G, Rajaraman R, Thapa B (2008) On the performance of IEEE 802.11 under jamming. In. Proc. of the 27th IEEE Conf. on Computer Communications (INFOCOM), pp. 1265–1273, Phoenix, AZ, USA, 2008.
3. Dolev S, Gilbert S, Guerraoui R, Newport C (2007) Gossiping in a multi-channel radio network: an oblivious approach to coping with malicious interference. In: Proc. of the 21st Intl. Symp. on Distributed Computing (DISC), pp. 208–222, 2007.
4. Dolev S, Gilbert S, Guerraoui R, Newport C (2008) Secure communication over radio channels. In: Proc. of the 27th ACM Symp. on Principles of Distributed Computing (PODC), pp. 105–114, Toronto, Canada, 2008.
5. Gilbert S, Guerraoui R, Kowalski D, Newport C (2009) Interference-resilient information exchange. In: Proc. of the 28th IEEE Conf. on Computer Communication (INFOCOM), pp. 2249–2257, Rio de Janeiro, Brazil.
6. Richa A, Scheideler C, Schmid S, Zhang J (2010) A jamming-resistant MAC protocol for multi-hop wireless networks. In: Proc. of the 24th Intl. Symp. on Distributed Computing (DISC), pp. 179–193, Cambridge, MA, USA, 2010.
7. Richa A, Scheideler C, Schmid S, Zhang J (2011) Competitive and fair medium access despite reactive jamming. In: Proc. of the 31st Intl. Conf. on Distributed Computing Systems (ICDCS), pp. 507–516, Minneapolis, MN, USA.
8. Richa A, Scheideler C, Schmid S, Zhang J (2012) Competitive and fair throughput for co-existing networks under adversarial interference. In: Proc. of the 31st ACM Symp. on Principles of Distributed Computing (PODC), pp. 291–300, Madeira, Portugal, 2012.
9. Richa A, Scheideler C, Schmid S, Zhang J (2014) Competitive MAC under adversarial SINR. In: Proc. of the 33rd IEEE Conf. on Computer Communication (INFOCOM), pp. 2751–2759, Toronto, Canada, 2014.

K

k-Best Enumeration

David Eppstein
Donald Bren School of Information and Computer Sciences, Computer Science Department, University of California, Irvine, CA, USA

Keywords

k minimum-weight matchings; k shortest paths; k shortest simple paths; k smallest spanning trees

Years and Authors of Summarized Original Work

1959; Hoffman, Pavley
1963; Clarke, Krikorian, Rausen
1968; Murty
1971; Yen
1972; Lawler
1977; Gabow
1982; Katoh, Ibaraki, Mine
1985; Hamacher, Queyranne
1987; Chegireddy, Hamacher
1993; Frederickson
1997; Eppstein, Galil, Italiano, Nissenzweig
1998; Eppstein
2007; Hershberger, Maxel, Suri
2013; Chen, Kanj, Meng, Xia, Zhang

Problem Definition

K-best enumeration problems are a type of combinatorial enumeration in which, rather than seeking a single best solution, the goal is to find a set of k solutions (for a given parameter value k) that are better than all other possible solutions. Many of these problems involve finding structures in a graph that can be represented by subsets of the graph's edges. In particular, the k shortest paths between two vertices s and t in a weighted network are a set of k distinct paths that are shorter than all other paths, and other problems such as the k smallest spanning trees of a graph or the k minimum weight matchings in a graph are defined in the same way.

Key Results

One of the earliest works in the area of k-best optimization was by Hoffman and Pavley [10] formulating the k-shortest path problem; their paper cites unpublished work by Bock, Kantner, and Hayes on the same problem. Later research by Lawler [12], Gabow [7], and Hamacher and Queyranne [8] described a general approach to k-best optimization, suitable for many of these problems, involving the hierarchical partitioning of the solution space into subproblems. One way

This material is based upon work supported by the National Science Foundation under Grant CCF-1228639 and by the Office of Naval Research under Grant No. N00014-08-1-1015.

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

of doing this is to view the optimal solution to a problem as a sequence of edges, and define one subproblem for each edge, consisting of the solutions that first deviate from the optimal solution at that edge. Continuing this subdivision recursively leads to a tree of subproblems, each having a worse solution value than its parent, such that each possible solution is the best solution for exactly one subproblem in the hierarchy. A best-first search of this tree allows the k-best solutions to be found. Alternatively, if both the first and second best solutions can be found, and differ from each other at an edge e, then one can form only two subproblems, one consisting of the solutions that include e and one consisting of the solutions that exclude e. Again, the subdivision continues recursively; each solution (except the global optimum) is the second-best solution for exactly one subproblem, allowing a best-first tree search to find the k-best solutions. An algorithm of Frederickson [6] solves this tree search problem in a number of steps proportional to k times the degree of the tree; each step involves finding the solution (or second-best solution) to a single subproblem.

Probably the most important and heavily studied of the k-best optimization problems is the problem of finding k shortest paths, first formulated by Hoffman and Pavley [10]. In the most basic version of this problem, the paths are allowed to have repeated vertices or edges (unless the input is acyclic, in which case repetitions are impossible). An algorithm of Eppstein [4] solves this version of the problem in the optimal time bound $O(m + n \log n + k)$, where m and n are the numbers of edges and vertices in the given graph; that is, after a preprocessing stage that is dominated by the time to use Dijkstra's algorithm to find a single shortest-path tree, the algorithm takes constant time per path. Eppstein's algorithm follows Hoffman and Pavley in representing a path by its sequence of *deviations*, the edges that do not belong to a tree T of shortest paths to the destination node. The deviation edges that can be reached by a path in T from a given node v are represented as a binary heap (ordered by how much additional length the deviation would cause) and these heaps are used to define a partition of the solution space into subproblems, consisting of the paths that follow a certain sequence of deviations followed by one more deviation from a specified heap. The best path in a subproblem is the one that chooses the deviation at the root of its heap, and the remaining paths can be partitioned into three sub-subproblems, two for the children of the root and one for the paths that use the root deviation but then continue with additional deviations. In this way, Eppstein constructs a tree of subproblems to which Frederickson's tree-searching method can be applied.

In a graph with cycles (or in an undirected graph which, when its edges are converted to directed edges, has many cycles), it is generally preferable to list only the k shortest simple (or loopless) paths, not allowing repetitions within a path. This variation of the k shortest paths problem was formulated by Clarke et al. [3]. Yen's algorithm [14] still remains the one with the best asymptotic time performance, $O(kn(m + n \log n))$; it is based on best-solution partitioning using Dijkstra's algorithm to find the best solution in each subproblem. A more recent algorithm of Hershberger et al. [9] is often faster, but is based on a heuristic that can sometimes fail, causing it to become no faster than Yen's algorithm. In the undirected case, it is possible to find the k shortest simple paths in time $O(k(m + n \log n))$ [11].

Gabow [7] introduced both the problem of finding the k minimum-weight spanning trees of an edge-weighted graph, and the technique of finding a binary hierarchical subdivision of the space of solutions, which he used to solve the problem. In any graph, the best and second-best spanning trees differ only by one edge swap (the removal of one edge from a tree and its replacement by a different edge that reconnects the two subtrees formed by the removal), a property that simplifies the search for a second-best tree as needed for Gabow's partitioning technique. The fastest known algorithms for the k-best spanning trees problem are based on Gabow's partitioning technique, together with dynamic graph data structures that keep track of the best swap in a network as that network undergoes a sequence of edge insertion and deletion operations. To use this technique, one initializes a fully-persistent best-swap data structure (one in

which each update creates a new version of the structure without modifying the existing versions, and in which updates may be applied to any version) and associates its initial version with the root of the subproblem tree. Then, whenever an algorithm for selecting the k best nodes of the subproblem tree generates a new node (a subproblem formed by including or excluding an edge from the allowed solutions) the parent node's version of the data structure is updated (by either increasing or decreasing the weight of the edge to force it to be included or excluded in all solutions) and the updated version of the data structure is associated with the child node. In this way, the data structure can be used to quickly find the second-best solution for each of the subproblems explored by the algorithm. Based on this method, the k-best spanning trees of a graph with n vertices and m edges can be found (in an implicit representation based on sequences of swaps rather than explicitly listing all edges in each tree) in time $O(\mathrm{MST}(m,n) + k \min(n,k)^{1/2})$ where $\mathrm{MST}(m,n)$ denotes the time for finding a single minimum spanning tree (linear time, if randomized algorithms are considered) [5].

After paths and spanning trees, probably the next most commonly studied k-best enumeration problem concerns matchings. The problem of finding the k minimum-weight perfect matchings in an edge-weighted graph was introduced by Murty [13]. A later algorithm by Chegireddy and Hamacher [1] solves the problem in time $O(kn^3)$ (where n is the number of vertices in the graph) using the technique of building a binary partition of the solution space. Other problems whose k-best solutions have been studied include the Chinese postman problem, the traveling salesman problem, spanning arborescences in a directed network, the matroid intersection problem, binary search trees and Huffman coding, chess strategies, integer flows, and network cuts.

For many NP-hard optimization problems, where even finding a single best solution is difficult, an approach that has proven very successful is *parameterized complexity*, in which one finds an integer parameter describing the input instance or its solution that is often much smaller than the input size, and designs algorithms whose running time is a fixed polynomial of the input size multiplied by a non-polynomial function of the parameter value. Chen et al. [2] extend this paradigm to k-best problems, showing that, for instance, many NP-hard k-best problems can be solved in polynomial time per solution for graphs of bounded treewidth.

Applications

The k shortest path problem has many applications. The most obvious of these are in the generation of alternative routes, in problems involving communication networks, transportation networks, or building evacuation planning. In bioinformatics, it has been applied to shortest-path formulations of dynamic programming algorithms for biological sequence alignment and also applied in the reconstruction of metabolic pathways, and reconstruction of gene regulation networks. The problem has been used frequently in natural language and speech processing, where a path in a network may represent a hypothesis for the correct decoding of an utterance or piece of writing. Other applications include motion tracking, genealogy, the design of power, communications, and transportation networks, timing analysis of circuits, and task scheduling.

The problem of finding the k-best spanning trees has been applied to point process intensity estimation, the analysis of metabolic pathways, image segmentation and classification, the reconstruction of pedigrees from genetic data, the parsing of natural-language text, and the analysis of electronic circuits.

Cross-References

- Minimum Spanning Trees
- Single-Source Shortest Paths

Recommended Reading

1. Chegireddy CR, Hamacher HW (1987) Algorithms for finding K-best perfect matchings. Discret Appl Math 18(2):155–165. doi:10.1016/0166-218X(87)90017-5

2. Chen J, Kanj IA, Meng J, Xia G, Zhang F (2013) Parameterized top-K algorithms. Theor Comput Sci 470:105–119. doi:10.1016/j.tcs.2012.10.052
3. Clarke S, Krikorian A, Rausen J (1963) Computing the N best loopless paths in a network. J SIAM 11:1096–1102
4. Eppstein D (1998) Finding the k shortest paths. SIAM J Comput 28(2):652–673. doi:10.1137/S0097539795290477
5. Eppstein D, Galil Z, Italiano GF, Nissenzweig A (1997) Sparsification–a technique for speeding up dynamic graph algorithms. J ACM 44(5):669–696. doi:10.1145/265910.265914
6. Frederickson GN (1993) An optimal algorithm for selection in a min-heap. Inf Comput 104(2):197–214. doi:10.1006/inco.1993.1030
7. Gabow HN (1977) Two algorithms for generating weighted spanning trees in order. SIAM J Comput 6(1):139–150. doi:10.1137/0206011
8. Hamacher HW, Queyranne M (1985) K best solutions to combinatorial optimization problems. Ann Oper Res 4(1–4):123–143. doi:10.1007/BF02022039
9. Hershberger J, Maxel M, Suri S (2007) Finding the k shortest simple paths: a new algorithm and its implementation. ACM Trans Algorithms 3(4):A45. doi:10.1145/1290672.1290682
10. Hoffman W, Pavley R (1959) A method for the solution of the Nth best path problem. J ACM 6(4):506–514. doi:10.1145/320998.321004
11. Katoh N, Ibaraki T, Mine H (1982) An efficient algorithm for K shortest simple paths. Networks 12(4):411–427. doi:10.1002/net.3230120406
12. Lawler EL (1972) A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. Manag Sci 18:401–405. doi:10.1287/mnsc.18.7.401
13. Murty KG (1968) Letter to the editor–an algorithm for ranking all the assignments in order of increasing cost. Oper Res 16(3):682–687. doi:10.1287/opre.16.3.682
14. Yen JY (1971) Finding the K shortest loopless paths in a network. Manag Sci 17:712–716. http://www.jstor.org/stable/2629312

Kernelization, Bidimensionality and Kernels

Daniel Lokshtanov
Department of Informatics, University of Bergen, Bergen, Norway

Keywords

Bidimensionality; Graph algorithms; Kernelization; Parameterized complexity; Polynomial time pre-processing

Years and Authors of Summarized Original Work

2010; Fomin, Lokshtanov, Saurabh, Thilikos

Problem Definition

The theory of bidimensionality simultaneously provides subexponential time parameterized algorithms and efficient approximation schemes for a wide range of optimization problems on planar graphs and, more generally, on classes of graphs excluding a fixed graph H as a minor. It turns out that bidimensionality also provides *linear kernels* for a multitude of problems on these classes of graphs. The results stated here unify and generalize a number of kernelization results for problems on planar graphs and graphs of bounded genus; see [2] for a more thorough discussion.

Kernelization

Kernelization is a mathematical framework for the study of polynomial time preprocessing of instances of computationally hard problems. Let $\mathcal{G}$ be the set of all graphs. A *parameterized graph problem* is a subset Π of $\mathcal{G} \times \mathbb{N}$. An *instance* is a pair $(G,k) \in \mathcal{G} \times \mathbb{N}$. The instance (G,k) is a "yes"-instance of Π if $(G,k) \in \Pi$ and a "no"-instance otherwise. A *strict kernel with ck vertices* for a parameterized graph problem Π and constant $c > 0$ is an algorithm $\mathcal{A}$ with the following properties:

- $\mathcal{A}$ takes as input an instance (G,k), runs in polynomial time, and outputs another instance (G',k').
- (G',k') is a "yes"-instance of Π if and only if (G,k) is.
- $|V(G')| \leq c \cdot k$ and $k' \leq k$.

A *linear kernel* for a parameterized graph problem is a strict kernel with ck vertices for some constant c. We remark that our definition of a linear kernel is somewhat simplified compared to the classic definition [8], but that it is essentially equivalent. For a discussion of the definition of a kernel, we refer to the textbook of Cygan et al. [4].

Graph Classes

Bidimensionality theory primarily concerns itself with graph problems where the input graph is restricted to be in a specific *graph class*. A *graph class* $\mathcal{C}$ is simply a subset of the set $\mathcal{G}$ of all graphs. As an example, the set of all planar graphs is a graph class. Another example of a graph class is the set of all *apex* graphs. Here a graph H is *apex* if H contains a vertex v such that deleting v from H leaves a planar graph. Notice that every planar graph is apex.

A graph H is a *minor* of a graph G if H can be obtained from G by deleting vertices, deleting edges, or contracting edges. Here *contracting* the edge $\{u, v\}$ in G means identifying the vertices u and v and removing all self-loops and double edges. If H can be obtained from G just by contracting edges, then H is a *contraction* of G.

A graph class $\mathcal{C}$ is *minor closed* if every minor of a graph in $\mathcal{C}$ is also in $\mathcal{C}$. A graph class $\mathcal{C}$ is *minor-free* if $\mathcal{C}$ is minor closed and there exists a graph $H \notin \mathcal{C}$. A graph class $\mathcal{C}$ is *apex-minor-free* if $\mathcal{C}$ is minor closed and there exists an apex graph $H \notin \mathcal{C}$. Notice that $H \notin \mathcal{C}$ for a minor closed class $\mathcal{C}$ implies that H cannot be a minor of any graph $G \in \mathcal{C}$.

CMSO Logic

CMSO logic stands for *Counting Monadic Second Order* logic, a formal language to describe properties of graphs. A *CMSO-sentence* is a formula ψ with variables for single vertices, vertex sets, single edges and edge sets, existential and universal quantifiers ($\exists$ and $\forall$), logical connectives $\vee$, $\wedge$ and $\neg$, as well as the following operators:

- $v \in S$, where v is a vertex variable and S is a vertex set variable. The operator returns true if the vertex v is in the vertex set S. Similarly, CMSO has an operator $e \in X$ where e is an edge variable and X is an edge set variable.
- $v_1 = v_2$, where v_1 and v_2 are vertex variables. The operator returns true if v_1 and v_2 are the same vertex of G. There is also an operator $e_1 = e_2$ to check equality of two edge variables e_1 and e_2.
- $\mathbf{adj}(v_1, v_2)$ is defined for vertex variables v_1 and v_2 and returns true if v_1 and v_2 are adjacent in G.
- $\mathbf{inc}(v, e)$ is defined for a vertex variable v and edge variable e. $\mathbf{inc}(v, e)$ returns true if the edge e is incident to the vertex v in G, in other words, if v is one of the two endpoints of e.
- $\mathbf{card}_{p,q}(S)$ is defined for every pair of integers p, q, and vertex or edge set variable S. $\mathbf{card}_{p,q}(S)$ returns true if $|S| \equiv q \mod p$. For an example, $\mathbf{card}_{2,1}(S)$ returns true if $|S|$ is odd.

When we quantify a variable, we need to specify whether it is a vertex variable, edge variable, vertex set variable, or edge set variable. To specify that an existentially quantified variable x is a vertex variable we will write $\exists x \in V(G)$. We will use $\forall e \in E(G)$ to universally quantify edge variables and $\exists X \subseteq V(G)$ to existentially quantify vertex set variables. We will always use lower case letters for vertex and edge variables and upper case letters for vertex set and edge set variables.

A graph G on which the formula ψ is true is said to *model* ψ. The notation $G \models \psi$ means that G models ψ. As an example, consider the formula

$$\psi_1 = \forall v \in V(G)\ \forall x \in V(G)\ \forall y \in V(G)\ \forall z \in V(G) : \\ (x = y) \vee (x = z) \vee (y = z) \vee \neg\mathbf{adj}(v, x) \vee \neg\mathbf{adj}(v, y) \vee \neg\mathbf{adj}(v, z)$$

The formula ψ_1 states that for every four (not necessarily distinct) vertices v, x, y, and z, if x, y, and z are distinct, then v is not adjacent to all of $\{x, y, z\}$. In other words, a graph G models ϕ_1 if and only if the degree of every vertex G is at most 2. CMSO can be used to express many

graph properties, such as G having a Hamiltonian cycle, G being 3-colorable, or G being planar.

In CMSO, one can also write formulas where one uses *free variables*. These are variables that are used in the formula but never quantified with an $\exists$ or $\forall$ quantifier. As an example, consider the formula

$$\psi_{DS} = \forall u \in V(G)\, \exists v \in V(G) : (v \in S) \wedge (u = v \vee \mathbf{adj}(u, v))$$

The variable S is a free variable in ψ_{DS} because it is used in the formula, but is never quantified. It does not make sense to ask whether a graph G models ψ_{DS} because when we ask whether the vertex v is in S, the set S is not well defined. However, if the set $S \subseteq V(G)$ is provided together with the graph G, we can evaluate the formula ψ_{DS}. ψ_{DS} will be true for a graph G and set $S \subseteq V(G)$ if, for every vertex $u \in V(G)$, there exists a vertex $v \in V(G)$ such that v is in S and either $u = v$ or u and v are neighbors in G. In other words, the pair (G, S) models ψ_{DS} (written $(G, S) \models \psi_{DS}$) if and only if S is a dominating set in G (i.e., every vertex not in S has a neighbor in S).

CMSO-Optimization Problems

We are now in position to define the parameterized problems for which we will obtain kernelization results. For every CMSO formula ψ with a single free vertex set variable S, we define the following two problems:

ψ-CMSO-Min (Max):

INPUT: *Graph G and integer k.*
QUESTION: *Does there exist a vertex set $S \subseteq V(G)$ such that $(G, S) \models \psi$ and $|S| \leq k$ ($|S| \geq k$ for Max).*

Formally, ψ-CMSO-MIN (MAX) is a parameterized graph problem where the "yes" instances are exactly the pairs (G, k) such that there exists a vertex set S of size at most k (at least k) and $(G, S) \models \psi$. We will use the term *CMSO-optimization problems* to refer to ψ-CMSO-MIN (MAX) for some CMSO formula ψ.

Many well-studied and not so well-studied graph problems are CMSO-optimization problems. Examples include VERTEX COVER, DOMINATING SET, CYCLE PACKING, and the list goes on and on (see [2]). We encourage the interested reader to attempt to formulate the problems mentioned above as CMSO-optimization problems. We will be discussing CMSO-optimization problems *on planar graphs* and on minor-free classes of graphs.

Our results are for problems where the input graph is promised to belong to a certain graph class $\mathcal{C}$. We formalize this by encoding membership in $\mathcal{C}$ in the formula ψ. For an example, ψ_{DS}-CMSO-MIN is the well-studied DOMINATING SET problem. If we want to restrict the problem to planar graphs, we can make a new CMSO logic formula ψ_{planar} such that $G \models \psi_{\text{planar}}$ if and only if G is planar. We can now make a new formula

$$\psi'_{DS} = \psi_{DS} \wedge \psi_{\text{planar}}$$

and consider the problem ψ'_{DS}-CMSO-MIN. Here (G, k) is a "yes" instance if G has a dominating set S of size at most k *and* G is planar. Thus, this problem also forces us to check planarity of G, but this is polynomial time solvable and therefore not an issue with respect to kernelization. In a similar manner, one can restrict any CMSO-optimization problem to a graph class $\mathcal{C}$, as long as there exists a CMSO formula $\psi_{\mathcal{C}}$ such that $G \models \psi_{\mathcal{C}}$ if and only if $G \in \mathcal{C}$. Luckily, such a formula is known to exist for every minor-free class $\mathcal{C}$. We will say that a parameterized problem Π is a problem *on the graph class $\mathcal{C}$* if, for every "yes" instance (G, k) of Π, the graph G is in $\mathcal{C}$.

For any CMSO-MIN problem Π, we have that $(G, k) \in \Pi$ implies that $(G, k') \in \Pi$ for all $k' \geq k$. Similarly, for a CMSO-MAX problem Π, we have that $(G, k) \in \Pi$ implies that $(G, k') \in \Pi$ for all $k' \leq k$. Thus, the notion of "optimality" is well defined for CMSO-optimization problems. For the problem $\Pi = \psi$-CMSO-MIN, we define

$$OPT_{\Pi}(G) = \min\{k \,:\, (G, k) \in \Pi\}.$$

If no k such that $(G,k) \in \Pi$ exists, $OPT_\Pi(G)$ returns $+\infty$. Similarly, for the problem $\Pi = \psi$-CMSO-MAX,

$$OPT_\Pi(G) = \max\{k \ : \ (G,k) \in \Pi\}.$$

If no k such that $(G,k) \in \Pi$ exists, $OPT_\Pi(G)$ returns $-\infty$. We define $SOL_\Pi(G)$ to be a function that given as input a graph G returns a set S of size $OPT_\Pi(G)$ such that $(G,S) \models \psi$ and returns **null** if no such set S exists.

Bidimensionality

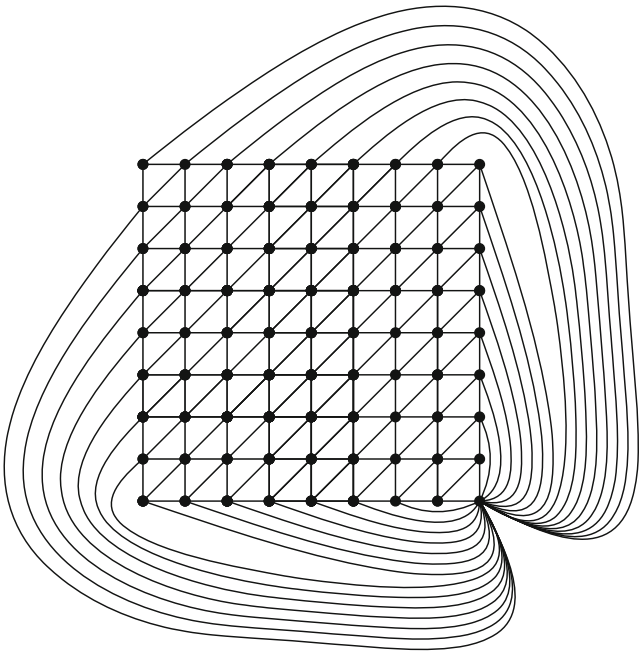
For many problems, it holds that contracting an edge cannot increase the size of the optimal solution. We will say that such problems are contraction closed. Formally, a CMSO-optimization problem Π is *contraction closed* if for any G and $uv \in E(G)$, $OPT_\Pi(G/uv) \leq OPT_\Pi(G)$. If contracting edges, deleting edges, and deleting vertices cannot increase the size of the optimal solution, we say that the problem is *minor closed*.

Informally, a problem is *bidimensional* if it is minor closed and the value of the optimum grows with both dimensions of a grid. In other words, on a $(k \times k)$-grid, the optimum should be approximately quadratic in k. To formally define bidimensional problems, we first need to define the $(k \times k)$-grid $\boxplus_k$, as well as the related graph Γ_k.

For a positive integer k, a $k \times k$ *grid*, denoted by $\boxplus_k$, is a graph with vertex set $\{(x,y) \ : \ x,y \in \{1,\ldots,k\}\}$. Thus, $\boxplus_k$ has exactly k^2 vertices. Two different vertices (x,y) and (x',y') are adjacent if and only if $|x-x'|+|y-y'| = 1$. For an integer $k > 0$, the graph Γ_k is obtained from the grid $\boxplus_k$ by adding in every grid cell the diagonal edge going up and to the right and making the bottom right vertex of the grid adjacent to all border vertices. The graph Γ_9 is shown in Fig. 1.

We are now ready to give the definition of bidimensional problems. A CMSO-optimization problem Π is *contraction-bidimensional* if it is contraction closed, and there exists a constant $c > 0$ such that $OPT_\Pi(\Gamma_k) \geq ck^2$. Similarly, Π is *minor-bidimensional* if it is minor closed, and there exists a constant $c > 0$ such that $OPT_\Pi(\boxplus_k) \geq ck^2$.

As an example, the DOMINATING SET problem is contraction-bidimensional. It is easy to verify that contracting an edge may not increase the size of the smallest dominating set of a graph G and that Γ_k does not have a dominating set of size smaller than $\frac{(k-2)^2}{7}$.



Kernelization, Bidimensionality and Kernels, Fig. 1 The graph Γ_9

Separability

Our kernelization algorithms work by recursively splitting the input instance by small separators. For this to work, the problem has to be somewhat well behaved in the following sense. Whenever a graph is split along a small separator into two independent sub-instances L and R, the size of the optimum solution for the graph $G[L]$ is relatively close to the size of the intersection between L and the optimum solution to the original graph G. We now proceed with a formal definition of what it means for a problem to be well behaved.

For a set $L \subseteq V(G)$, we define $\partial(L)$ to be the set of vertices in L with at least one neighbor outside L. A CMSO-optimization problem Π is *linear separable* if there exists a constant $c \geq 0$ such that for every set $L \subseteq V(G)$, we have

$$|SOL_\Pi(G) \cap L| - c \cdot |\partial(L)| \leq OPT_\Pi(G[L]) \leq |SOL_\Pi(G) \cap L| + c \cdot |\partial(L)|.$$

For a concrete example, we encourage the reader to consider the DOMINATING SET problem and to prove that for DOMINATING SET the inequalities above hold. The crux of the argument is to augment optimal solutions of G and $G[L]$ by adding all vertices in $\partial(L)$ to them.

Key Results

We can now state our main theorem.

Theorem 1 *Let Π be a separable CMSO-optimization problem on the graph class $\mathcal{C}$. Then, if Π is minor-bidimensional and $\mathcal{C}$ is minor-free, or if Π is contraction-bidimensional and $\mathcal{C}$ is apex-minor-free, Π admits a linear kernel.*

The significance of Theorem 1 is that it is, in general, quite easy to formulate graph problems as CMSO-optimization problems and prove that the considered problem is bidimensional and separable. If we are able to do this, Theorem 1 immediately implies that the problem admits a linear kernel on all minor-free graph classes, or on all apex-minor-free graph classes. As an example, the DOMINATING SET problem has been shown to have a linear kernel on planar graphs [1], and the proof of this fact is quite tricky. However, in our examples, we have shown that DOMINATING SET is a CMSO-MIN problem, that it is contraction-bidimensional, and that it is separable. Theorem 1 now implies that DOMINATING SET has a linear kernel not only on planar graphs but on all apex-minor-free classes of graphs! One can go through the motions and use Theorem 1 to give linear kernels for quite a few problems. We refer the reader to [9] for a non-exhaustive list.

We remark that the results stated here are generalizations of results obtained by Bodlaender et al. [2]. Theorem 1 is proved by combining "algebraic reduction rules" (fully developed by Bodlaender et al. [2]) with new graph decomposition theorems (proved in [9]). The definitions here differ slightly from the definitions in the original work [9] and appear here in the way they will appear in the journal version of [9].

Cross-References

- Bidimensionality
- Data Reduction for Domination in Graphs

Recommended Reading

1. Alber J, Fellows MR, Niedermeier R (2004) Polynomial-time data reduction for dominating set. J ACM 51(3):363–384
2. Bodlaender HL, Fomin FV, Lokshtanov D, Penninkx E, Saurabh S, Thilikos DM (2013) (Meta) Kernelization. CoRR abs/0904.0727. http://arxiv.org/abs/0904.0727
3. Borie RB, Parker RG, Tovey CA (1992) Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. Algorithmica 7(5&6):555–581
4. Cygan M, Fomin FV, Kowalik Ł, Lokshtanov D, Marx D, Pilipczuk M, Pilipczuk M, Saurabh S (2015, to appear) Parameterized algorithms. Springer, Heidelberg
5. Demaine ED, Hajiaghayi M (2005) Bidimensionality: new connections between FPT algorithms and PTASs. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms (SODA), Vancouver. SIAM, pp 590–601

6. Demaine ED, Hajiaghayi M (2008) The bidimensionality theory and its algorithmic applications. Comput J 51(3):292–302
7. Demaine ED, Fomin FV, Hajiaghayi M, Thilikos DM (2005) Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs. J ACM 52(6):866–893
8. Downey RG, Fellows MR (2013) Fundamentals of parameterized complexity. Texts in computer science. Springer, London
9. Fomin FV, Lokshtanov D, Saurabh S, Thilikos DM (2010) Bidimensionality and kernels. In: Proceedings of the 20th annual ACM-SIAM symposium on discrete algorithms (SODA), Austin. SIAM, pp 503–510
10. Fomin FV, Lokshtanov D, Raman V, Saurabh S (2011) Bidimensionality and EPTAS. In: Proceedings of the 21st annual ACM-SIAM symposium on discrete algorithms (SODA), San Francisco. SIAM, pp 748–759

Kernelization, Constraint Satisfaction Problems Parameterized above Average

Gregory Gutin
Department of Computer Science, Royal Holloway, University of London, Egham, UK

Keywords

Bikernel; Kernel; MaxCSP; MaxLin; MaxSat

Years and Authors of Summarized Original Work

2011; Alon, Gutin, Kim, Szeider, Yeo

Problem Definition

Let r be an integer, let $V = \{v_1, \ldots, v_n\}$ be a set of variables, each taking values -1 (TRUE) and 1 (FALSE), and let Φ be a set of Boolean functions, each involving at most r variables from V. In the problem MAX-r-CSP, we are given a collection $\mathcal{F}$ of m Boolean functions, each $f \in \mathcal{F}$ being a member of Φ and each with a positive integral weight. Our aim is to find a truth assignment that maximizes the total weight of satisfied functions from $\mathcal{F}$. We will denote the maximum by $\mathrm{sat}(\mathcal{F})$.

Let A be the average weight (over all truth assignments) of satisfied functions. Observe that A is a lower bound for $\mathrm{sat}(\mathcal{F})$. In fact, A is a tight lower bound, whenever the family Φ is closed under replacing each variable by its complement [1]. Thus, it is natural to parameterize MAX-r-CSP as follows (AA stands for *Above Average*).

MAX-r-CSP-AA

Instance: A collection $\mathcal{F}$ of m Boolean functions, each $f \in \mathcal{F}$ being a member of Φ, each with a positive integral weight, and a nonnegative integer k.
Parameter: k.
Question: $\mathrm{sat}(\mathcal{F}) \geq A + k$?

If Φ is the set of clauses with at most r literals, then we get a subproblem of MAX-r-CSP-AA, abbreviated MAX-r-SAT-AA, whose unparameterized version is simply MAX-r-SAT. Assign -1 or 1 to each variable in V randomly and uniformly. Since a clause c of an MAX-r-SAT-AA instance can be satisfied with probability $1 - 2^{r_c}$, where r_c is the number of literals in c, we have $A = \sum_{c \in \mathcal{F}} (1 - 2^{r_c})$. Clearly, A is a tight lower bound.

If Φ is the set S of equations $\prod_{i \in I_j} v_i = b_j$, $j = 1, \ldots, m$, where $v_i, b_j \in \{-1, 1\}$, b_js are constants, $|I_j| \leq r$, then we get a subproblem of MAX-r-CSP-AA, abbreviated MAX-r-LIN2-AA, whose unparameterized version is simply MAX-r-LIN2. Assign -1 or 1 to each variable in V randomly and uniformly. Since each equation of $\mathcal{F}$ can be satisfied with probability $1/2$, we have $A = W/2$, where W is the sum of the weights of equations in $\mathcal{F}$. For an assignment $v = v^0$ of values to the variables, let $\mathrm{sat}(S, v^0)$ denote the total weight of equations of S satisfied by the assignment. The difference $\mathrm{sat}(S, v^0) - W/2$ is called the *excess* of x^0. Let $\mathrm{sat}(S)$ be the maximum of $\mathrm{sat}(S, v^0)$ over all possible assignments v^0.

The following notion was introduced in [1]. Let Π and Π' be parameterized problems. A *bikernel* for Π is a polynomial-time algorithm that maps an instance (I,k) of Π to an instance (I',k') of Π' such that (i) $(I,k) \in \Pi$ if and only if $(I',k') \in \Pi'$ and (ii) $k' \leq g(k)$ and $|I'| \leq g(k)$ for some function g. The function $g(k)$ is called the *size* of the bikernel. It is known that a decidable problem is fixed-parameter tractable if and only if it admits a bikernel [1]. However, in general a bikernel can have an exponential size, in which case the bikernel may not be useful as a data reduction. A bikernel is called a *polynomial* bikernel if both $f(k)$ and $g(k)$ are polynomials in k.

When $\Pi = \Pi'$ we say that a bikernel for Π is simply a *kernel* of Π. A great deal of research has been devoted to decide whether a problem admits a polynomial kernel.

The following lemma of Alon et al. [1] shows that polynomial bikernels imply polynomial kernels.

Lemma 1 *Let Π, Π' be a pair of decidable parameterized problems such that the nonparameterized version of Π' is in NP and the nonparameterized version of Π is NP-complete. If there is a bikernelization from Π to Π' producing a bikernel of polynomial size, then Π has a polynomial-size kernel.*

Key Results

Following [2], for a Boolean function f of weight $w(f)$ and on $r(f) \leq r$ Boolean variables $v_{i_1}, \ldots, v_{i_{r(f)}}$, we introduce a polynomial $h_f(v)$, $v = (v_1, \ldots, v_n)$ as follows. Let $S_f \subseteq \{-1,1\}^{r(f)}$ denote the set of all satisfying assignments of f. Then

$$h_f(v) = w(f)2^{r-r(f)} \sum_{(a_1,\ldots,a_{r(f)}) \in S_f} \left[\prod_{j=1}^{r(f)} (1 + v_{i_j} a_j) - 1 \right].$$

Let $h(v) = \sum_{f \in \mathcal{F}} h_f(v)$. It is easy to see (cf. [1]) that the value of $h(v)$ at some v^0 is precisely $2^r(U - A)$, where U is the total weight of the functions satisfied by the truth assignment v^0. Thus, the answer to MAX-r-CSP-AA is YES if and only if there is a truth assignment v^0 such that $h(v^0) \geq k2^r$.

Algebraic simplification of $h(v)$ will lead us the following (Fourier expansion of $h(v)$, cf. [7]):

$$h(v) = \sum_{S \in \mathcal{F}} c_S \prod_{i \in S} v_i, \tag{1}$$

where $\mathcal{F} = \{\emptyset \neq S \subseteq \{1,2,\ldots,n\} : c_S \neq 0, |S| \leq r\}$. Thus, $|\mathcal{F}| \leq n^r$. The sum $\sum_{S \in \mathcal{F}} c_S \prod_{i \in S} v_i$ can be viewed as the excess of an instance of MAX-r-LIN2-AA, and, thus, we can reduce MAX-r-CSP-AA into MAX-r-LIN2-AA in polynomial time (since r is fixed, the algebraic simplification can be done in polynomial time and it does not matter whether the parameter of MAX-r-LIN2-AA is k or $k' = k2^r$). It is proved in [5] that MAX-r-LIN2-AA has a kernel with $O(k^2)$ variables and equations. This kernel is a bikernel from MAX-r-CSP-AA to MAX-r-LIN2-AA. Thus, by Lemma 1, we obtain the following theorem of Alon et al. [1].

Theorem 1 MAX-r-CSP-AA *admits a polynomial-size kernel.*

Applying a reduction from MAX-r-LIN2-AA to MAX-r-SAT-AA in which each monomial in (1) is replaced by 2^{r-1} clauses, Alon et al. [1] obtained the following:

Theorem 2 MAX-r-SAT-AA *admits a kernel with $O(k^2)$ clauses and variables.*

It is possible to improve this theorem with respect to the number of variables in the kernel. The following result was first obtained by Kim and Williams [6] (see also [3]).

Theorem 3 MAX-r-SAT-AA *admits a kernel with $O(k)$ variables.*

Crowston et al. [4] studied the following natural question: How parameterized complexity of MAX-r-SAT-AA changes when r is no longer

a constant, but a function $r(n)$ of n. They proved that MAX-$r(n)$-SAT-AA is para-NP-complete for any $r(n) \geq \lceil \log n \rceil$. They also proved that assuming the exponential time hypothesis, MAX-$r(n)$-SAT-AA is not even in XP for any integral $r(n) \geq \log\log n + \phi(n)$, where $\phi(n)$ is any real-valued unbounded strictly increasing computable function. This lower bound on $r(n)$ cannot be decreased much further as they proved that MAX-$r(n)$-SAT-AA is (i) in XP for any $r(n) \leq \log\log n - \log\log\log n$ and (ii) fixed-parameter tractable for any $r(n) \leq \log\log n - \log\log\log n - \phi(n)$, where $\phi(n)$ is any real-valued unbounded strictly increasing computable function. The proofs use some results on MAXLIN2-AA.

Cross-References

- Kernelization, MaxLin Above Average
- Kernelization, Permutation CSPs Parameterized above Average

Recommended Reading

1. Alon N, Gutin G, Kim EJ, Szeider S, Yeo A (2011) Solving MAX-k-SAT above a tight lower bound. Algorithmica 61:638–655
2. Alon N, Gutin G, Krivelevich M (2004) Algorithms with large domination ratio. J Algorithms 50: 118–131
3. Crowston R, Fellows M, Gutin G, Jones M, Kim EJ, Rosamond F, Ruzsa IZ, Thomassé S, Yeo A (2014) Satisfying more than half of a system of linear equations over GF(2): a multivariate approach. J Comput Syst Sci 80:687–696
4. Crowston R, Gutin G, Jones M, Raman V, Saurabh S (2013) Parameterized complexity of MaxSat above average. Theor Comput Sci 511:77–84
5. Gutin G, Kim EJ, Szeider S, Yeo A (2011) A probabilistic approach to problems parameterized above tight lower bound. J Comput Syst Sci 77: 422–429
6. Kim EJ, Williams R (2012) Improved parameterized algorithms for above average constraint satisfaction. In: IPEC 2011, Saarbrücken. Lecture notes in computer science, vol 7112, pp 118–131
7. O'Donnell R (2008) Some topics in analysis of Boolean functions. Technical report, ECCC report TR08-055. Paper for an invited talk at STOC'08. www.eccc.uni-trier.de/eccc-reports/2008/TR08-055/

Kernelization, Exponential Lower Bounds

Hans L. Bodlaender
Department of Computer Science, Utrecht University, Utrecht, The Netherlands

Keywords

Composition; Compression; Fixed parameter tractability; Kernelization; Lower bounds

Years and Authors of Summarized Original Work

2009; Bodlaender, Downey, Fellows, Hermelin

Problem Definition

Research on kernelization is motivated in two ways. First, when solving a hard (e.g., NP-hard) problem in practice, a common approach is to first *preprocess* the instance at hand before running more time-consuming methods (like integer linear programming, branch and bound, etc.). The following is a natural question. Suppose we use polynomial time for this preprocessing phase: what can be predicted of the size of the instance resulting from preprocessing? The theory of kernelization gives us such predictions. A second motivation comes from the fact that a decidable parameterized problem belongs to the class FPT (i.e., is *fixed parameter tractable*,) if and only if the problem has kernelization algorithm.

A parameterized problem is a subset of $\Sigma^* \times \mathbf{N}$, for some finite set Σ. A *kernelization algorithm* (or, in short *kernel*) for a parameterized problem $Q \subseteq \Sigma^* \times \mathbf{N}$ is an algorithm A that receives as input a pair $(x,k) \in \Sigma^* \times \mathbf{N}$ and outputs a pair $(x',k') = A(x,k)$, such that:

- A uses time, polynomial in $|x| + k$.
- $(x,k) \in Q$, if and only if $(x',k') \in Q$.
- There are functions f, g, such that $|x'| \leq f(k)$ and $k' \leq g(k)$.

In the definition above, f and g give an upper bound on the size, respectively the parameter of the reduced instance. Many well-studied problems have kernels with $k' \leq k$. The running time of an exact algorithm that starts with a kernelization step usually is exponential in the size of the kernel (i.e., $f(k)$), and thus small kernels are desirable. A kernel is said to be *polynomial*, if f and g are bounded by a polynomial. Many well-known parameterized problems have a polynomial kernel, but there are also many for which such a polynomial kernel is not known.

Recent techniques allow us to show, under a complexity theoretic assumption, for some parameterized problems that they do not have a polynomial kernel. The central notion is that of *compositionality*; with the help of transformations and cross compositions, a larger set of problems can be handled.

Key Results

Compositionality

The basic building block of showing that problems do not have a polynomial kernel (assuming $NP \not\subseteq coNP/poly$) is the notion of compositionality. It comes in two types: or-composition and and-composition.

Definition 1 An *or-composition* for a parameterized problem $Q \subseteq \Sigma^* \times \mathbf{N}$ is an algorithm that:

- Receives as input a sequence of instances for Q with the same parameter $(s_1, k), (s_2, k), \ldots, (s_r, k)$
- Uses time, polynomial in $k + \sum_{i=1}^{r} |s_i|$
- Outputs one instance for Q, $(s', k') \in \Sigma^* \times \mathbf{N}$, such that:

  1. $(s', k') \in Q$, if and only if there is an i, $1 \leq i \leq r$, with $(s_i, k) \in Q$.
  2. k' is bounded by a polynomial in k.

The notion of *and-composition* is defined similarly, with the only difference that condition (1) above is replaced by

$(s', k') \in Q$, if and only if for all i, $1 \leq i \leq r$: $(s_i, k) \in Q$.

We define the *classic variant* of a parameterized problem $Q \subseteq \Sigma^* \times \mathbf{N}$ as the decision problem, denoted Q^c where we assume that the parameter is encoded in unary, or, equivalently, an instance (s, k) is assumed to have size $|s| + k$.

Combining results of three papers gives the following results.

Theorem 1 *Let $Q \subseteq \Sigma^* \times \mathbf{N}$ be a parameterized problem. Suppose that the classic variant of Q, Q^c is NP-hard. Assume that $NP \not\subseteq coNP/poly$.*

1. *(Bodlaender et al. [3], Fortnow and Santhanam [12]) If Q has an or-composition, then Q has no polynomial kernel.*
2. *(Bodlaender et al. [3], Drucker [11]) If Q has an and-composition, then Q has no polynomial kernel.*

The condition that $NP \not\subseteq coNP/poly$ is equivalent to $coNP \not\subseteq NP/poly$; if it does not hold, the polynomial time hierarchy collapses to the third level [19].

For many parameterized problems, one can establish (sometimes trivially, and sometimes with quite involved proofs) that they are or-compositional or and-compositional. Taking the disjoint union of instances often gives a trivial composition. A simple example is the LONG PATH problem; it gets as input a pair (G, k) with G an undirected graph and asks whether G has a simple path of length at least k.

Lemma 1 *If $NP \not\subseteq coNP/poly$, then* LONG PATH *has no kernel polynomial in k.*

Proof LONG PATH is well known to be NP-complete. Mapping $(G_1, k), \ldots, (G_r, k)$ to the pair (H, k) with H the disjoint union of $G_1, \ldots, G_r$ is an or-composition. So, the result follows directly as a corollary of Theorem 1. □

The TREEWIDTH problem gets as input a pair (G, k) and asks whether the treewidth of G is most k. As it is NP-hard to decide if the treewidth of a given graph G is at most a given

number k [1] and the treewidth of a graph is the maximum of its connected components, taking the disjoint union gives an and-composition for the TREEWIDTH problem and shows that TREEWIDTH has no polynomial kernel unless $NP \subseteq coNP/poly$. Similar proofs work for many more problems. Many problems can be seen to be and- or or-compositional and thus have no polynomial kernels under the assumption that $NP \not\subseteq coNP/poly$. See, e.g., [3,5,9,17].

Transformations

Several researchers observed independently (see [2,5,9]) that transformations can be used to show results for additional problems. The formalization is due to Bodlaender et al. [5].

Definition 2 A *polynomial parameter transformation* (*ppt*) from parameterized problem $Q \subseteq \Sigma^* \times \mathbf{N}$ to parameterized problem $R \subseteq \Sigma^* \times \mathbf{N}$ is an algorithm A that:

- Has as input an instance of Q, $(s,k) \in \Sigma^* \times \mathbf{N}$.
- Outputs an instance of R, $(s',k') \in \Sigma^* \times \mathbf{N}$.
- $(s,k) \in Q$ if and only if $(s',k') \in R$.
- A uses time polynomial in $|s| + k$.
- k' is bounded by a polynomial in k.

The differences with the well-known polynomial time or Karp reductions from NP-completeness theory are small: note in particular that it is required that the new value of the parameter is polynomially bounded in the old value of the parameter. The following theorem follows quite easily.

Theorem 2 (See [5,6]) *Let R have a polynomial kernel. If there is a ppt from Q to R, and a polynomial time reduction from R to the classic variant of Q, then Q has a polynomial kernel.*

This implies that if we have a ppt from Q to R, Q^c is NP-hard, $R^c \in NP$, then when Q has no polynomial kernel, R has no polynomial kernel.

Cross Composition

Bodlaender et al. [4] introduced the concept of *cross composition*. It gives a more powerful mechanism to show that some problems have no polynomial kernel, assuming $NP \not\subseteq coNP/poly$. We need first the definition of a polynomial equivalence relation.

Definition 3 A *polynomial equivalence relation* is an equivalence relation on Σ^* that can be decided in polynomial time and has for each n, a polynomial number of equivalence classes that contain strings of length at most n.

A typical example may be that strings represent graphs and two graphs are equivalent if and only if they have the same number of vertices and edges.

Definition 4 Let L be a language, R a polynomial equivalence relation, and Q a parameterized problem. An *OR cross composition* of L to Q (w.r.t. R) is an algorithm that:

- Gets as input a sequence of instances $s_1, \ldots, s_r$ of L that belong to the same equivalence class of R.
- Uses time, polynomial in $\sum_{i=1}^{r} |s_i|$.
- Outputs an instance (s',k) of Q.
- k is polynomial in $\max |s_i| + \log k$.
- $(s',k) \in Q$ if and only if there is an i with $s_i \in L$.

The definition for an AND cross composition is similar; the last condition is replaced by

$(s',k) \in Q$ if and only if for all i with $s_i \in L$.

Theorem 3 (Bodlaender et al. [4]) *If we have an OR cross composition, or an AND cross composition from an NP-hard language L into a parameterized problem Q, then Q does not have a polynomial kernel, unless $NP \subseteq coNP/poly$.*

The main differences with or-composition and and-composition are we do not need to start with a collection of instances from Q, but can use a collection of instances of any NP-hard language; the bound on the new value of k usually allows us to restrict to collections of at most 2^k instances, and with the polynomial equivalence relation, we can make assumptions on "similarities" between these instances.

For examples of OR cross compositions, and of AND cross compositions, see, e.g., [4, 8, 13, 17].

Other Models and Improvements

Different models of compressibility and stronger versions of the lower bound techniques have been studied, including more general models of compressibility (see [11] and [7]), the use of co-nondeterministic composition [18], weak composition [15], Turing kernelization [16], and a different measure for compressibility based on witness size of problems in NP [14].

Problems Without Kernels

Many parameterized problems are known to be hard for the complexity class $W[1]$. As decidable problems are known to have a kernel, if and only if they are fixed parameter tractable, it follows that $W[1]$-hard problems do not have a kernel, unless $W[1] = FPT$ (which would imply that the exponential time hypothesis does not hold). See, e.g., [10].

Cross-References

- ▸ Kernelization, Polynomial Lower Bounds
- ▸ Kernelization, Preprocessing for Treewidth
- ▸ Kernelization, Turing Kernels

Recommended Reading

1. Arnborg S, Corneil DG, Proskurowski A (1987) Complexity of finding embeddings in a k-tree. SIAM J Algebr Discret Methods 8:277–284
2. Binkele-Raible D, Fernau H, Fomin FV, Lokshtanov D, Saurabh S, Villanger Y (2012) Kernel(s) for problems with no kernel: on out-trees with many leaves. ACM Trans Algorithms 8(5):38
3. Bodlaender HL, Downey RG, Fellows MR, Hermelin D (2009) On problems without polynomial kernels. J Comput Syst Sci 75:423–434
4. Bodlaender HL, Jansen BMP, Kratsch S (2011) Cross-composition: a new technique for kernelization lower bounds. In: Schwentick T, Dürr C (eds) Proceedings 28th international symposium on theoretical aspects of computer science, STACS 2011, Dortmund. Schloss Dagstuhl – Leibnitz-Zentrum fuer Informatik, Leibniz International Proceedings in Informatics (LIPIcs), vol 9, pp 165–176
5. Bodlaender HL, Thomassé S, Yeo A (2011) Kernel bounds for disjoint cycles and disjoint paths. Theor Comput Sci 412:4570–4578
6. Bodlaender HL, Jansen BMP, Kratsch S (2012) Kernelization lower bounds by cross-composition. CoRR abs/1206.5941
7. Chen Y, Flum J, Müller M (2011) Lower bounds for kernelizations and other preprocessing procedures. Theory Comput Syst 48(4):803–839
8. Cygan M, Kratsch S, Pilipczuk M, Pilipczuk M, Wahlström M (2012) Clique cover and graph separation: new incompressibility results. In: Czumaj A, Mehlhorn K, Pitts AM, Wattenhofer R (eds) Proceedings of the 39th international colloquium on automata, languages and programming, ICALP 2012, Part I, Warwick. Lecture notes in computer science, vol 7391. Springer, pp 254–265
9. Dom M, Lokshtanov D, Saurabh S (2009) Incompressibility through colors and IDs. In: Albers S, Marchetti-Spaccamela A, Matias Y, Nikoletseas SE, Thomas W (eds) Proceedings of the 36th international colloquium on automata, languages and programming, ICALP 2009, Part I, Rhodes. Lecture notes in computer science, vol 5555. Springer, pp 378–389
10. Downey RG, Fellows MR (2013) Fundamentals of parameterized complexity. Texts in computer science. Springer, London
11. Drucker A (2012) New limits to classical and quantum instance compression. In: Proceedings of the 53rd annual symposium on foundations of computer science, FOCS 2012, New Brunswick, pp 609–618
12. Fortnow L, Santhanam R (2011) Infeasibility of instance compression and succinct PCPs for NP. J Comput Syst Sci 77:91–106
13. Gutin G, Muciaccia G, Yeo: A (2013) (Non-)existence of polynomial kernels for the test cover problem. Inf Process Lett 113:123–126
14. Harnik D, Naor M (2010) On the compressibility of $\mathcal{NP}$ instances and cryptographic applications. SIAM J Comput 39:1667–1713
15. Hermelin D, Wu X (2012) Weak compositions and their applications to polynomial lower bounds for kernelization. In: Rabani Y (ed) Proceedings of the 22nd annual ACM-SIAM symposium on discrete algorithms, SODA 2012, Kyoto. SIAM, pp 104–113
16. Hermelin D, Kratsch S, Soltys K, Wahlström M, Wu X (2013) A completeness theory for polynomial (turing) kernelization. In: Gutin G, Szeider S (eds) Proceedings of the 8th international symposium on parameterized and exact computation, IPEC 2013, Sophia Antipolis. Lecture notes in computer science, vol 8246. Springer, pp 202–215
17. Jansen BMP, Bodlaender HL (2013) Vertex cover kernelization revisited – upper and lower bounds for a refined parameter. Theory Comput Syst 53:263–299
18. Kratsch S (2012) Co-nondeterminism in compositions: a kernelization lower bound for a Ramsey-type

problem. In: Proceedings of the 22nd annual ACM-SIAM symposium on discrete algorithms, SODA 2012, Kyoto, pp 114–122
19. Yap HP (1986) Some topics in graph theory. London mathematical society lecture note series, vol 108. Cambridge University Press, Cambridge

Kernelization, Matroid Methods

Magnus Wahlström
Department of Computer Science, Royal Holloway, University of London, Egham, UK

Keywords

Kernelization; Matroids; Parameterized complexity

Years and Authors of Summarized Original Work

2012; Kratsch, Wahlström

Problem Definition

Kernelization is the study of the power of polynomial-time instance simplification and preprocessing and relates more generally to questions of compact information representation. Given an instance x of a decision problem $\mathcal{P}$, with an associated *parameter* k (e.g., a bound on the solution size in x), a *polynomial kernelization* is an algorithm which in polynomial time produces an instance x' of $\mathcal{P}$, with parameter k', such that $x \in \mathcal{P}$ if and only if $x' \in \mathcal{P}$ and such that both $|x'|$ and k' are bounded by $p(k)$ for some $p(k) = \text{poly}(k)$. A *polynomial compression* is the variant where the output x' is an instance of a new problem $\mathcal{P}'$ (and may not have any associated parameter).

Matroid theory provides the tools for a very powerful framework for kernelization and more general information-preserving sparsification. As an example application, consider the following question. You are given a graph $G = (V, E)$ and two sets $S, T \subseteq V$ of terminal vertices, where potentially $|V| \gg |S|, |T|$. The task is to reduce G to a smaller graph $G' = (V', E')$, with $S, T \subseteq V'$ and $|V'|$ bounded by a function of $|S| + |T|$, such that for any sets $A \subseteq S$, $B \subseteq T$, the minimum (A, B)-cut in G' equals that in G. Here, all cuts are vertex cuts and may overlap A and B (i.e., the terminal vertices are also deletable). It is difficult to see how to do this without using both exponential time in $|S| + |T|$ (due to the large number of choices of A and B) and an exponential dependency of $|V'|$ on $|S|$ and $|T|$ (due to potentially having to include one min cut for every choice of A and B), yet using the appropriate tools from matroid theory, we can in polynomial time produce such a graph G' with $|V'| = O(|S| \cdot |T| \cdot \min(|S|, |T|))$. Call (G, S, T) a *terminal cut system*; we will revisit this example later.

The main power of the framework comes from two sources. The first is a class of matroids known as *gammoids*, which enable the representation of graph-cut properties as linear independence of vectors; the second is a tool known as the *representative sets lemma* (due to Lovász [4] via Marx [5]) applied to such a representation. To describe these closer, we need to review several definitions.

Background on Matroids

We provide only the bare essential definitions; for more, see Oxley [6]. Also see the relevant chapters of Schrijver [8] for a more computational perspective and Marx [5] for a concise, streamlined, and self-contained presentation of the issues most relevant to our concerns. For $s \in \mathbb{N}$, we let $[s]$ denote the set $\{1, \ldots, s\}$.

A *matroid* is a pair $M = (V, \mathcal{I})$ where V is a *ground set* and $\mathcal{I} \subseteq 2^V$ a collection of *independent sets*, subject to three axioms:

1. $\emptyset \in \mathcal{I}$.
2. If $B \in \mathcal{I}$ and $A \subseteq B$, then $A \in \mathcal{I}$.
3. If $A, B \in \mathcal{I}$ and $|A| < |B|$, then there exists some $b \in (B \setminus A)$ such that $A \cup \{b\} \in \mathcal{I}$.

All matroids we deal with will be finite (i.e., have finite ground sets). A set $S \subseteq V$ is *independent* in M if and only if $S \in \mathcal{I}$. A *basis* is a maximal independent set in M; observe that all bases of a matroid have the same cardinality. The *rank* of a set $X \subseteq V$ is the maximum cardinality of an independent set $S \subseteq X$; again, observe that this is well defined.

Linearly Represented Matroids

A prime example of a matroid is a *linear matroid*. Let A be a matrix over some field $\mathbb{F}$, and let V index the column set of A. Let $\mathcal{I}$ contain exactly those sets of columns of A that are linearly independent. Then $M = (V, \mathcal{I})$ defines a matroid, denoted $M(A)$, known as a linear matroid. For an arbitrary matroid M, if M is isomorphic to a linear matroid $M(A)$ (over a field $\mathbb{F}$), then M is *representable* (*over* $\mathbb{F}$), and the matrix A *represents* M. Observe that this is a compact representation, as $|\mathcal{I}|$ would in the general case be exponentially large, while the matrix A would normally have a coding size polynomial in $|V|$. In general, more powerful tools are available for linearly represented matroids than for arbitrary matroids (see, e.g., the MATROID MATCHING problem [8]). In particular, this holds for the representative sets lemma (see below).

Gammoids

The class of matroids central to our concern is the class of *gammoids*, first defined by Perfect [7]. Let $G = (V, E)$ be a (possibly directed) graph, $S \subseteq V$ a set of *source vertices*, and $T \subseteq V$ a set of *sink vertices* (where S and T may overlap). Let $X \subseteq T$ be independent if and only if there exists a collection of $|X|$ pairwise vertex-disjoint directed paths in G, each path starting in S and ending in X; we allow paths to have length zero (e.g., we allow a path from a vertex $x \in S \cap X$ to itself). This notion of independence defines a matroid on the ground set T, referred to as the *gammoid* defined by G, S, and T. By Menger's theorem, the rank of a set $X \subseteq T$ equals the cardinality of an (S, X)-min cut in G.

Gammoids are representable over any sufficiently large field [6], although only randomized procedures for computing a representation are known. An explicit randomized procedure was given in [2], computing a representation of the gammoid (G, S, T) in space (essentially) cubic in $|S| + |T|$. Hence, gammoids imply a polynomial-sized representation of terminal cut systems, as defined in the introduction. This has implications in kernelization [2], though it is not on its own the most useful form, since it is not a representation in terms of graphs.

Representative Sets

Let $M = (V, \mathcal{I})$ be a matroid, and X and Y independent sets in M. We say that Y *extends* X if $X \cup Y$ is independent and $X \cap Y = \emptyset$. The *representative sets lemma* states the following.

Lemma 1 ([4,5]) *Let $M = (V, \mathcal{I})$ be a linearly represented matroid of rank $r + s$, and let $\mathcal{S} = \{S_1, \ldots, S_m\}$ be a collection of independent sets, each of size s. In polynomial time, we can compute a set $\mathcal{S}^* \subseteq \mathcal{S}$ such that $|\mathcal{S}^*| \leq \binom{r+s}{s}$, and for any independent set X, there is a set $S \in \mathcal{S}$ that extends X if and only if there is a set $S' \in \mathcal{S}^*$ that extends X.*

We refer to $\mathcal{S}^*$ as a *representative set* for $\mathcal{S}$ in M. This result is due to Lovász [4], made algorithmic by Marx [5]; recently, Fomin et al. [1] improved the running time and gave algorithmic applications of the result. The power of the lemma is extended by several tools which construct new linearly represented matroids from existing ones; see Marx [5]. For a particularly useful case, for each $i \in [s]$ let $M_i = (V_i, \mathcal{I}_i)$ be a matroid, where each set V_i is a new copy of an original ground set V. Given a representation of these matroids over the same field $\mathbb{F}$, we can form a represented matroid $M = (V_1 \cup \ldots \cup V_s, \mathcal{I}_1 \times \ldots \times \mathcal{I}_s)$ as a *direct sum* of these matroids, where an independent set X in M is the union of an independent set X_i in M_i for each i. For an element $v \in V$, let $v(i)$ denote the copy of v in V_i. Then the set $\{v(1), \ldots, v(s)\}$ extends $X = X_1 \cup \ldots \cup X_s$ if and only if $\{v(i)\}$ extends X_i for each $i \in [s]$. In other words, we have constructed an *AND operation* for the notion of an extending set.

Closest Sets and Gammoids

We need one last piece of terminology. For a (possibly directed) graph $G = (V, E)$ and sets $A, X \subseteq V$, let $R_G(A, X)$ be the set of vertices reachable from A in $G \setminus X$. The set X is *closest to* A if there is no set X' such that $|X'| \leq |X|$ and X' separates X from A, i.e., $X \cap R_G(A, X') = \emptyset$. This is equivalent to X being the unique minimum (A, X)-vertex cut. For every pair of sets $A, B \subseteq V$, there is a unique minimum (A, B)-vertex cut closest to A, which can be computed in polynomial time. Finally, for sets S and X, the set X *pushed towards* S is the unique minimum (S, X)-vertex cut closest to S; this operation is well defined and has no effect if X is already closest to S. The following is central to our applications.

Lemma 2 *Let M be a gammoid defined from a graph $G = (V, E)$ and source set S. Let X be independent in M, and let X' be X pushed towards S. For any $v \in V$, the set $\{v\}$ extends X if and only if $v \in R_G(S, X')$.*

Key Results

The most powerful version of the terminal cut system result is the following.

Theorem 1 *Let $G = (V, E)$ be a (possibly directed) graph, and $X \subseteq V$ a set of vertices. In randomized polynomial time, we can find a set $Z \subseteq V$ of $|Z| = O(|X|^3)$ vertices such that for every partition $X = A \cup B \cup C \cup D$, the set Z contains a minimum (A, B)-vertex cut in the graph $G \setminus D$.*

There is also a variant for cutting into more than two parts, as follows.

Theorem 2 *Let $G = (V, E)$ be an undirected graph, and $X \subseteq V$ a set of vertices. In randomized polynomial time, we can find a set $Z \subseteq V$ of $|Z| = O(|X|^{s+1})$ vertices such that for every partition of X into at most s parts, the set Z contains a minimum solution to the corresponding multiway cut problem.*

We also have the following further kernelization results; see [3] for problem statements.

Theorem 3 *The following problems admit randomized polynomial kernels parameterized by the solution size*: ALMOST 2-SAT, VERTEX MULTICUT *with a constant number of cut requests, and* GROUP FEEDBACK VERTEX SET *with a constant-sized group.*

Applications

We now review the strategy behind kernelization usage of the representative sets lemma.

Representative Sets: Direct Usage

There have been various types of applications of the representative sets lemma in kernelization, from the more direct to the more subtle. We briefly review one more direct and one indirect. The most direct one is for reducing *constraint systems*. We illustrate with the DIGRAPH PAIR CUT problem (which is closely related to a central problem in kernelization [3]). Let $G = (V, E)$ be a digraph, with a source vertex $s \in V$, and let $\mathcal{P} \subseteq V^2$ be a set of pairs. The task is to find a set X of at most k vertices (with $s \notin X$) such that $R_G(s, X)$ does not contain any complete pair from $\mathcal{P}$. We show that it suffices to keep $O(k^2)$ of the pairs $\mathcal{P}$. For this, replace s by a set S of $k + 1$ copies of s, and let M be the gammoid of (G, S, V). By Lemma 2, if X is closest to S and $|X| \leq k$, then $\{u, v\} \subseteq R_G(s, X)$ if and only if both $\{u\}$ and $\{v\}$ extend X in M. Hence, using the direct sum construction, we can construct a representative set $\mathcal{P}^* \subseteq \mathcal{P}$ with $|\mathcal{P}^*| = O(k^2)$ such that for any set X closest to S, the set $R_G(s, X)$ contains a pair $\{u, v\} \in \mathcal{P}$ if and only if it contains a pair $\{u', v'\} \in \mathcal{P}^*$. Furthermore, for an arbitrary set X, pushing X towards S yields a set X' that can only be an improvement on X (i.e., the set of pairs in $R_G(s, X)$ shrinks); hence for any set X with $|X| \leq k$, either pushing X towards S yields a solution to the problem, or there is a pair in $\mathcal{P}^*$ witnessing that X is not a solution. Thus, the set $\mathcal{P}^*$ may be used to replace $\mathcal{P}$, taking the first step towards a kernel for the problem.

Indirect Usage

For more advanced applications, we "force" the lemma to reveal some set Z of special vertices in G, as follows. Let M be a linearly represented matroid, and let $\mathcal{S} = \{S(v) : v \in V\}$ be a collection of subsets of M of bounded size. Assume that we have shown that for every $z \in Z$, there is a carefully chosen set $X(z)$, such that $S(v)$ extends $X(z)$ if and only if $v = z$. Then, necessarily, the representative set $\mathcal{S}^*$ for $\mathcal{S}$ must contain $S(z)$ for every $z \in Z$, by letting $X = X(z)$ in the statement of the lemma. Furthermore, we do not need to provide the set $X(z)$ ahead of time, since the (possibly non-constructive) *existence* of such a set $X(z)$ is sufficient to force $S(z) \in \mathcal{S}^*$. Hence, the set $V^* = \{v \in V : S(v) \in \mathcal{S}^*\}$ must contain Z, among a polynomially bounded number of other vertices. The critical challenge, of course, is to construct the matroid M and sets $S(v)$ and $X(z)$ such that $S(z)$ indeed extends $X(z)$, while $S(v)$ fails to extend $X(z)$ for every $v \neq z$.

We illustrate the application to reducing terminal cut systems. Let $G = (V, E)$ be an undirected graph (the directed construction is similar), with $S, T \subseteq V$, and define a set of vertices Z where $z \in Z$ if and only if there are sets $A \subseteq S, B \subseteq T$ such that every minimum (A, B)-vertex cut contains z. We wish to learn Z. Let a *sink-only copy* of a vertex $v \in V$ be a copy v' of v with all edges oriented towards v'. Then the following follows from Lemma 2 and the definition of closest sets.

Lemma 3 *Let $A, B \subseteq V$, and let X be a minimum (A, B)-vertex cut. Then a vertex $v \in V$ is a member of every minimum (A, B)-vertex cut if and only if $\{v'\}$ extends X in both the gammoid (G, A, V) and the gammoid (G, B, V).*

Via a minor modification, we can replace the former gammoid by the gammoid (G, S, V) and the latter by (G, T, V) (for appropriate adjustments to the set X); we can then compute a set V^* of $O(|S| \cdot |T| \cdot k)$ vertices (where k is the size of an (S, T)-min cut) which contains Z. From this, we may compute the sought-after smaller graph G', by iteratively bypassing a single vertex $v \in V \setminus (S \cup T \cup V^*)$ and recomputing V^*, until $V^* \cup S \cup T = V$; observe that bypassing v does not change the size of any (A, B)-min cut. Theorem 1 follows by considering a modification of the graph G, and Theorem 2 follows by a generalization of the above, pushing into s different directions.

Further Applications

A polynomial kernel for MULTIWAY CUT (in the variants with only s terminals or with deletable terminals) essentially follows from the above, but the further kernelization applications in Theorem 3 require a few more steps. However, they follow a common pattern: First, we find an approximate solution X of size poly(k) to "bootstrap" the process; second, we use X to transform the problem into a more manageable form (e.g., for ALMOST 2-SAT, this manageable form is DIGRAPH PAIR CUT); and lastly, we use the above methods to kernelize the resulting problem. This pattern covers the problems listed in Theorem 3.

Finally, the above results have some implications beyond kernelization. In particular, the existence of the smaller graph G' computed for terminal cut systems, and correspondingly an implementation of a gammoid as a graph with poly$(|S| + |T|)$ vertices, was an open problem, solved in [3].

Cross-References

- Kernelization, Exponential Lower Bounds
- Kernelization, Polynomial Lower Bounds
- Matroids in Parameterized Complexity and Exact Algorithms

Recommended Reading

1. Fomin FV, Lokshtanov D, Saurabh S (2014) Efficient computation of representative sets with applications in parameterized and exact algorithms. In: SODA, Portland, pp 142–151
2. Kratsch S, Wahlström M (2012) Compression via matroids: a randomized polynomial kernel for odd cycle transversal. In: SODA, Kyoto, pp 94–103
3. Kratsch S, Wahlström M (2012) Representative sets and irrelevant vertices: new tools for kernelization. In: FOCS, New Brunswick, pp 450–459

4. Lovász L (1977) Flats in matroids and geometric graphs. In: Proceedings of the sixth British combinatorial conference, combinatorial surveys, Egham, pp 45–86
5. Marx D (2009) A parameterized view on matroid optimization problems. Theor Comput Sci 410(44):4471–4479
6. Oxley J (2006) Matroid theory. Oxford graduate texts in mathematics. Oxford University Press, Oxford
7. Perfect H (1968) Applications of Menger's graph theorem. J Math Anal Appl 22:96–111
8. Schrijver A (2003) Combinatorial optimization: polyhedra and efficiency. Algorithms and combinatorics. Springer, Berlin/New York

Kernelization, Max-Cut Above Tight Bounds

Mark Jones
Department of Computer Science, Royal Holloway, University of London, Egham, UK

Keywords

Kernel; Lambda extendible; Max Cut; Parameterization above tight bound

Years and Authors of Summarized Original Work

2012; Crowston, Jones, Mnich

Problem Definition

In the problem MAX CUT, we are given a graph G with n vertices and m edges, and asked to find a bipartite subgraph of G with the maximum number of edges.

In 1973, Edwards [5] proved that if G is connected, then G contains a bipartite subgraph with at least $\frac{m}{2} + \frac{n-1}{4}$ edges, proving a conjecture of Erdős. This lower bound on the size of a bipartite subgraph is known as the *Edwards-Erdős bound*. The bound is tight – for example, it is an upper bound when G is a clique with odd number of vertices. Thus, it is natural to consider parameterized MAX CUT above this bound, as follows (AEE stands for *Above Edwards-Erdős*).

MAX CUT AEE

Instance: A connected graph G with n vertices and m edges, and a nonnegative integer k.
Parameter: k.
Question: Does G have a bipartite subgraph with at least $\frac{m}{2} + \frac{n-1}{4} + k$ edges?

Mahajan and Raman [6], in their first paper on above-guarantee parameterizations, asked whether this problem is fixed-parameter tractable. As such, the problem was one of the first open problems in above-guarantee parameterizations.

λ-Extendibility and the Poljak-Turzík Bound

In 1982, Poljak and Turzík [8] investigated extending the Edwards-Erdős bound to cases when the desired subgraph is something other than bipartite. To this end, they introduced the notion of λ-extendibility, which generalizes the notion of "bipartiteness." We will define the slightly stronger notion of *strong* λ-extendibility, introduced in [7], as later results use this stronger notion.

Recall that a *block* of a graph G is a maximal 2-connected subgraph of G. The blocks of a graph form a partition of its edges, and a vertex that appears in two or more blocks is a cut-vertex of the graph.

Definition 1 For a family of graphs Π and $0 \leq \lambda \leq 1$, we say Π is *s*trongly λ-extendible if the following conditions are satisfied:

1. If G is connected and $|G| = 1$ or 2, then $G \in \Pi$.
2. G is in Π if and only if each of its blocks is in Π.
3. For any real-valued positive weight function w on the edges of G, if $X \subseteq V(G)$ is such that $G[X]$ is connected and $G[X], G - X \in \Pi$, then G has a subgraph $H \in \Pi$ that uses all the edges of $G[X]$, all the edges of $G - X$, and at least a fraction λ (by weight) of the edges between X and $V(G) \setminus X$.

The definition of λ-extendibility given in [8] is the same as the above, except that the third condition is only required when $|X| = 2$. Clearly strong λ-extendibility implies λ-extendibility; it is an open question whether the converse holds.

The property of being bipartite is strongly λ-extendible for $\lambda = 1/2$. Other strongly λ-extendible properties include being acyclic for directed graphs ($\lambda = 1/2$) and being r-colorable ($\lambda = 1/r$).

Poljak and Turzík [8] extended Edwards' result by showing that for any connected graph G with n vertices and m edges, and any λ-extendible property Π, G contains a subgraph in Π with at least $\lambda m + \frac{1-\lambda}{2}(n-1)$ edges.

Thus, for any λ-extendible property Π, we can consider the following variation of MAX CUT AEE, for any λ-extendible Π (APT stands for *Above Poljak-Turzík*).

Π-SUBGRAPH APT

Instance: A connected graph G with n vertices and m edges, and a nonnegative integer k.
Parameter: k.
Question: Does G have a subgraph in Π with at least $\lambda m + \frac{1-\lambda}{2}(n-1) + k$ edges?

Key Results

We sketch a proof of the polynomial kernel result for MAX CUT AEE, first shown in [2] (although the method described here is slightly different to that in [2]).

For a connected graph G with n vertices and m edges, let $\beta(G)$ denote the maximum number of edges of a bipartite subgraph of G, let $\gamma(G) = \frac{m}{2} + \frac{n-1}{4}$, and let $\varepsilon(G) = \beta(G) - \gamma(G)$. Thus, for an instance (G, k), our aim is to determine whether $\varepsilon(G) \geq k$.

Now consider a connected graph G with a set X of three vertices such that $G' = G - X$ is connected and $G[X] = P_3$, the path with two edges. Note that $G[X]$ is bipartite. Let H' be a subgraph of G' with $\beta(G')$ edges. As bipartiteness is a $1/2$-extendible property, we can create a bipartite subgraph H of G using the edges of H', the edges of $G[X]$, and at least half of the edges between X and $G - X$. It follows that $\beta(G) \geq \beta(G') + \frac{|E(X,V(G)\setminus X)|}{2} + 2$. As $\gamma(G) = \gamma(G') + \frac{|E(X,V(G)\setminus X)|+2}{2} + \frac{3}{4}$, we have that $\varepsilon(G) = \beta(G) - \gamma(G) \geq \beta(G') - \gamma(G') + 2 - \frac{2}{2} - \frac{3}{4} = \varepsilon(G') + \frac{1}{4}$.

Consider a reduction rule in which, if there exists a set X as described above, we delete X from the graph. If we were able to apply such a reduction rule $4k$ times on a graph G, we would end up with a reduced graph G' such that G' is connected and $\varepsilon(G) \geq \varepsilon(G') + \frac{4k}{4} \geq 0 + k$, and therefore we would know that (G, k) is a YES-instance. Of course there may be many graphs for which such a set X cannot be found. However, we can adapt this idea as follows. Given a connected graph G, we recursively calculate a set of vertices $S(G)$ and a rational number $t(G)$ as follows:

- If G is a clique or G is empty, then set $S(G) = \emptyset$ and $t(G) = 0$.
- If G contains a set X such that $|X| = 3$, $G' = G - X$ is connected and $G[X] = P_3$, then set $S(G) = S(G') \cup X$ and set $t(G) = t(G') + \frac{1}{4}$.
- If G contains a cut-vertex v, then there exist non-empty sets of vertices X, Y such that $X \cap Y = \{v\}$, $G[X]$ and $G[Y]$ are connected, and all edges of G are in $G[X]$ or $G[Y]$. Then set $S(G) = S(G[X]) \cup S(G[Y])$ and set $t(G) = t(G[X]) + t(G[Y])$.

It can be shown that for a connected graph G, one of these cases will always hold, and so $S(G)$ and $t(G)$ are well defined. In the first case, we have that $\varepsilon(G) \geq 0$ by the Edwards-Erdős bound. In the second case, we have already shown that $\varepsilon(G) \geq \varepsilon(G') + \frac{1}{4}$. In the third case, we have that $\varepsilon(G) = \varepsilon(G[X]) + \varepsilon(G[Y])$ (note that the union of a bipartite subgraph of $G[X]$ and a bipartite subgraph of $G[Y]$ is a bipartite subgraph of G). It follows that $\varepsilon(G) \geq t(G)$. Note also that $|S(G)| \leq 12t(G)$. If we remove $S(G)$ from G, the resulting graph can be built by joining disjoint graphs at a single vertex, using only cliques as the initial graphs. Thus, $G - S(G)$ has the property

that each of its blocks is a clique. We call such a graph a *forest of cliques*.

We therefore get the following lemma.

Lemma 1 ([2]) *Given a connected graph G with n vertices and m edges, and an integer k, we can in polynomial time either decide that (G, k) is a* YES*-instance of* MAX CUT AEE*, or find a set S of at most $12k$ vertices such that $G - S$ is a forest of cliques.*

By guessing a partition of S and then using a dynamic programming algorithm based on the structure of $G - S$, we get a fixed-parameter algorithm.

Theorem 1 ([2]) MAX CUT AEE *can be solved in time $2^{O(k)} \cdot n^4$.*

Using the structure of $G - S$ and the fact that $|S| \leq 12k$, it is possible (using reduction rules) to show first that the number of blocks in $G - S$ must be bounded for any NO-instance, and then that the size of each block must be bounded (see [2]).

Thus, we get a polynomial kernel for MAX CUT AEE.

Theorem 2 ([2]) MAX CUT AEE *admits a kernel with $O(k^5)$ vertices.*

Crowston et al. [3] were later able to improve this to a kernel with $O(k^3)$ vertices.

Extensions to Π-SUBGRAPH APT

A similar approach can be used to show polynomial kernels for Π-SUBGRAPH APT, for other $1/2$-extendible properties. In particular, the property of being an acyclic directed graph is $1/2$-extendible, and therefore every directed graph with n vertices and m arcs has an acyclic subgraph with at least $\frac{m}{2} + \frac{n-1}{4}$ arcs. The problem of deciding whether there exists an acyclic subgraph with at least $\frac{m}{2} + \frac{n-1}{4} + k$ arcs is fixed-parameter tractable, and has a $O(k^2)$-vertex kernel [1].

The notion of a bipartite graph can be generalized in the following way. Consider a graph G with edges labeled either $+$ or $-$. Then we say G is *balanced* if there exists a partition V_1, V_2 of the vertices of G, such that all edges between V_1 and V_2 are labeled $-$ and all other edges are labeled $+$. (Note that if all edges of a graph are labeled $-$, then it is balanced if and only if it is bipartite.) The property of being a balanced graph is $1/2$-extendible, just as the property of being bipartite is. Therefore a graph with n vertices and m vertices, and all edges labeled $+$ or $-$, will have a balanced subgraph with at least $\frac{m}{2} + \frac{n-1}{4}$ edges. The problem of deciding whether there exists a balanced subgraph with at least $\frac{m}{2} + \frac{n-1}{4} + k$ edges is fixed-parameter tractable and has a $O(k^3)$-vertex kernel [3].

Mnich et al. [7] showed that Lemma 1 applies not just for MAX CUT AEE, but for Π-SUBGRAPH APT for any Π which is strongly λ-extendible for some λ (with the bound $12k$ replaced with $\frac{6k}{1-\lambda}$). Thus, Π-SUBGRAPH APT is fixed-parameter tractable as long as it is fixed-parameter tractable on graphs which are close to being a forest of cliques. Using this observation, Mnich et al. showed fixed-parameter tractability for a number of versions of Π-SUBGRAPH APT, including when Π is the family of acyclic directed graphs and when Π is the set of r-colorable graphs.

Crowston et al. [4] proved the existence of polynomial kernels for a wide range of strongly λ-extendible properties:

Theorem 3 ([4]) *Let $0 < \lambda < 1$, and let Π be a strongly λ-extendible property of (possibly oriented and/or labeled) graphs. Then Π-*SUBGRAPH APT *has a kernel on $O(k^2)$ vertices if Condition 1 or 2 holds, and a kernel on $O(k^3)$ vertices if only Condition 3 holds:*

1. *$\lambda \neq \frac{1}{2}$.*
2. *All orientations and labels (if applicable) of the graph K_3 belong to Π.*
3. *Π is a hereditary property of simple or oriented graphs.*

Open Problems

The Poljak-Turzík's bound extends to edge-weighted graphs. The weighted version is as follows: for a graph G with nonnegative real weights on the edges, and a λ-extendible family

of graphs Π, there exists a subgraph H of G such that $H \in \Pi$ and H has total weight $\lambda \cdot w(G) + \frac{1-\lambda}{2} \cdot MST(G)$, where $w(G)$ is the total weight of G and $MST(G)$ is the minimum weight of a spanning tree in G.

Thus, we can consider the weighted versions of MAX CUT AEE and Π-SUBGRAPH APT. It is known that a weighted equivalent of Lemma 1 holds (in which all edges in a block of $G-S$ have the same weight), and as a result, the integer-weighted version of MAX CUT AEE can be shown to be fixed-parameter tractable. However, nothing is known about kernelization results for these problems. In particular, it remains an open question whether the integer-weighted version of MAX CUT AEE has a polynomial kernel.

Cross-References

- Kernelization, Constraint Satisfaction Problems Parameterized above Average
- Kernelization, MaxLin Above Average

Recommended Reading

1. Crowston R, Gutin G, Jones M (2012) Directed acyclic subgraph problem parameterized above the Poljak-Turzík bound. In: FSTTCS 2012, Hyderabad. LIPICS, vol 18, pp 400–411
2. Crowston R, Jones M, Mnich M (2012) Max-Cut parameterized above the Edwards-Erdős bound. In: ICALP 2012, Warwick. Lecture notes in computer science, vol 7391, pp 242–253
3. Crowston R, Gutin G, Jones M, Muciaccia G (2013) Maximum balanced subgraph problem parameterized above lower bounds. Theor Comput Sci 513:53–64
4. Crowston R, Jones M, Muciaccia G, Philip G, Rai A, Saurabh S (2013) Polynomial kernels for λ-extendible properties parameterized above the Poljak-Turzík bound. In: FSTTCS 2013, Guwahati. LIPICS, vol 24, pp 43–54
5. Edwards CS (1973) Some extremal properties of bipartite subgraphs. Can J Math 25:475–485
6. Mahajan M, Raman V (1999) Parameterizing above guaranteed values: MaxSat and MaxCut. J Algorithms 31(2):335–354
7. Mnich M, Philip G, Saurabh S, Suchý O (2014) Beyond Max-Cut: λ-extendible properties parameterized above the Poljak-Turzík bound. J Comput Syst Sci 80(7):1384–1403
8. Poljak S, Turzík D (1982) A polynomial algorithm for constructing a large bipartite subgraph, with an application to a satisfiability problem. Can J Math 34(4):519–524

Kernelization, MaxLin Above Average

Anders Yeo
Engineering Systems and Design, Singapore University of Technology and Design, Singapore, Singapore
Department of Mathematics, University of Johannesburg, Auckland Park, South Africa

Keywords

Fixed-parameter tractability above lower bounds; Kernelization; Linear equations; M-sum-free sets

Years and Authors of Summarized Original Work

2010; Crowston, Gutin, Jones, Kim, Ruzsa
2011; Gutin, Kim, Szeider, Yeo
2014; Crowston, Fellows, Gutin, Jones, Kim, Rosamond, Ruzsa, Thomassé, Yeo

Problem Definition

The problem MAXLIN2 can be stated as follows. We are given a system of m equations in variables $x_1, \ldots, x_n$ where each equation is $\prod_{i \in I_j} x_i = b_j$, for some $I_j \subseteq \{1, 2, \ldots, n\}$ and $x_i, b_j \in \{-1, 1\}$ and $j = 1, \ldots, m$. Each equation is assigned a positive integral weight w_j. We are required to find an assignment of values to the variables in order to maximize the total weight of the satisfied equations. MAXLIN2 is a well-studied problem, which according to Håstad [8] "is as basic as satisfiability."

Note that one can think of MAXLIN2 as containing equations, $\sum_{i \in I_j} y_i = a_j$ over $\mathbb{F}_2$. This is equivalent to the previous definition by letting $y_i = 0$ if and only if $x_i = 1$ and letting $y_i = 1$ if and only if $x_i = -1$ (and $a_j = 1$ if and only if $b_j = -1$ and $a_j = 0$ if and only if $b_j = 1$). We will however use the original definition as this was the formulation used in [1].

Let W be the sum of the weights of all equations in an instance, S, of MAXLIN2 and let $\text{sat}(S)$ be the maximum total weight of equations that can be satisfied simultaneously. To see that $W/2$ is a tight lower bound on $\text{sat}(S)$, choose assignments to the variables independently and uniformly at random. Then $W/2$ is the expected weight of satisfied equations (as the probability of each equation being satisfied is $1/2$) and thus $W/2$ is a lower bound. It is not difficult to see that this bound is tight. For example, consider a system consisting of pairs of equations of the form $\prod_{i \in I} x_i = -1$, $\prod_{i \in I} x_i = 1$ of the same weight, for some nonempty sets $I \subseteq \{1, 2, \ldots, n\}$.

As MAXLIN2 is an NP-hard problem, we look for parameterized algorithms. We will give the basic definitions of fixed-parameter tractability (FPT) here and refer the reader to [4, 5] for more information. A *parameterized problem* is a subset $L \subseteq \Sigma^* \times \mathbb{N}$ over a finite alphabet Σ. L is *fixed-parameter tractable* (FPT, for short) if membership of an instance (x, k) in $\Sigma^* \times \mathbb{N}$ can be decided in time $f(k)|x|^{O(1)}$, where f is a function of the parameter k only.

If we set the parameter, k, of an instance, S, of MAXLIN2 to $\text{sat}(S)$, then it is easy to see that there exists an $O(f(k)|S|^c)$ algorithm, due to the fact that $k = \text{sat}(S) \geq W/2 \geq |S|/2$. Therefore, this parameter is not of interest (it is never small in practice), and a better parameter would be k, where we want to decide if $\text{sat}(S) \geq W/2 + k$. Parameterizing above tight lower bounds in this way was first introduced in 1997 in [11]. This leads us to define the following problem, where AA stands for *Above Average*.

MAXLIN2-AA

Instance: A system S of equations $\prod_{i \in I_j} x_i = b_j$, where $x_i, b_j \in \{-1, 1\}$, $j = 1, \ldots, m$ and where each equation is assigned a positive integral weight w_j and a nonnegative integer k.

Question: $\text{sat}(S) \geq W/2 + k$?

The above problem has also been widely studied when the number of variables in each equation is bounded by some constant, say r, which leads to the following problem.

MAX-r-LIN2-AA

Instance: A system S of equations $\prod_{i \in I_j} x_i = b_j$, where $x_i, b_j \in \{-1, 1\}$, $|I_j| \leq r$, $j = 1, \ldots, m$; equation j is assigned a positive integral weight w_j and a nonnegative integer k.

Question: $\text{sat}(S) \geq W/2 + k$?

Given a parameterized problem, Π, a *kernel* of Π is a polynomial-time algorithm that maps an instance (I, k) of Π to another instance, (I', k'), of Π such that (i) $(I, k) \in \Pi$ if and only if $(I', k') \in \Pi'$, (ii) $k' \leq f(k)$, and (iii) $|I'| \leq g(k)$ for some functions f and g. The function $g(k)$ is called the *size* of the kernel. It is well known that a problem is FPT if and only if it has a kernel.

A kernel is called a polynomial kernel if both $f(k)$ and $g(k)$ are polynomials in k. A great deal of research has been devoted to finding small-sized kernels and in particular to decide if a problem has a polynomial kernel.

We will show that both the problems stated above are FPT and in fact contain kernels with a polynomial number of variables. The number of equations may be non-polynomial, so these kernels are not real polynomial kernels. The above problems were investigated in a number of papers; see [1–3, 6].

Key Results

We will below outline the key results for both MaxLin2-AA and Max-r-Lin2-AA. See [1] for all the details not given here.

MaxLin2-AA

Recall that MaxLin2-AA considers a system S of equations $\prod_{i\in I_j} x_i = b_j$, where $x_i, b_j \in \{-1, 1\}$, $j = 1, \ldots, m$ and where each equation is assigned a positive integral weight w_j. Let $\mathcal{F}$ denote the m different sets I_j in the equations of S and let $b_{I_j} = b_j$ and $w_{I_j} = w_j$ for each $j = 1, 2, \ldots, m$.

Let $\varepsilon(x) = \sum_{I\in\mathcal{F}} w_I b_I \prod_{i\in I} x_i$ and note that $\varepsilon(x)$ is the difference between the total weight of satisfied and falsified equations. Crowston et al. [3] call $\varepsilon(x)$ the *excess* and the maximum possible value of $\varepsilon(x)$ the *maximum excess*.

Remark 1 Observe that the answer to MaxLin2-AA and Max-r-Lin2-AA is YES if and only if the maximum excess is at least $2k$.

Let A be the matrix over $\mathbb{F}_2$ corresponding to the set of equations in S, such that $a_{ji} = 1$ if $i \in I_j$ and 0, otherwise. Consider the following two reduction rules, where Rule 1 was introduced in [9] and Rule 2 in [6].

Reduction Rule 1 ([9]) *If we have, for a subset I of $\{1, 2, \ldots, n\}$, an equation $\prod_{i\in I} x_i = b'_I$ with weight w'_I, and an equation $\prod_{i\in I} x_i = b''_I$ with weight w''_I, then we replace this pair by one of these equations with weight $w'_I + w''_I$ if $b'_I = b''_I$ and, otherwise, by the equation whose weight is bigger, modifying its new weight to be the difference of the two old ones. If the resulting weight is 0, we delete the equation from the system.*

Reduction Rule 2 ([6]) *Let $t = \mathrm{rank}A$ and suppose columns $a^{i_1}, \ldots, a^{i_t}$ of A are linearly independent. Then delete all variables not in $\{x_{i_1}, \ldots, x_{i_t}\}$ from the equations of S.*

Lemma 1 ([6]) *Let S' be obtained from S by Rule 1 or 2. Then the maximum excess of S' is equal to the maximum excess of S. Moreover, S' can be obtained from S in time polynomial in n and m.*

If we cannot change a weighted system S using Rules 1 and 2, we call it *irreducible*. Let S be an irreducible system of MaxLin2-AA. Consider the following algorithm introduced in [3]. We assume that, in the beginning, no equation or variable in S is marked.

ALGORITHM $\mathcal{H}$
While the system S is nonempty, do the following:

1. Choose an equation $\prod_{i\in I} x_i = b$ and mark a variable x_l such that $l \in I$.
2. Mark this equation and delete it from the system.
3. Replace every equation $\prod_{i\in I'} x_i = b'$ in the system containing x_l by $\prod_{i\in I\Delta I'} x_i = bb'$, where $I\Delta I'$ is the symmetric difference of I and I' (the weight of the equation is unchanged).
4. Apply Reduction Rule 1 to the system.

The *maximum $\mathcal{H}$-excess* of S is the maximum possible total weight of equations marked by $\mathcal{H}$ for S taken over all possible choices in Step 1 of $\mathcal{H}$. The following lemma indicates the potential power of $\mathcal{H}$.

Lemma 2 ([3]) *Let S be an irreducible system. Then the maximum excess of S equals its maximum $\mathcal{H}$-excess.*

Theorem 1 ([1]) *There exists an $O(n^{2k}(nm)^{O(1)})$-time algorithm for* MaxLin2-AA$[k]$ *that returns an assignment of excess of at least $2k$ if one exists, and returns* NO *otherwise.*

In order to prove the above, the authors pick n equations $e_1, \ldots, e_n$ such that their rows in A are linearly independent. An assignment of excess at least $2k$ must either satisfy one of these equations or falsify them all. If they are all falsified, then the value of all variables is completely determined. Thus, by Lemma 2, algorithm $\mathcal{H}$ can mark one of these equations, implying a search tree of depth at most $2k$ and width at most k. This implies the desired time bound.

Theorem 2 below is proved using M-sum-free sets, which are defined as follows (see [3]). Let K and M be sets of vectors in $\mathbb{F}_2^n$ such that $K \subseteq M$. We say K is *M-sum-free* if no sum of two or more distinct vectors in K is equal to a vector in M.

Theorem 2 ([1]) *Let S be an irreducible system of* MaxLin2-AA$[k]$ *and let $k \geq 1$. If $2k \leq m \leq \min\{2^{n/(2k-1)} - 1, 2^n - 2\}$, then the maximum excess of S is at least $2k$. Moreover, we can find an assignment with excess of at least $2k$ in time $O(m^{O(1)})$.*

Using the above, we can solve the problem when $2k \leq m \leq 2^{n/(2k-1)} - 2$ and when $m \geq n^{2k} - 1$ (using Theorem 1). The case when $m < 2k$ immediately gives a kernel and the remaining case when $2^{n/(2k-1)} - 2 \leq m \leq n^{2k} - 2$ can be shown to imply that $n \in O(k^2 \log k)$, thereby giving us the main theorem and corollary of this section.

Theorem 3 ([1]) *The problem* MaxLin2-AA$[k]$ *has a kernel with at most $O(k^2 \log k)$ variables.*

Corollary 1 ([1]) *The problem* MaxLin2-AA$[k]$ *can be solved in time $2^{O(k \log k)}(nm)^{O(1)}$.*

Max-r-Lin2-AA

In [6] it was proved that the problem Max-r-Lin2-AA admits a kernel with at most $O(k^2)$ variables and equations (where r is treated as a constant). The bound on the number of variables can be improved and it was done by Crowston et al. [3] and Kim and Williams [10]. The best known improvement is by Crowston et al. [1].

Theorem 4 ([1]) *The problem* Max-r-Lin2-AA *admits a kernel with at most $(2k - 1)r$ variables.*

Both Theorem 4 and a slightly weaker analogous result of the results in [10] imply the following:

Lemma 3 ([1,10]) *There is an algorithm of runtime $2^{O(k)} + m^{O(1)}$ for* Max-r-Lin2-AA.

Kim and Williams [10] proved that the last result is best possible, in a sense, if the exponential time hypothesis holds.

Theorem 5 ([10]) *If* Max-3-Lin2-AA *can be solved in $O(2^{\epsilon k}2^{\epsilon m})$ time for every $\epsilon > 0$, then 3-SAT can be solved in $O(2^{\delta n})$ time for every $\delta > 0$, where n is the number of variables.*

Open Problems

The kernel for MaxLin2-AA contains at most $O(k^2 \log k)$ variables, but may contain an exponential number of equations. It would be of interest to decide if MaxLin2-AA admits a kernel that has at most a polynomial number of variables and equations.

Cross-References

- Kernelization, Constraint Satisfaction Problems Parameterized above Average
- Kernelization, Permutation CSPs Parameterized above Average

Recommended Reading

1. Crowston R, Fellows M, Gutin G, Jones M, Kim EJ, Rosamond F, Ruzsa IZ, Thomassé S, Yeo A (2014) Satisfying more than half of a system of linear equations over GF(2): a multivariate approach. J Comput Syst Sci 80(4):687–696
2. Crowston R, Gutin G, Jones M (2010) Note on Max Lin-2 above average. Inform Proc Lett 110:451–454
3. Crowston R, Gutin G, Jones M, Kim EJ, Ruzsa I (2010) Systems of linear equations over $\mathbb{F}_2$ and problems parameterized above average. In: SWAT 2010, Bergen. Lecture notes in computer science, vol 6139, pp 164–175
4. Downey RG, Fellows MR (2013) Fundamentals of parameterized complexity. Springer, London/Heidelberg/New York
5. Flum J, Grohe M (2006) Parameterized complexity theory. Springer, Berlin
6. Gutin G, Kim EJ, Szeider S, Yeo A (2011) A probabilistic approach to problems parameterized above or below tight bounds. J Comput Syst Sci 77:422–429
7. Gutin G, Yeo A (2012) Constraint satisfaction problems parameterized above or below tight bounds: a survey. Lect Notes Comput Sci 7370:257–286

8. Håstad J (2001) Some optimal inapproximability results. J ACM 48:798–859
9. Håstad J, Venkatesh S (2004) On the advantage over a random assignment. Random Struct Algorithms 25(2):117–149
10. Kim EJ, Williams R (2012) Improved parameterized algorithms for above average constraint satisfaction. In: IPEC 2011, Saarbrücken. Lecture notes in computer science, vol 7112, pp 118–131
11. Mahajan M, Raman V (1999) Parameterizing above guaranteed values: MaxSat and MaxCut. J Algorithms 31(2):335–354. Preliminary version in Electr. Colloq. Comput. Complex. (ECCC), TR-97-033, 1997

Kernelization, Partially Polynomial Kernels

Christian Komusiewicz
Institute of Software Engineering and Theoretical Computer Science, Technical University of Berlin, Berlin, Germany

Keywords

Data reduction; Fixed-parameter algorithms; Kernelization; NP-hard problems

Years and Authors of Summarized Original Work

2011; Betzler, Guo, Komusiewicz, Niedermeier
2013; Basavaraju, Francis, Ramanujan, Saurabh
2014; Betzler, Bredereck, Niedermeier

Problem Definition

In parameterized complexity, each instance (I, k) of a problem comes with an additional parameter k which describes structural properties of the instance, for example, the maximum degree of an input graph. A problem is called *fixed-parameter tractable* if it can be solved in $f(k)\cdot\text{poly}(n)$ time, that is, the super-polynomial part of the running time depends only on k. Consequently, instances of the problem can be solved efficiently if k is small.

One way to show fixed-parameter tractability of a problem is the design of a polynomial-time data reduction algorithm that reduces any input instance (I, k) to one whose size is bounded in k. This idea is captured by the notion of kernelization.

Definition 1 Let (I, k) be an instance of a parameterized problem P, where $I \in \Sigma^*$ denotes the input instance and $k \in \mathbb{N}$ is a parameter. Problem P admits a *problem kernel* if there is a polynomial-time algorithm, called *problem kernelization*, that computes an instance (I', k') of the same problem P such that:

- (I, k) is a yes-instance if and only if (I', k') is a yes-instance, and
- $|I'| + k' \leq g(k)$

for a function g of k only.

Kernelization gives a performance guarantee for the effectiveness of data reduction: instances (I, k) with $|I| > g(k)$ are provably reduced to smaller instances. Thus, one aim in the design of kernelization algorithms is to make the function g as small as possible. In particular, one wants to obtain kernelizations where g is a polynomial function. These algorithms are called *polynomial problem kernelizations*.

For many parameterized problems, however, the existence of such a polynomial problem kernelization is considered to be unlikely (under a standard complexity-theoretic assumption) [4]. Consequently, alternative models of parameterized data reduction, for example Turing kernelization, have been proposed.

The concept of *partial kernelization* offers a further approach to obtain provably useful data reduction algorithms. Partial kernelizations do not aim for a decrease of the instance *size* but for a decrease of some *part* or *dimension* of the instance. For example, if the problem input is a binary matrix with m rows and n columns, the instance size is $\Theta(n \cdot m)$. A partial kernelization can now aim for reducing one dimension of the input, for example the number of rows n. Of course,

such a reduction is worthwhile only if we can algorithmically exploit the fact that the number of rows n is small. Hence, the aim is to reduce a dimension of the problem for which there are fixed-parameter algorithms. The dimension can thus be viewed as a secondary parameter.

Altogether, this idea is formalized as follows.

Definition 2 Let (I, k) be an instance of a parameterized problem P, where $I \in \Sigma^*$ denotes the input instance and k is a parameter. Let $d : \Sigma^* \to \mathbb{N}$ be a computable function such that P is fixed-parameter tractable with respect to $d(I)$. Problem P admits a *partial problem kernel* if there is a polynomial-time algorithm, called *partial problem kernelization*, that computes an instance (I', k') of the same problem such that:

- (I, k) is a yes-instance if and only if (I', k') is a yes-instance, and
- $d(I') + k' \leq g(k)$

for a computable function g.

Any parameterized problem P which has a partial kernel for some appropriate dimension d is fixed-parameter tractable with respect to k: First, one may reduce the original input instance (I, k) to the partial kernel (I', k'). In this partial kernel, we have $d(I') \leq g(k)$ and, since P can be solved in $f(d(I')) \cdot \text{poly}(n)$ time, it can thus be solved in $f(g(k)) \cdot \text{poly}(n)$ time.

Using partial problem kernelization instead of classic problem kernelization can be motivated by the following two arguments.

First, the function d in the partial problem kernelization gives us a different goal in the design of efficient data reduction rules. For instance, if the main parameter determining the hardness of a graph problem is the maximum degree, then an algorithm that produces instances whose maximum degree is $O(k)$ but whose size is unbounded might be more useful than an algorithm that produces instances whose size is $O(k^4)$ but the maximum degree is $\Omega(k^2)$.

Second, if the problem does not admit a polynomial-size problem kernel, then it might still admit a *partially polynomial kernel*, that is, a partial kernel in which $d(I') + k' \leq \text{poly}(k)$.

We now give two examples for applications of partial kernelizations.

Key Results

The partial kernelization concept was initially developed to obtain data reduction algorithms for consensus problems, where one is given a collection of combinatorial objects and one is asked to find one object that represents this collection [2].

In the KEMENY SCORE problem, these objects are permutations of a set U and the task is to find a permutation that is close to these permutations with respect to what is called *Kendall's Tau distance*, here denoted by τ. The formal definition of the (unparameterized) problem is as follows.

Input: A multiset $\mathcal{P}$ of permutations of a ground set U and an integer ℓ.
Question: Is there a permutation P such that $\sum_{P' \in \mathcal{P}} \tau(P, P') \leq \ell$?

The parameter k under consideration is the average distance between the input partitions, that is,

$$k := \sum_{\{P, P'\} \subseteq \mathcal{P}} \tau(P, P') / \binom{|\mathcal{P}|}{2}.$$

Observe that, since τ can be computed efficiently, KEMENY SCORE is fixed-parameter tractable with respect to $|U|$: try all possible permutations of U and choose the best one. Hence, if U is small, then the problem is easy. Furthermore, the number of input permutations is not such a crucial feature since KEMENY SCORE is already NP-hard for a constant number of permutations; the partial kernelization thus aims for a reduction of $|U|$ and ignores the—less important—number of input permutations.

This reduction is obtained by removing elements in U that are, compared to the other elements, in roughly the same position in many input permutations. The idea is based on a

generalization of the following observation: If some element u is the first element of at least $3|\mathcal{P}|/4$ input permutations, then this element is the first element of an optimal partition. Any instance containing such an element u can thus be reduced to an equivalent with one less element.

By removing such elements, one obtains a sub-instance of the original instance in which every element contributes a value of $16/3$ to the average distance k between the input permutations. This leads to the following result.

Theorem 1 ([3]) KEMENY SCORE *admits a partial kernel with* $|U| \leq 16/3k$.

Further partial kernelizations for consensus problems have been obtained for CONSENSUS CLUSTERING [2, 7] and SWAP MEDIAN PARTITION [2], the partial kernelization for KEMENY SCORE has been experimentally evaluated [3].

Another application of partial kernelization has been proposed for covering problems such as SET COVER [1].

Input: A family $\mathcal{S}$ of subsets of a ground set U.
Question: Is there a subfamily $\mathcal{S}' \subseteq \mathcal{S}$ of size at most ℓ such that every element in U is contained in at least one set of $\mathcal{S}'$?

If $\ell \geq |U|$, then SET COVER has a trivial solution. Thus, a natural parameter is the amount that can be saved compared to this trivial solution, that is, $k := |U| - \ell$. A polynomial problem kernelization for SET COVER parameterized by k is deemed unlikely, again under standard complexity-theoretic assumptions. There is, however, a partially polynomial problem kernel. The dimension d is the universe size $|U|$. SET COVER is fixed-parameter tractable with respect to $|U|$ as it can be solved in $f(|U|) \cdot \text{poly}(n)$ time, for example, by dynamic programming.

The idea behind the partial kernelization is to greedily compute a subfamily $\mathcal{T} \subseteq \mathcal{S}$ of size k. Then, it is observed that either this subfamily has a structure that can be used to efficiently compute a solution of the problem, or $|U| \leq 2k^2 - 2$, or there are elements in U whose removal yields an equivalent instance. Altogether this leads to the following.

Theorem 2 ([1]) SET COVER *admits a partial problem kernel with* $|U| \leq 2k^2 - 2$.

Open Problems

The notion of partial kernelization is quite recent. Hence, the main aim for the near future is to identify further useful applications of the technique. We list some problem areas that contain natural candidates for such applications. Problems that are defined on set families, such as SET COVER, have two obvious dimensions: the number m of sets in the set family and the size n of the universe. Matrix problems also have two obvious dimensions: the number m of rows and the number n of columns. For graph problems, useful dimensions could be identified by examining the so-called parameter hierarchy [6, 8]. Here, the idea is to find dimensions whose value can be much smaller than the number of vertices in the graph. If the size $|I|$ of the instance cannot be reduced to be smaller than $\text{poly}(k)$, then this might be still possible for the smaller dimension $d(I)$. A further interesting research direction could be to study the relationship between partial kernelization and other relaxed notions of kernelization such as Turing kernelization.

For some problems, the existence of partially polynomial kernels has been proven, but it is still unknown whether polynomial kernels exist. One such example is MAXLIN2-AA [5].

Cross-References

- Kernelization, MaxLin Above Average
- Kernelization, Polynomial Lower Bounds
- Kernelization, Turing Kernels

Recommended Reading

1. Basavaraju M, Francis MC, Ramanujan MS, Saurabh S (2013) Partially polynomial kernels for set cover and test cover. In: FSTTCS '13, Guwahati. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, LIPIcs, vol 24, pp 67–78
2. Betzler N, Guo J, Komusiewicz C, Niedermeier R (2011) Average parameterization and partial kernelization for computing medians. J Comput Syst Sci 77(4):774–789

3. Betzler N, Bredereck R, Niedermeier R (2014) Theoretical and empirical evaluation of data reduction for exact Kemeny rank aggregation. Auton Agents Multi-Agent Syst 28(5):721–748
4. Bodlaender HL, Downey RG, Fellows MR, Hermelin D (2009) On problems without polynomial kernels. J Comput Syst Sci 75(8):423–434
5. Crowston R, Fellows M, Gutin G, Jones M, Kim EJ, Rosamond F, Ruzsa IZ, Thomassé S, Yeo A (2014) Satisfying more than half of a system of linear equations over GF(2): a multivariate approach. J Comput Syst Sci 80(4):687–696
6. Fellows MR, Jansen BMP, Rosamond FA (2013) Towards fully multivariate algorithmics: parameter ecology and the deconstruction of computational complexity. Eur J Comb 34(3): 541–566
7. Komusiewicz C (2011) Parameterized algorithmics for network analysis: clustering & querying. PhD thesis, Technische Universität Berlin, Berlin
8. Komusiewicz C, Niedermeier R (2012) New races in parameterized algorithmics. In: MFCS '12, Bratislava. Lecture notes in computer science, vol 7464. Springer, pp 19–30

Kernelization, Permutation CSPs Parameterized above Average

Gregory Gutin
Department of Computer Science, Royal Holloway, University of London , Egham, UK

Keywords

Betweenness; Linear ordering; Parameterization above tight bound

Years and Authors of Summarized Original Work

2012; Gutin, van Iersel, Mnich, Yeo

Problem Definition

Let r be an integer and let V be a set of n variables. An *ordering* α is a bijection from V to $\{1, 2, \ldots, n\}$; a *constraint* is an ordered r-tuple $(v_1, v_2, \ldots, v_r)$ of distinct variables of V; α *satisfies* $(v_1, v_2, \ldots, v_r)$ if $\alpha(v_1) < \alpha(v_2) < \cdots < \alpha(v_r)$. An instance of MAX-$r$-LIN-ORDERING consists of a multiset $\mathcal{C}$ of constraints, and the objective is to find an ordering that satisfies the maximum number of constraints. Note that MAX-2-LIN ORDERING is equivalent to the problem of finding a maximum weight acyclic subgraph in an integer-weighted directed graph. Since the FEEDBACK ARC SET problem is NP-hard, MAX-2-LIN ORDERING is NP-hard, and thus MAX-r-LIN-ORDERING is NP-hard for each $r \geq 2$.

Let α be an ordering chosen randomly and uniformly from all orderings and let $c \in \mathcal{C}$ be a constraint. Then the probability that α satisfies c is $1/r!$. Thus the expected number of constraints in $\mathcal{C}$ satisfied by α equals $|\mathcal{C}|/r!$. This is a lower bound on the maximum number of constraints satisfied by an ordering, and, in fact, it is a tight lower bound. This allows us to consider the following parameterized problem (AA stands for *Above Average*).

MAX-r-LIN-ORDERING-AA

Instance: A multiset $\mathcal{C}$ of constraints and a nonnegative integer k.
Parameter: k.
Question: Is there an ordering satisfying at least $|\mathcal{C}|/r! + k$ constraints?

$(1, 2, \ldots, r)$ is the identity permutation of the symmetric group $\mathcal{S}_r$. We can extend MAX-r-LIN-ORDERING by considering an arbitrary subset of $\mathcal{S}_r$ rather than just $\{(1, 2, \ldots, r)\}$. Instead of describing the extension for each arity $r \geq 2$, we will do it only for $r = 3$, which is our main interest, and leave the general case to the reader.

Let $\Pi \subseteq \mathcal{S}_3 = \{(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)\}$ be arbitrary. For an ordering $\alpha\colon V \to \{1, 2, \ldots, n\}$, a constraint $(v_1, v_2, v_3) \in \mathcal{C}$ is *Π-satisfied by α* if there is a permutation $\pi \in \Pi$ such that $\alpha(v_{\pi(1)}) < \alpha(v_{\pi(2)}) < \alpha(v_{\pi(3)})$. Given Π, the problem Π-CSP is the problem of deciding if there exists an ordering of V that Π-satisfies all the constraints. Every such problem is called a Permutation CSP of arity 3. We will consider the maximization version of these problems, denoted by MAX-Π-

Kernelization, Permutation CSPs Parameterized above Average, Table 1 Permutation CSPs of arity 3 (after symmetry considerations)

$\Pi \subseteq \mathcal{S}_3$	Name	Complexity
$\Pi_0 = \{(123)\}$	3-LIN-ORDERING	Polynomial
$\Pi_1 = \{(123), (132)\}$		Polynomial
$\Pi_2 = \{(123), (213), (231)\}$		Polynomial
$\Pi_3 = \{(132), (231), (312), (321)\}$		Polynomial
$\Pi_4 = \{(123), (231)\}$		NP-comp.
$\Pi_5 = \{(123), (321)\}$	BETWEENNESS	NP-comp.
$\Pi_6 = \{(123), (132), (231)\}$		NP-comp.
$\Pi_7 = \{(123), (231), (312)\}$	CIRCULAR ORDERING	NP-comp.
$\Pi_8 = \mathcal{S}_3 \setminus \{(123), (231)\}$		NP-comp.
$\Pi_9 = \mathcal{S}_3 \setminus \{(123), (321)\}$	NON-BETWEENNESS	NP-comp.
$\Pi_{10} = \mathcal{S}_3 \setminus \{(123)\}$		NP-comp.

CSP, parameterized above the average number of constraints satisfied by a random ordering of V (which can be shown to be a tight bound).

It is easy to see that there is only one distinct Π-CSP of arity 2. Guttmann and Maucher [5] showed that there are in fact only 13 distinct Π-CSPs of arity 3 up to symmetry, of which 11 are nontrivial. They are listed in Table 1 together with their complexity. Some of the problems listed in the table are well known and have special names. For example, the problem for $\Pi = \{(123), (321)\}$ is called the BETWEENNESS problem.

Gutin et al. [4] proved that all 11 nontrivial MAX-Π-CSP problems are NP-hard (even though four of the Π-CSP are polynomial).

Now observe that given a variable set V and a constraint multiset $\mathcal{C}$ over V, for a random ordering α of V, the probability of a constraint in $\mathcal{C}$ being Π-satisfied by α equals $\frac{|\Pi|}{6}$. Hence, the expected number of satisfied constraints from $\mathcal{C}$ is $\frac{|\Pi|}{6}|\mathcal{C}|$, and thus there is an ordering α of V satisfying at least $\frac{|\Pi|}{6}|\mathcal{C}|$ constraints (and this bound is tight). A derandomization argument leads to $\frac{|\Pi|}{6}$-approximation algorithms for the problems MAX-Π-CSP [1]. No better constant factor approximation is possible assuming the Unique Games Conjecture [1].

We will study the parameterization of MAX-Π-CSP above tight lower bound:

Π-ABOVE AVERAGE (Π-AA)

Instance: A finite set V of variables, a multiset $\mathcal{C}$ of ordered triples of distinct variables from V and a nonnegative integer k.

Parameter: k.

Question: Is there an ordering α of V such that at least $\frac{|\Pi|}{6}|\mathcal{C}| + k$ constraints of $\mathcal{C}$ are Π-satisfied by α?

Key Results

The following is a simple but important observation in [4] allowing one to reduce Π-AA to MAX-3-LIN-ORDERING-AA.

Proposition 1 *Let Π be a subset of $\mathcal{S}_3$ such that $\Pi \notin \{\emptyset, \mathcal{S}_3\}$. There is a polynomial time transformation f from Π-AA to* MAX-3-LIN-ORDERING-AA *such that an instance $(V, \mathcal{C}, k)$ of Π-AA is a Yes-instance if and only if $(V, \mathcal{C}', k) = f(V, \mathcal{C}, k)$ is a Yes-instance of* MAX-3-LIN-ORDERING-AA.

Using a nontrivial reduction from MAX-3-LIN-ORDERING-AA to a combination of MAX-2-LIN-ORDERING-AA and BETWEENNESS-AA and the facts that both problems admit kernels with quadratic numbers of variables and constraints (proved in [3] and [2], respectively),

Gutin et al. [4] showed that MAX-3-LIN-ORDERING-AA also admits a kernel with quadratic numbers of variables and constraints. Kim and Williams [6] partially improved this result by showing that MAX-3-LIN-ORDERING-AA admits a kernel with $O(k)$ variables.

The polynomial-size kernel result for MAX-3-LIN-ORDERING-AA and Proposition 1 imply the following (see [4] for details):

Theorem 1 ([4]) *Let Π be a subset of $\mathcal{S}_3$ such that $\Pi \notin \{\emptyset, \mathcal{S}_3\}$. The problem Π-AA admits a polynomial-size kernel with $O(k^2)$ variables.*

Open Problems

Similar to Proposition 1, it is easy to prove that, for each fixed r every Π-AA can be reduced to LIN-r-ORDERING-AA. Gutin et al. [4] conjectured that for each fixed r the problem MAX-r-LIN-ORDERING-AA is fixed-parameter tractable.

Cross-References

- Kernelization, Constraint Satisfaction Problems Parameterized above Average
- Kernelization, MaxLin Above Average

Recommended Reading

1. Charikar M, Guruswami V, Manokaran R (2009) Every permutation CSP of arity 3 is approximation resistant. In: Computational complexity 2009, Paris, pp 62–73
2. Gutin G, Kim EJ, Mnich M, Yeo A (2010) Betweenness parameterized above tight lower bound. J Comput Syst Sci 76:872–878
3. Gutin G, Kim EJ, Szeider S, Yeo A (2011) A probabilistic approach to problems parameterized above tight lower bound. J Comput Syst Sci 77:422–429
4. Gutin G, van Iersel L, Mnich M, Yeo A (2012) All ternary permutation constraint satisfaction problems parameterized above average have Kernels with quadratic number of variables. J Comput Syst Sci 78:151–163
5. Guttmann W, Maucher M (2006) Variations on an ordering theme with constraints. In: 4th IFIP international conference on theoretical computer science-TCS 2006, Santiago. Springer, pp 77–90
6. Kim EJ, Williams R (2012) Improved parameterized algorithms for above average constraint satisfaction. In: IPEC 2011, Saarbrücken. Lecture notes in computer science, vol 7112, pp 118–131

Kernelization, Planar F-Deletion

Neeldhara Misra
Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India

Keywords

Finite-integer index; Meta-theorems; Protrusions; Treewidth

Years and Authors of Summarized Original Work

2012; Fomin, Lokshtanov, Misra, Saurabh
2013; Kim, Langer, Paul, Reidl, Rossmanith, Sau, Sikdar

Problem Definition

Several combinatorial optimization problems on graphs involve identifying a subset of nodes S, of the smallest cardinality, such that the graph obtained after removing S satisfies certain properties. For example, the VERTEX COVER problem asks for a minimum-sized subset of vertices whose removal makes the graph edgeless, while the FEEDBACK VERTEX SET problem involves finding a minimum-sized subset of vertices whose removal makes the graph acyclic. The $\mathcal{F}$-DELETION problem is a generic formulation that encompasses several problems of this flavor.

Let $\mathcal{F}$ be a finite set of graphs. In the $\mathcal{F}$-DELETION problem, the input is an n-vertex graph G and an integer k, and the question is if

G has a subset S of at most k vertices, such that $G-S$ does not contain a graph from $\mathcal{F}$ as a minor. The optimization version of the problem seeks such a subset of the smallest possible size. The PLANAR $\mathcal{F}$-DELETION problem is the version of the problem where $\mathcal{F}$ contains at least one planar graph. The $\mathcal{F}$-DELETION problem was introduced by [3], who gave a non-constructive algorithm running in time $O(f(k) \cdot n^2)$ for some function $f(k)$. This result was improved by [1] to $O(f(k) \cdot n)$, for $f(k) = 2^{2^{O(k \log k)}}$.

For different choices of sets of forbidden minors $\mathcal{F}$, one can obtain various fundamental problems. For example, when $\mathcal{F} = \{K_2\}$, a complete graph on two vertices, this is the VERTEX COVER problem. When $\mathcal{F} = \{C_3\}$, a cycle on three vertices, this is the FEEDBACK VERTEX SET problem. The cases of $\mathcal{F}$ being $\{K_{2,3}, K_4\}$, $\{K_4\}$, $\{\theta_c\}$, and $\{K_3, T_2\}$, correspond to removing vertices to obtain an outerplanar graph, a series-parallel graph, a diamond graph, and a graph of pathwidth one, respectively.

Tools

Most algorithms for the PLANAR $\mathcal{F}$-DELETION problem appeal to the notion of *protrusions* in graphs. An r-protrusion in a graph G is a subgraph H of treewidth at most r such that the number of neighbors of H in $G - H$ is at most r. Intuitively, a protrusion H in a graph G may be thought of as subgraph of small treewidth which is cut off from the rest of the graph by a small separator.

Usually, as a means of preprocessing, protrusions are identified and *replaced* by smaller ones, while maintaining equivalence. The notion of graph replacement in this fashion originates in the work of [4]. The modern notion of protrusion reductions have been employed in various contexts [2, 5, 6, 12]. A widely used method for developing a protrusion replacement algorithm is via the notion of *finite-integer index*. Roughly speaking, this property ensures that graphs can be related under some appropriate notion of equivalence with respect to the problem, and that there are only finitely many equivalence classes. This allows us to identify the class that the protrusion belongs to and replace it with a canonical representative for that class.

Key Results

The algorithms proposed for PLANAR $\mathcal{F}$-DELETION usually have the following ingredients. First, the fact that $\mathcal{F}$ contains a planar graph implies that any YES-instance of the problem must admit a small subset of vertices whose removal leads to a graph of small treewidth. It turns out that such graphs admit a convenient structure from the perspective of the existence of protrusions. In particular, most of the graph can be decomposed into protrusions. From here, there are two distinct themes.

In the first approach, the protrusions are *replaced* by smaller, equivalent graphs. Subsequently, we have a graph that has no large protrusions. For such instances, it can be shown that if there is a solution, there is always one that is incident to a constant fraction of the edges in the graph, and this leads to a randomized algorithm by branching. Notably, the protrusion replacement can be performed by an algorithm that guarantees the removal of a constant fraction of vertices in every application. This helps in ensuring that the overall running time of the algorithm has a linear dependence on the size of the input. This algorithm is limited to the case when all graphs in $\mathcal{F}$ are connected, as is required in demonstrating finite-integer index.

Theorem 1 ([8]) *When every graph in $\mathcal{F}$ is connected, there is a randomized algorithm solving* PLANAR $\mathcal{F}$-DELETION *in time* $2^{O(k)} \cdot n$.

The second approach involves exploring the structure of the instance further. Here, an $O(k)$-sized subset of vertices is identified, with the key property that there is a solution that lives within it. The algorithm then proceeds to exhaustively branch on these vertices. This technique requires a different protrusion decomposition from the previous one. The overall algorithm is implemented using iterative compression. Since the protrusions are not replaced, this algorithm works for all instances of PLANAR $\mathcal{F}$-DELETION, with-

out any further assumptions on the family $\mathcal{F}$. While both approaches lead to algorithms that are single-exponential in k, the latter has a quadratic dependence on the size of the input.

Theorem 2 ([11]) PLANAR-$\mathcal{F}$-DELETION *can be solved in time* $2^{O(k)} \cdot n^2$.

In the context of approximation algorithms, the protrusion replacement is more intricate, because the notion of equivalence is now more demanding. The replacement should preserve not only the exact solutions, but also approximate ones. By appropriately adapting the machinery of replacements with lossless protrusion replacers, the problem admits the following approximation algorithm.

Theorem 3 ([8]) PLANAR $\mathcal{F}$-DELETION *admits a randomized constant ratio approximation algorithm.*

The PLANAR $\mathcal{F}$-DELETION problem also admits efficient preprocessing algorithms. Formally, a kernelization algorithm for the problem takes an instance (G, k) as input and outputs an equivalent instance (H, k') where the size of the output is bounded by a function of k. If the size of the output is bounded by a polynomial function of k, then it is called a polynomial kernel. The reader is referred to the survey [13] for a more detailed introduction to kernelization.

The technique of protrusion replacement was developed and used successfully for kernelization algorithms on sparse graphs [2,5]. These methods were also used for the special case of the PLANAR $\mathcal{F}$-DELETION problem when $\mathcal{F}$ is a graph with two vertices and constant number of parallel edges [6]. In the general setting of PLANAR $\mathcal{F}$-DELETION, kernelization involves anticipating protrusions, that is, identifying subgraphs that become protrusions after the removal of some vertices from an optimal solution. These "near-protrusions" are used to find irrelevant edges, i.e., an edge whose removal does not change the problem, leading to natural reduction rules. The process of finding an irrelevant edge appeals to the well-quasi-ordering of a certain class of graphs as a subroutine.

Theorem 4 ([8]) PLANAR $\mathcal{F}$-DELETION *admits a polynomial kernel.*

Applications

The algorithms for PLANAR $\mathcal{F}$-DELETION apply to any vertex deletion problem that can be described as hitting minor models of some fixed finite family that contains a planar graph.

For a finite set of graphs $\mathcal{F}$, let $\mathcal{G}_{\mathcal{F},k}$ be a class of graphs such that for every $G \in \mathcal{G}_{\mathcal{F},k}$ there is a subset of vertices S of size at most k such that $G \setminus S$ has no minor from $\mathcal{F}$. The following combinatorial result is a consequence of the kernelization algorithm for PLANAR $\mathcal{F}$-DELETION.

Theorem 5 ([8]) *For every set* $\mathcal{F}$ *that contains a planar graph, every minimal obstruction for* $\mathcal{G}_{\mathcal{F},k}$ *is of size polynomial in* k.

Kernelization algorithms on apex-free and H-minor-free graphs for all bidimensional problems from [5] can be implemented in linear time by employing faster protrusion reducers. This leads to randomized linear time, linear kernels for several problems.

In the framework for obtaining EPTAS on H-minor-free graphs in [7], the running time of approximation algorithms for many problems is $f(1/\varepsilon) \cdot n^{O(g(H))}$, where g is some function of H only. The only bottleneck for improving polynomial-time dependence is a constant factor approximation algorithm for TREEWIDTH η-DELETION. Using Theorem 3 instead, each EPTAS from [7] runs in time $O(f(1/\varepsilon) \cdot n^2)$. For the same reason, the PTAS algorithms for many problems on unit disk and map graphs from [9] become EPTAS algorithms.

Open Problems

An interesting direction for further research is to investigate PLANAR $\mathcal{F}$-DELETION when none of the graphs in $\mathcal{F}$ is planar. The most interesting case here is when $\mathcal{F} = \{K_5, K_{3,3}\}$, also known

as the VERTEX PLANARIZATION problem. The work in [10] demonstrates an algorithm with running time $2^{O(k \log k)}n$, which notably has a linear-time dependence on n. It remains open as to whether VERTEX PLANARIZATION can be solved in $2^{O(k)}n$ time. The question of polynomial kernels in the non-planar setting is also open, in particular, even the specific case of $\mathcal{F} = \{K_5\}$ is unresolved.

Cross-References

- Bidimensionality
- Kernelization, Preprocessing for Treewidth
- Treewidth of Graphs

Recommended Reading

1. Bodlaender HL (1997) Treewidth: algorithmic techniques and results. In: 22nd international symposium on mathematical foundations of computer science (MFCS), Bratislava, vol 1295, pp 19–36
2. Bodlaender HL, Fomin FV, Lokshtanov D, Penninkx E, Saurabh S, Thilikos DM (2009) (Meta) kernelization. In: Proceedings of the 50th annual IEEE symposium on foundations of computer science (FOCS), Atlanta, pp 629–638
3. Fellows MR, Langston MA (1988) Nonconstructive tools for proving polynomial-time decidability. J ACM 35(3):727–739
4. Fellows MR, Langston MA (1989) An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations (extended abstract). In: Proceedings of the 30th annual IEEE symposium on foundations of computer science (FOCS), Research Triangle Park, pp 520–525
5. Fomin FV, Lokshtanov D, Saurabh S, Thilikos DM (2010) Bidimensionality and kernels. In: Proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms (SODA), Austin, pp 503–510
6. Fomin FV, Lokshtanov D, Misra N, Philip G, Saurabh S (2011) Hitting forbidden minors: approximation and kernelization. In: Proceedings of the 8th international symposium on theoretical aspects of computer science (STACS), LIPIcs, Dortmund, vol 9, pp 189–200
7. Fomin FV, Lokshtanov D, Raman V, Saurabh S (2011) Bidimensionality and EPTAS. In: Proceedings of the 22nd annual ACM-SIAM symposium on discrete algorithms (SODA), San Francisco. SIAM, pp 748–759
8. Fomin FV, Lokshtanov D, Misra N, Saurabh S (2012) Planar F-deletion: approximation, kernelization and optimal FPT algorithms. In: Proceedings of the 2012 IEEE 53rd annual symposium on foundations of computer science, New Brunswick, pp 470–479
9. Fomin FV, Lokshtanov D, Saurabh S (2012) Bidimensionality and geometric graphs. In: Proceedings of the 23rd annual ACM-SIAM symposium on discrete algorithms (SODA), Kyoto. SIAM, pp 1563–1575
10. Jansen BMP, Lokshtanov D, Saurabh S (2014) A near-optimal planarization algorithm. In: Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms, (SODA), Portland, pp 1802–1811
11. Kim EJ, Langer A, Paul C, Reidl F, Rossmanith P, Sau I, Sikdar S (2013) Linear kernels and single-exponential algorithms via protrusion decompositions. In: Proceedings of the 40th international colloquium on automata, languages, and programming (ICALP), Riga, Part I, pp 613–624
12. Langer A, Reidl F, Rossmanith P, Sikdar S (2012) Linear kernels on graphs excluding topological minors. CoRR abs/1201.2780
13. Lokshtanov D, Misra N, Saurabh S (2012) Kernelization – preprocessing with a guarantee. In: Bodlaender HL, Downey R, Fomin FV, Marx D (eds) The multivariate algorithmic revolution and beyond. Bodlaender, HansL. and Downey, Rod and Fomin, FedorV. and Marx, Dániel (eds) Lecture notes in computer science, vol 7370. Springer, Berlin/Heidelberg, pp 129–161

Kernelization, Polynomial Lower Bounds

Stefan Kratsch
Department of Software Engineering and Theoretical Computer Science, Technical University Berlin, Berlin, Germany

Keywords

Kernelization; Parameterized complexity; Satisfiability; Sparsification

Years and Authors of Summarized Original Work

2010; Dell, van Melkebeek

Problem Definition

The work of Dell and van Melkebeek [4] refines the framework for lower bounds for kernelization introduced by Bodlaender et al. [1] and Fortnow and Santhanam [6]. The main contribution is that their results yield a framework for proving polynomial lower bounds for kernelization rather than ruling out all polynomial kernels for a problem; this, for the first time, gives a technique for proving that some polynomial kernelizations are actually best possible, modulo reasonable complexity assumptions. A further important aspect is that, rather than studying kernelization directly, the authors give lower bounds for a far more general oracle communication protocol. In this way, they also obtain strong lower bounds for sparsification, lossy compression (in the sense of Harnik and Naor [7]), and probabilistically checkable proofs (PCPs).

To explain the connection between kernelization and oracle communication protocols, let us first recall the following. A *parameterized problem* is a language $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$; the second component k of instances $(x,k) \in \Sigma^* \times \mathbb{N}$ is called the *parameter*. A *kernelization for* $\mathcal{Q}$ *with size* $h\colon \mathbb{N} \to \mathbb{N}$ is an efficient algorithm that gets as input an instance $(x,k) \in \Sigma^* \times \mathbb{N}$ and returns an equivalent instance (x',k'), i.e., such that $(x,k) \in \mathcal{Q}$ if and only if $(x',k') \in \mathcal{Q}$, with $|x'|, k' \leq h(k)$. If $h(k)$ is polynomially bounded in k, then we also call it a polynomial kernelization.

One way to use a kernelization is to first simplify a given input instance and then solve the reduced instance by any (possibly brute-force) algorithm; together this yields an algorithm for solving the problem in question. If we abstract out the algorithm by saying that the answer for the reduced instance is given by an oracle, then we arrive at a special case of the following communication protocol.

Definition 1 (oracle communication protocol [4]) An *oracle communication protocol* for a language L is a communication protocol for two players. The first player is given the input x and has to run in time polynomial in the length of the input; the second player is computationally unbounded but is not given any part of x. At the end of the protocol, the first player should be able to decide whether $x \in L$. The cost of the protocol is the number of bits of communication from the first player to the second player.

As an example, if $\mathcal{Q}$ has a kernelization with size h, then instances (x,k) can be solved by a protocol of cost $h(k)$. It suffices that the first player can compute a reduced instance (x',k') and send it to the oracle who decides membership of (x',k') in $\mathcal{Q}$; this yields the desired answer for whether $(x,k) \in \mathcal{Q}$. Note that the communication protocol is far more general than kernelization because it makes no assumption about what exactly is sent (or in what encoding). More importantly, it also allows multiple rounds of communication, and the behavior of the oracle could also be active rather than just answering queries for the first player. Thus, the obtained lower bounds for oracle communication protocols are very robust, covering also relaxed forms of kernelization (like bikernels and compressions), and also yield the other mentioned applications.

Key Results

A central result in the work of Dell and van Melkebeek [4] (see also [5]) is the following lemma, called *complementary witness lemma*.

Lemma 1 (complementary witness lemma [4]) *Let L be a language and $t\colon \mathbb{N} \to \mathbb{N} \setminus \{0\}$ be polynomially bounded such that the problem of deciding whether at least one out of $t(s)$ inputs of length at most s belongs to L has an oracle communication protocol of cost $\mathcal{O}(t(s)\log t(s))$, where the first player can be conondeterministic. Then $L \in$ coNP/poly.*

A previous work of Fortnow and Santhanam [6] showed that an efficient algorithm for encoding any t instances $x_1, \ldots, x_t$ of size at most s into one instance y of size poly(s) such that $y \in L$ if and only if at least one x_i is in L implies $L \in$ coNP/poly. (We recall that this

settled the *OR-distillation conjecture* of Bodlaender et al. [1] and allowed their framework to rule out polynomial kernels under the assumption that NP $\not\subseteq$ coNP/poly.) Lemma 1 is obtained by a more detailed analysis of this result and requires an encoding of the OR of $t(s)$ instances into one instance of size $\mathcal{O}(t(s)\log t(s))$ rather than allowing only size poly(s) for all values of t. This focus on the number $t(s)$ of instances in relation to the maximum instance size s is the key for getting polynomial lower bounds for kernelization (and other applications). In this overview, we will not discuss the possibility of conondeterministic behavior of the first player, but the interested reader is directed to [9, 10] for applications thereof.

Before outlining further results of Dell and van Melkebeek [4], let us state a lemma that captures one way of employing the complementary witness lemma for polynomial lower bounds for kernelization. The lemma is already implicit in [4] and is given explicitly in follow-up work of Dell and Marx [3] (it can also be found in the current full version [5] of [4]). We recall that OR(L) refers to the language of all tuples $(x_1, \ldots, x_t)$ such that at least one x_i is contained in L.

Lemma 2 ([3, 5]) *Suppose that a parameterized problem Π has the following property for some constant c: For some* NP*-complete language L, there exists a polynomial-time mapping reduction from* OR(L) *to Π that maps an instance $(x_1, \ldots, x_t)$ of* OR(L) *in which each x_i has size at most s to an instance of Π with parameter $k \leq t^{1/c+o(1)} \cdot \text{poly}(s)$. Then Π does not have a communication protocol of cost $\mathcal{O}(k^{c-\epsilon})$ for any constant $\epsilon > 0$ unless* NP $\subseteq$ coNP/poly*, even when the first player is conondeterministic.*

Intuitively, Lemma 2 follows from Lemma 1 because if the reduction and communication protocol in Lemma 2 both exist (for all t), then we can choose $t(s)$ large enough (but polynomially bounded in s) such that for all s we get an oracle communication protocol of cost $\mathcal{O}(t(s))$ as required for Lemma 1. This implies $L \in$ coNP/poly and, hence, NP $\subseteq$ coNP/poly (since L is NP-complete). As discussed earlier, any kernelization yields an oracle communication protocol with cost equal to the kernel size and, thus, this bound carries over directly to kernelization.

Let us now state the further results of Dell and van Melkebeek [4] using the context of Lemma 2. The central result is the following theorem on lower bounds for vertex cover on d-uniform hypergraphs.

Theorem 1 ([4]) *Let $d \geq 2$ be an integer and ϵ a positive real. If* NP $\not\subseteq$ coNP/poly*, there is no protocol of cost $\mathcal{O}(n^{d-\epsilon})$ to decide whether a d-uniform hypergraph on n vertices has a vertex cover of at most k vertices, even when the first player is conondeterministic.*

To prove Theorem 1, Dell and van Melkebeek devise a reduction from OR(SAT) to CLIQUE on d-uniform hypergraphs parameterized by the number of vertices (fulfilling the assumption of Lemma 2 for $c = d$). This reduction relies on an intricate lemma, the *packing lemma*, that constructs a d-uniform hypergraph with t cliques on s vertices each, but having only about $\mathcal{O}(t^{1/d+o(1)} \cdot s)$ vertices and no further cliques of size s. In follow-up work, Dell and Marx [3] give a simpler proof for Theorem 1 without making use of the packing lemma, but use the lemma for another of their results.

Note that the stated bound for VERTEX COVER in d-uniform hypergraphs follows by complementation. Furthermore, since every nontrivial instance has $k \leq n$, this also rules out kernelization to size $\mathcal{O}(k^{d-\epsilon})$. The following lower bound for SATISFIABILITY is obtained by giving a reduction from VERTEX COVER on d-uniform hypergraphs with parameter n. In the reduction, hyperedges of size d are encoded by positive clauses on d variables (one per vertex), and an additional part of the formula (which requires $d \geq 3$) checks that at most k of these variables are set to true.

Theorem 2 ([4]) *Let $d \geq 3$ be an integer and ϵ a positive real. If* NP $\not\subseteq$ coNP/poly*, there is no*

protocol of cost $\mathcal{O}(n^{d-\epsilon})$ to decide whether an n-variable d-CNF formula is satisfiable, even when the first player is conondeterministic.

Finally, the following theorem proves that several known kernelizations for graph modification problems are already optimal. The theorem is proved by a reduction from VERTEX COVER (on graphs) with parameter k that is similar in spirit to the classical result of Lewis and Yannakakis on NP-completeness of the Π-VERTEX DELETION problem for nontrivial hereditary properties Π. Note that Theorem 3 requires that the property Π is not only hereditary, i.e., inherited by *induced* subgraphs, but inherited by *all subgraphs*.

Theorem 3 ([4]) *Let Π be a graph property that is inherited by subgraphs and is satisfied by infinitely many but not all graphs. Let ϵ be a positive real. If $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, there is no protocol of cost $\mathcal{O}(k^{2-\epsilon})$ for deciding whether a graph satisfying Π can be obtained from a given graph by removing at most k vertices.*

As an example, the theorem implies that the FEEDBACK VERTEX SET problem does not admit a kernelization with size $\mathcal{O}(k^{2-\epsilon})$. This is in fact tight since a kernelization by Thomassé [12] achieves $\mathcal{O}(k^2)$ vertices and $\mathcal{O}(k^2)$ edges (cf. [4]); improving to $\mathcal{O}(k^{2-\epsilon})$ edges is ruled out since it would yield an encoding in size $\mathcal{O}(k^{2-\epsilon'})$. Similarly, the well-known kernelization for VERTEX COVER to $2k$ vertices is tight and cannot, in general, be expected to yield instances with less than the trivial $\mathcal{O}(k^2)$ edges.

Applications

Several authors have used the present approach to get polynomial lower bounds for kernelizations of certain parameterized problems; see, e.g., [2, 3, 8, 11]. Similarly, some results make use of conondeterminism [9, 10] and the more general setting of lower bounds for oracle communication protocols [11].

Open Problems

Regarding applications it would be interesting to have more lower bounds that use the full generality of the oracle communication protocols. Furthermore, it is an open problem to relax the assumption of $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$ to the minimal $\mathsf{P} \neq \mathsf{NP}$.

Recommended Reading

1. Bodlaender HL, Downey RG, Fellows MR, Hermelin D (2009) On problems without polynomial kernels. J Comput Syst Sci 75(8):423–434
2. Cygan M, Grandoni F, Hermelin D (2013) Tight kernel bounds for problems on graphs with small degeneracy – (extended abstract). In: Bodlaender HL, Italiano GF (eds) ESA. Lecture notes in computer science, vol 8125. Springer, pp 361–372
3. Dell H, Marx D (2012) Kernelization of packing problems. In: SODA, Kyoto. SIAM, pp 68–81
4. Dell H, van Melkebeek D (2010) Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In: Schulman LJ (ed) STOC, Cambridge. ACM, pp 251–260
5. Dell H, van Melkebeek D (2010) Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. Electron Colloq Comput Complex 17:38
6. Fortnow L, Santhanam R (2011) Infeasibility of instance compression and succinct PCPs for NP. J Comput Syst Sci 77(1):91–106
7. Harnik D, Naor M (2010) On the compressibility of $\mathcal{NP}$ instances and cryptographic applications. SIAM J Comput 39(5):1667–1713
8. Hermelin D, Wu X (2012) Weak compositions and their applications to polynomial lower bounds for kernelization. In: SODA, Kyoto. SIAM, pp 104–113
9. Kratsch S (2012) Co-nondeterminism in compositions: a kernelization lower bound for a Ramsey-type problem. In: SODA, Kyoto. SIAM, pp 114–122
10. Kratsch S, Pilipczuk M, Rai A, Raman V (2012) Kernel lower bounds using co-nondeterminism: finding induced hereditary subgraphs. In: Fomin FV, Kaski P (eds) SWAT, Helsinki. Lecture notes in computer science, vol 7357. Springer, pp 364–375
11. Kratsch S, Philip G, Ray S (2014) Point line cover: the easy kernel is essentially tight. In: SODA, Portland. SIAM, pp 1596–1606
12. Thomassé S (2010) A quadratic kernel for feedback vertex set. ACM Trans Algorithms 6(2), 32:1–32:8

Kernelization, Preprocessing for Treewidth

Stefan Kratsch
Department of Software Engineering and Theoretical Computer Science, Technical University Berlin, Berlin, Germany

Keywords

Kernelization; Parameterized complexity; Preprocessing; Structural parameters; Treewidth

Years and Authors of Summarized Original Work

2013; Bodlaender, Jansen, Kratsch

Problem Definition

This work undertakes a theoretical study of preprocessing for the NP-hard TREEWIDTH problem of finding a *tree decomposition* of width at most k for a given graph G. In other words, given G and $k \in \mathbb{N}$, the question is whether G has *treewidth* at most k. Several efficient reduction rules are known that provably preserve the correct answer, and experimental studies show significant size reductions [3, 5]. The present results study these and further newly introduced rules and obtain upper and lower bounds within the framework of *kernelization* from parameterized complexity.

The general interest in computing tree decompositions is motivated by the well-understood approach of using dynamic programming on tree decompositions that is known to allow fast algorithms on graphs of bounded treewidth (but with runtime exponential in the treewidth). A bottleneck for practical applications is the need for finding, as a first step, a sufficiently good tree decomposition; the best known exact algorithm due to Bodlaender [2] runs in time exponential in k^3 and is thus only of theoretical interest. This motivates the use of heuristics and preprocessing to find a reasonably good tree decomposition quickly.

Tree Decompositions and Treewidth

A *tree decomposition* for a graph $G = (V, E)$ consists of a tree $T = (N, F)$ and a family $\mathcal{X} := \{X_i \mid i \in N, X_i \subseteq V\}$. The sets X_i are also called *bags* and the vertices of T are usually referred to as *nodes* to avoid confusion with G; there is exactly one bag X_i associated with each node $i \in N$. The pair $(T, \mathcal{X})$ must fulfill the following three properties: (1) Every vertex of G is contained in at least one bag; (2) For each edge $\{u, v\} \in E$ there must be a bag X_i containing both u and v; (3) For each vertex v of G the set of nodes i of T with $v \in X_i$ induce a (connected) subtree of T. The *width of a tree decomposition* $(T, \mathcal{X})$ is equal to the size of the largest bag $X_i \in \mathcal{X}$ minus one. The *treewidth* of a graph G, denoted $\mathsf{tw}(G)$, is the smallest width taken over all tree decompositions of G.

Parameters

The framework of parameterized complexity allows the study of the TREEWIDTH problem with respect to different *parameters*. A parameter is simply an integer value associated with each problem instance. The *standard parameter* for an optimization problem like TREEWIDTH is the desired solution quality k and we denote this problem by TREEWIDTH(k). Apart from this, *structural parameters* are considered that capture structural aspects of G. For example, the work considers the behavior of TREEWIDTH when the input graph G has a small vertex cover S, i.e., such that deletion of $\ell = |S|$ vertices yields an independent set, with ℓ being used as the parameter. Similarly, several other parameters are discussed, foremost among them the feedback vertex set number and the vertex deletion distance to a single clique; the corresponding vertex sets are called *modulators*, e.g., a feedback vertex set is a modulator to a forest. We denote the arising parameterized problems by TREEWIDTH(vc), TREEWIDTH(fvs), and TREEWIDTH($\mathsf{vc}(\overline{G})$). To decouple the overhead of finding, e.g., a mini-

mum vertex cover for G, all these variants assume that an appropriate modulator is given along with the input and the obtained guarantees are in terms of the size of this modulator. Since all studied parameters can be efficiently approximated to within a constant factor of the optimum, not providing an (optimal) modulator gives only a constant-factor blowup in the obtained results.

Kernelization

A kernelization for a problem with parameter ℓ is an efficient algorithm that given an instance (x, ℓ) returns an equivalent instance (x', ℓ') of size and parameter value ℓ' bounded by some computable function of ℓ. If the bound is polynomial in ℓ then we have a *polynomial kernelization*. Specialized to, for example, TREEWIDTH(vc) a polynomial kernelization would have the following behavior: It gets as input an instance (G, S, k), asking whether the treewidth of G is at most k, where S is a vertex cover for G. In polynomial time it creates an instance (G', S', k') such that: (1) The size of the instance (G', S', k') and the parameter value $|S'|$ are bounded polynomially in k; (2) The set S' is a vertex cover of G'; (3) The graph G has treewidth at most k if and only if G' has treewidth at most k'.

Key Results

The kernelization lower bound framework of Bodlaender et al. [6] together with recent results of Drucker [9] is known to imply that TREEWIDTH(k) admits no polynomial kernelization unless NP $\subseteq$ coNP/poly and the polynomial hierarchy collapses. The present work takes a more detailed look at polynomial kernelization for TREEWIDTH with respect to structural parameters. The results are as follows.

Theorem 1 TREEWIDTH(vc) *i.e., parameterized by vertex cover number, admits a polynomial kernelization to an equivalent instance with* $\mathcal{O}((\mathsf{vc}(G))^3)$ *vertices.*

An interesting feature of this result is that it uses only three simple reduction rules that are well known and often used (cf. [5]). Two rules address so-called simplicial vertices, whose neighborhood is a clique, and a third rule inserts edges between certain pairs of vertices that have a large number of shared vertices. Analyzing these empirically successful rules with respect to the vertex cover number of the input graph yields a kernelization. A fact that nicely complements the observed experimental success.

Theorem 2 TREEWIDTH(fvs) *i.e., parameterized by feedback vertex set number, admits a polynomial kernelization to an equivalent instance with* $\mathcal{O}((\mathsf{fvs}(G))^4)$ *vertices.*

The feedback vertex set number of a graph is upper bounded by its vertex cover number, and forests have feedback vertex set number zero but arbitrarily large vertex cover number. Thus, for large families of input graphs, this second result is stronger. The result again builds on several known reduction rules (including the above ones), among others, for handling vertices that are *almost simplicial*, i.e., all but one neighboring vertex form a clique. On top of these, several new rules are added. One of them addresses a previously uncovered case of almost simplicial vertex removal, namely, when the vertex has degree exactly $k + 1$, where k is the desired treewidth bound. Furthermore, these reduction rules lead to a structure dubbed *clique-seeing paths*, which takes a series of fairly technical rules and analysis to reduce and bound. Altogether, this combination leads to the above result.

Theorem 3 TREEWIDTH($\mathsf{vc}(\overline{G})$) *i.e., parameterized by deletion distance to a single clique, admits no polynomial kernelization unless* NP $\subseteq$ coNP/poly *and the polynomial hierarchy collapses.*

The proof uses the notion of a *cross-composition* introduced by Bodlaender et al. [8], which builds directly on the kernelization lower bound framework of Bodlaender et al. [6] and Fortnow and Santhanam [10]. The cross-composition builds on the proof of NP-

completeness of TREEWIDTH by Arnborg et al. [1], which uses a Karp reduction from CUTWIDTH to TREEWIDTH. This construction is extended significantly to yield a cross-composition of CUTWIDTH ON SUBCUBIC GRAPHS (i.e., graphs of maximum degree three) into TREEWIDTH, which, roughly, requires an encoding of many CUTWIDTH instances into a single instance of TREEWIDTH with sufficiently small parameter.

Overall, together with previously known results, the obtained upper and lower bounds for TREEWIDTH cover a wide range of natural parameter choices (see the discussion in [7]). If $\mathcal{C}$ is any graph class that contains all cliques, then the vertex deletion distance of a graph G to $\mathcal{C}$ is upper bounded by $\mathsf{vc}(\overline{G})$. Thus, TREEWIDTH parameterized by distance to $\mathcal{C}$ does not admit a polynomial kernelization unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. This includes several well-studied classes like interval graphs, cographs, and perfect graphs. Since TREEWIDTH remains NP-hard on bipartite graphs, the result for parameterization by feedback vertex set number cannot be generalized to vertex deletion to a bipartite graph. It may, however, be possible to generalize this parameter to vertex deletion distance to an outerplanar graph, i.e., planar graphs having an embedding with all vertices appearing on the outer face. Since these graphs generalize forests, this value is upper bounded by the feedback vertex set number.

Theorem 4 WEIGHTED TREEWIDTH(vc) *i.e., parameterized by vertex cover number, admits no polynomial kernelization unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ *and the polynomial hierarchy collapses.*

In the WEIGHTED TREEWIDTH problem, each vertex comes with an integer weight, and the size of a bag in the tree decomposition is defined as the sum of the weights of its vertices. (To note, the present paper uses an extra deduction of one such that treewidth and weighted treewidth coincide for graphs with all vertices having weight one.) The result is proved by a cross-composition from TREEWIDTH (to WEIGHTED TREEWIDTH parameterized by vertex cover number) and complements the polynomial kernelization for the unweighted case. A key idea for the cross-composition is to use a result of Bodlaender and Möhring [4] on the behavior of treewidth under the join operation on graphs. This is combined with replacing all edges (in input graphs and join edges) by using a small number of newly introduced vertices of high weight.

Open Problems

A particular interesting case left open by existing results on polynomial kernelization for structural parameterizations of TREEWIDTH is the vertex deletion distance to outerplanar graphs.

Recommended Reading

1. Arnborg S, Corneil DG, Proskurowski A (1987) Complexity of finding embeddings in a k-tree. SIAM J Algebra Discret 8(2):277–284. doi:10.1137/0608024
2. Bodlaender HL (1996) A linear-time algorithm for finding tree-decom-positions of small treewidth. SIAM J Comput 25(6):1305–1317. doi:10.1145/167088.167161
3. Bodlaender HL, Koster AMCA (2006) Safe separators for treewidth. Discret Math 306(3): 337–350. doi:10.1016/j.disc.2005.12.017
4. Bodlaender HL, Möhring RH (1993) The pathwidth and treewidth of cographs. SIAM J Discret Math 6(2):181–188. doi:10.1137/0406014
5. Bodlaender HL, Koster AMCA, van den Eijkhof F (2005) Preprocessing rules for triangulation of probabilistic networks. Comput Intell 21(3):286–305. doi:10.1111/j.1467-8640.2005.00274.x
6. Bodlaender HL, Downey RG, Fellows MR, Hermelin D (2009) On problems without polynomial kernels. J Comput Syst Sci 75(8):423–434
7. Bodlaender HL, Jansen BMP, Kratsch S (2013) Preprocessing for treewidth: a combinatorial analysis through kernelization. SIAM J Discret Math 27(4):2108–2142
8. Bodlaender HL, Jansen BMP, Kratsch S (2014) Kernelization lower bounds by cross-composition. SIAM J Discret Math 28(1):277–305
9. Drucker A (2012) New limits to classical and quantum instance compression. In: FOCS, New Brunswick. IEEE Computer Society, pp 609–618
10. Fortnow L, Santhanam R (2011) Infeasibility of instance compression and succinct PCPs for NP. J Comput Syst Sci 77(1):91–106

Kernelization, Turing Kernels

Henning Fernau
Fachbereich 4, Abteilung
Informatikwissenschaften, Universität Trier, Trier, Germany
Institute for Computer Science, University of Trier, Trier, Germany

Keywords

Karp reduction; Kernelization; Kernel size; Turing kernelization; Turing reduction

Years and Authors of Summarized Original Work

2012; Binkele-Raible, Fernau, Fomin, Lokshtanov, Saurabh
2013; Hermelin, Kratsch, Soltys, Wahlström, Wu
2014; Jansen

Definition and Discussion

The basic definition of the field expresses kernelization as a Karp (many-one) self-reduction. Classical complexity and recursion theory offers quite a lot of alternative and more general notions of reducibilities. The most general notion, that of a Turing reduction, motivates the following definition:

Let (Q, κ) be a parameterized problem over a finite alphabet Σ.

- An *input-bounded oracle* for (Q, κ) is an oracle that, for any given input $x \in \Sigma^*$ of (Q, κ) and any bound t, first checks if $|x|, |\kappa(x)| \leq t$, and if this is certified, it decides in constant time whether the input x is a YES instance of (Q, κ).
- A *Turing kernelization (algorithm)* for (Q, κ) is an algorithm that, provided with access to some input-bounded oracle for (Q, κ), decides on input $x \in \Sigma^*$ in polynomial time whether x is a YES instance of (Q, κ) or not. During its computation, the algorithm can produce (polynomially many) oracle queries x' with bound $t = h(\kappa(x))$, where h is an arbitrary computable function. The function h is referred to as the *size* of the kernel.

If only one oracle access is permitted in a run of the algorithm, we basically get the classical notion of a (many-one or Karp) kernelization.

A more general definition was given in [4], allowing access to a different (auxiliary) problem (Q', κ'). As long as there is a computable reduction from Q' to Q, this does not make much of a difference, as we could translate the queries to Q' into queries of Q. Therefore, we prefer to use the definition given in [1].

Out-Branching: Showing the Difference

In [1], the first example of a natural problem is provided that admits a Turing kernel of polynomial size, but (most likely) no Karp kernel of polynomial size. We provide some details in the following.

Problem Definition

A subdigraph T of a digraph D is an *out-tree* if T is an oriented tree with only one vertex r of indegree zero (called the *root*). The vertices of T of outdegree zero are called *leaves*. If T is a spanning out-tree, i.e., $V(T) = V(D)$, then T is called an *out-branching* of D. The Directed Maximum Leaf Out-Branching problem is to find an out-branching in a given digraph with the maximum number of leaves. The parameterized version of the Directed Maximum Leaf Out-Branching problem is k-Leaf Out-Branching, where for a given digraph D and integer k, it is asked to decide whether D has an out-branching with at least k leaves. If we replace "out-branching" with "out-tree" in the definition of k-Leaf Out-Branching, we get a problem called k-Leaf

OUT-TREE. The parameterization κ is set to k in both problems. As the two problems are easily translatable into each other, we focus on k-LEAF OUT-BRANCHING as the digraph analogue of the well-known MAXIMUM LEAF SPANNING TREE problem.

Key Results

It is shown that the problem variant where an explicit root is given as additional input, called ROOTED k-LEAF OUT-BRANCHING, admits a polynomial Karp kernel. Alternatively, this variant can be seen as a special case of k-LEAF OUT-BRANCHING by adding one vertex of indegree zero and outdegree one, pointing to the designated root of the original graph. By making a call to this oracle for each of the vertices as potential roots, this provides a Turing kernelization of polynomial size for k-LEAF OUT-BRANCHING. This result is complemented by showing that k-LEAF OUT-TREE has no polynomial Karp kernel unless $coNP \subseteq NP/poly$.

We list the reduction rules leading to the polynomial-size kernel for the rooted version in the following.

Reachability Rule:If there exists a vertex u which is disconnected from the root r, then return NO.

Useless Arc Rule:If vertex u disconnects a vertex v from the root r, then remove the arc vu.

Bridge Rule:If an arc uv disconnects at least two vertices from the root r, contract the arc uv.

Avoidable Arc Rule:If a vertex set S, $|S| \leq 2$, disconnects a vertex v from the root r, $vw \in A(D)$ and $xw \in A(D)$ for all $x \in S$, then delete the arc vw.

Two Directional Path Rule:If there is a path $P = p_1 p_2 \ldots p_{l-1} p_l$ with $l = 7$ or $l = 8$ such that

- p_1 and $p_{\text{in}} \in \{p_{l-1}, p_l\}$ are the only vertices with in-arcs from the outside of P
- p_l and $p_{\text{out}} \in \{p_1, p_2\}$ are the only vertices with out-arcs to the outside of P
- The path P is the unique out-branching of $D[V(P)]$ rooted at p_1
- There is a path Q that is the unique out-branching of $D[V(P)]$ rooted at p_{in} and ending in p_{out}
- The vertex after p_{out} on P is not the same as the vertex after p_l on Q

then delete $R = P \setminus \{p_1, p_{\text{in}}, p_{\text{out}}, p_l\}$ and all arcs incident to these vertices from D. Add two vertices u and v and the arc set $\{p_{\text{out}}u, uv, vp_{\text{in}}, p_l v, vu, up_1\}$ to D.

This reduction was simplified and improved in [2] by replacing the rather complicated last reduction rule by a rule that shortens induced bipaths of length four to length two. Here, $P = \{x_1, \ldots, x_l\}$, with $l \geq 3$, is an *induced bipath of length* $l - 1$ if the set of arcs neighbored to $\{x_2, \ldots, x_{l-1}\}$ in D is exactly $\{(x_i, x_{i+1}), (x_{i+1}, x_i) \mid i \in \{1, \ldots, l-1\}\}$. This yielded a Karp kernel with a quadratic number of vertices (measured in terms of the parameter k) for the rooted version. For directed acyclic graphs (DAGs), even a Karp kernel with a linear number of vertices is known for the rooted version [3]. Notice that also for DAGs (in fact, for quite restricted DAGs called *willow graphs*), the unrooted problem versions have no polynomial Karp kernel unless $coNP \subseteq NP/poly$, as suggested by the hardness proof in [1]. Another direction of research is to obtain faster kernelization algorithms, often by restricting the use (and power) of reduction rules. For the k-LEAF OUT-BRANCHING, this was done by Kammer [6].

Hierarchies Based on Turing Kernels

Based on the notion of *polynomial parametric transformation*, in [4] an intertwined WK/MK hierarchy was defined, in analogy to the well-known W/M hierarchy of (hard) parameterized problems. The lowest level (MK[1]) corresponds to (NP) problems with polynomial-size Karp kernels. The second-lowest level is WK[1], and this

does not equal MK[1] unless $coNP \subseteq NP/poly$. Typical complete problems for WK[1] are:

- Given a graph G of order n and an integer k, does G contain a clique of size k? Here, the parameterization is $\kappa(G,k) = k \cdot \log(n)$.
- Given a nondeterministic Turing machine M and an integer k, does M stop within k steps? Here, the parameterization is $\kappa(M,k) = k \cdot \log(|M|)$.

As noticed in [4], the CLIQUE problem provides also another (less natural) example of a problem without polynomial-size Karp kernel that has a polynomial-size Turing kernel, taking as parameterization the maximum degree of the input graph.

How Much Oracle Access Is Needed?

The examples we gave so far make use of oracles in a very simple way. More precisely, a very weak notion of truth-table reduction (disjunctive reduction) is applied. The INDEPENDENT SET problem on bull-free graphs [7] seems to provide a first example where the power of Turing reductions is used more extensively, as the oracle input is based on the previous computation of the reduction. Therefore, it could be termed an adaptive kernelization [5]. Yet another way of constructing Turing kernels was described by Jansen [5]. There, in a first step, the instance is decomposed (according to some graph decomposition in that case), and then the fact is used that either a solution is already obtained or it only exists in one of the (small) components of the decomposition. This framework is then applied to deduce polynomial-size Turing kernels, e.g., for the problem of finding a path (or a cycle) of length at least k in a planar graph G, where k is the parameter of the problem.

Open Problems

One of the most simple open questions is whether LONGEST PATH, i.e., the problem of finding a path of length at least k, admits a polynomial-size Turing kernel on general graphs.

Conversely, no tools have been developed so far that allow for ruling out polynomial-size Turing kernels. For the question of practical applications of kernelization, this would be a much stronger statement than ruling out traditional Karp kernels of polynomial size, as a polynomial number of polynomial-size kernels can give a practical solution (see the discussion of k-LEAF OUT-BRANCHING above).

Cross-References

▸ Enumeration of Paths, Cycles, and Spanning Trees
▸ Kernelization, Exponential Lower Bounds

Recommended Reading

1. Binkele-Raible D, Fernau H, Fomin FV, Lokshtanov D, Saurabh S, Villanger Y (2012) Kernel(s) for problems with no kernel: on out-trees with many leaves. ACM Trans Algorithms 8(4):38
2. Daligault J, Thomassé S (2009) On finding directed trees with many leaves. In: Chen J, Fomin FV (eds) Parameterized and exact computation, 4th international workshop, IWPEC, Copenhagen. LNCS, vol 5917. Springer, pp 86–97
3. Daligault J, Gutin G, Kim EJ, Yeo A (2010) FPT algorithms and kernels for the directed k-leaf problem. J Comput Syst Sci 76(2):144–152
4. Hermelin D, Kratsch S, Soltys K, Wahlström M, Wu X (2013) A completeness theory for polynomial (Turing) kernelization. In: Gutin G, Szeider S (eds) Parameterized and exact computation – 8th international symposium, IPEC, Sophia Antipolis. LNCS, vol 8246. Springer, pp 202–215
5. Jansen BMP (2014) Turing kernelization for finding long paths and cycles in restricted graph classes. Tech. Rep. 1402.4718v1, arXiv.CS.DS
6. Kammer F (2013) A linear-time kernelization for the rooted k-leaf outbranching problem. In: Brandstädt A, Jansen K, Reischuk R (eds) Graph-theoretic concepts in computer science – 39th international workshop, WG, Lübeck. LNCS, vol 8165. Springer, pp 310–320
7. Thomassé S, Trotignon N, Vuskovic K (2013) Parameterized algorithm for weighted independent set problem in bull-free graphs. CoRR abs/1310.6205, a conference version appeared at WG (LNCS volume) 2014

Kinetic Data Structures

Bettina Speckmann
Department of Mathematics and Computer Science, Technical University of Eindhoven, Eindhoven, The Netherlands

Years and Authors of Summarized Original Work

1999; Basch, Guibas, Hershberger

Problem Definition

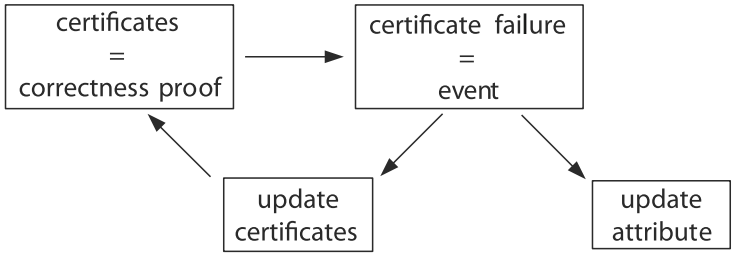
Many application areas of algorithms research involve objects in motion. Virtual reality, simulation, air-traffic control, and mobile communication systems are just some examples. Algorithms that deal with objects in motion traditionally discretize the time axis and compute or update their structures based on the position of the objects at every time step. If all objects move continuously then in general their configuration does not change significantly between time steps – the objects exhibit *spatial* and *temporal coherence*. Although *time-discretization* methods can exploit spatial and temporal coherence they have the disadvantage that it is nearly impossible to choose the perfect time step. If the distance between successive steps is too large, then important interactions might be missed, if it is too small, then unnecessary computations will slow down the simulation. Even if the time step is chosen just right, this is not always a satisfactory solution: some objects may have moved only slightly and in such a way that the overall data structure is not influenced.

One would like to use the temporal coherence to detect precisely those points in time when there is an actual change in the structure. The *kinetic data structure* (KDS) framework, introduced by Basch et al. in their seminal paper [2], does exactly that: by maintaining not only the structure itself, but also some additional information, they can determine when the structure will undergo a "real" (combinatorial) change.

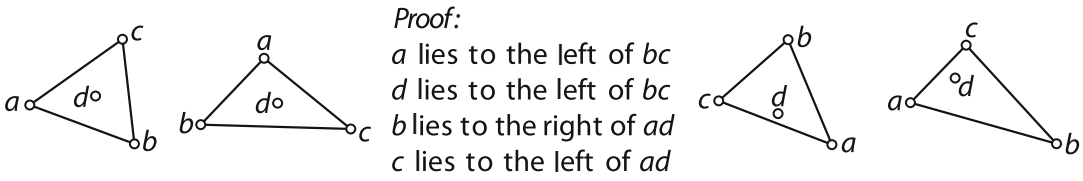
Key Results

A kinetic data structure is designed to maintain or monitor a discrete attribute of a set of moving objects, for example, the convex hull or the closest pair. The basic idea is, that although all objects move continuously, there are only certain discrete moments in time when the combinatorial structure of the attribute changes (in the earlier examples, the ordered set of convex-hull vertices or the pair that is closest, respectively). A KDS therefore contains a set of *certificates* that constitutes a proof of the property of interest. Certificates are generally simple inequalities that assert facts like "point c is on the left of the directed line through points a and b." These certificates are inserted in a priority queue (*event queue*) based on their time of expiration. The KDS then performs an event-driven simulation of the motion of the objects, updating the structure whenever an *event* happens, that is, when a certificate fails (see Fig. 1). It is part of the art of designing efficient kinetic data structures to find a small set of simple and easily updatable certificates that serve as a proof of the property one wishes to maintain.

A KDS assumes that each object has a known motion trajectory or *flight plan*, which may be subject to restrictions to make analysis tractable. Two common restrictions would be translation along paths parametrized by polynomials of fixed degree d, or translation and rotation described by algebraic curves. Furthermore, certificates are generally simple algebraic equations, which implies that the failure time of a certificate can be computed as the next largest root of an algebraic expression. An important aspect of kinetic data structures is their on-line character: although the positions and motions (flight plans) of the objects are known at all times, they are not necessarily known far in advance. In particular, any object can change its flight plan at any time. A good KDS should be able to handle such changes in flight plans efficiently.

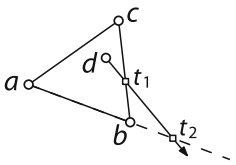


Kinetic Data Structures, Fig. 1 The basic structure of an event based simulation with a KDS



Kinetic Data Structures, Fig. 2 Equivalent convex hull configurations (*left* and *right*), a proof that a, b, and c form the convex hull of S (*center*)

Kinetic Data Structures, Fig. 3 Certificate structure for points a, b, and c being stationary and point d moving along a straight line



Certificate	Failure time
a lies to the left of bc	never
d lies to the left of bc	t_1
b lies to the right of ad	t_2
c lies to the left of ad	never

A detailed introduction to kinetic data structures can be found in Basch's Ph. D. thesis [1] or in the surveys by Guibas [3, 4]. In the following the principles behind kinetic data structures are illustrated by an easy example.

Consider a KDS that maintains the convex hull of a set S of four points a, b, c, and d as depicted in Fig. 2. A set of four simple certificates is sufficient to certify that a, b, and c form indeed the convex hull of S (see Fig. 2 center). This implies, that the convex hull of S will not change under any motion of the points that does not lead to a violation of these certificates. To put it differently, if the points move along trajectories that move them between the configurations depicted in Fig. 2 without the point d ever appearing on the convex hull, then the KDS in principle does not have to process a single event.

Now consider a setting in which the points a, b, and c are stationary and the point d moves along a linear trajectory (Fig. 3 left). Here the KDS has exactly two events to process. At time t_1 the certificate "d is to the left of bc" fails as the point d appears on the convex hull. In this easy setting, only the failed certificate is replaced by "d is to the *right* of bc" with failure time "never", generally processing an event would lead to the scheduling and descheduling of several events from the event queue. Finally at time t_2 the certificates "b is to the right of ad" fails as the point b ceases to be on the convex hull and is replaced by "b is to the *left* of ad" with failure time "never."

Kinetic data structures and their accompanying maintenance algorithms can be evaluated and compared with respect to four desired characteristics.

Responsiveness. One of the most important performance measures for a KDS is the time needed to update the attribute and to repair the certificate set when a certificate fails. A KDS is called *responsive* if this update time is "small", that is, polylogarithmic.

Compactness. A KDS is called *compact* if the number of certificates is near-linear in the

total number of objects. Note that this is not necessarily the same as the amount of storage the entire structure needs.

Locality. A KDS is called *local* if every object is involved in only a small number of certificates (again, "small" translates to polylogarithmic). This is important whenever an object changes its flight plane, because one has to recompute the failure times of all certificates this object is involved in, and update the event queue accordingly. Note that a local KDS is always compact, but that the reverse is not necessarily true.

Efficiency. A certificate failure does not automatically imply a change in the attribute that is being maintained, it can also be an *internal event*, that is, a change in some auxiliary structure that the KDS maintains. A KDS is called *efficient* if the worst-case number of events handled by the data structure for a given motion is small compared to the number of combinatorial changes of the attribute (*external events*) that must be handled for that motion.

Applications

The paper by Basch et al. [2] sparked a large amount of research activities and over the last years kinetic data structures have been used to solve various dynamic computational geometry problems. A number of papers deal foremost with the maintenance of discrete attributes for sets of moving points, like the closest pair, width and diameter, clusters, minimum spanning trees, or the constrained Delaunay triangulation. Motivated by ad hoc mobile networks, there have also been a number of papers that show how to maintain the connected components in a set of moving regions in the plane. Major research efforts have also been seen in the study of kinetic binary space partitions (BSPs) and kinetic kd-trees for various objects. Finally, there are several papers that develop KDSs for collision detection in the plane and in three dimensions. A detailed discussion and an extensive list of references can be found in the survey by Guibas [4].

Cross-References

▸ Fully Dynamic Minimum Spanning Trees
▸ Minimum Geometric Spanning Trees

Recommended Reading

1. Basch J (1999) Kinetic data structures. PhD thesis, Stanford University
2. Basch J, Guibas L, Hershberger J (1999) Data structures for mobile data. J Algorithms 31:1–28
3. Guibas L (1998) Kinetic data structures: a state of the art report. In: Proceedings of 3rd workshop on algorithmic foundations of robotics, pp 191–209
4. Guibas L (2004) Modeling motion. In: Goodman J, O'Rourke J (eds) Handbook of discrete and computational geometry, 2nd edn. CRC

Knapsack

Hans Kellerer
Department of Statistics and Operations Research, University of Graz, Graz, Austria

Keywords

Approximation algorithm; Fully polynomial time approximation scheme (FPTAS)

Years and Authors of Summarized Original Work

2000; Ibarra, Kim

Problem Definition

For a given set of items $N = \{1, \ldots, n\}$ with nonnegative integer weights w_j and profits p_j, $j = 1, \ldots, n$, and a knapsack of capacity c, the *knapsack problem* (KP) is to select a subset of the items such that the total profit of the selected items is maximized and the corresponding total weight does not exceed the knapsack capacity c.

Alternatively, a knapsack problem can be formulated as a solution of the following linear integer programming formulation:

$$(KP) \text{ maximize } \sum_{j=1}^{n} p_i x_j \tag{1}$$

$$\text{subject to } \sum_{j=1}^{n} w_j x_j \leq c, \tag{2}$$

$$x_j \in (0,1), \; j = 1, \ldots, n. \tag{3}$$

The knapsack problem is the simplest nontrivial integer programming model having binary variables, only a single constraint, and only positive coefficients. A large number of theoretical and practical papers have been published on this problem and its extensions. An extensive overview can be found in the books by Kellerer, Pferschy, and Pisinger [4] or Martello and Toth [7].

Adding the integrality condition (3) to the simple linear program (1)–(2) already puts (KP) into the class of $\mathcal{NP}$-hard problems. Thus, (KP) admits no polynomial time algorithms unless $\mathcal{P} = \mathcal{NP}$ holds.

Therefore, this entry will focus on approximation algorithms for (KP). A common method to judge the quality of an approximation algorithm is its worst-case performance. For a given instance I, define by $z^*(I)$ the optimal solution value of (KP) and by $z^H(I)$ the corresponding solution value of a heuristic H. For $\varepsilon \in [0,1[$, a heuristic H is called a $(1-\varepsilon)$-*approximation algorithm* for (KP) if for any instance I

$$z^H(I) \geq (1-\varepsilon) z^*(I)$$

holds. Given a parameter ε, a heuristic H is called a *fully polynomial approximation scheme*, or an FTPAS, if H is a $(1-\varepsilon)$-approximation algorithm for (KP) for any $\varepsilon \in [0,1[$, and its running time is polynomial both in the length of the encoded input n and $1/\varepsilon$. The first FTPAS for (KP) was suggested by Ibarra and Kim [1] in 1975. It was among the early FPTASes for discrete optimization problems. It will be described in detail in the following.

Key Results

(KP) can be solved in pseudopolynomial time by a simple dynamic programming algorithm. One possible variant is the so-called *dynamic programming by profits* (DP-Profits). The main idea of DP-Profits is to reach every possible total profit value with a subset of items of minimal total weight. Clearly, the highest total profit value, which can be reached by a subset of weight not greater than the capacity c, will be an optimal solution.

Let $y_j(q)$ denote the minimal weight of a subset of items from $\{1, \ldots, j\}$ with total profit equal to q. To bound the length of every array y_j, an upper bound u on the optimal solution value has to be computed. An obvious possibility would be to use the upper bound $U_{\mathrm{LP}} = \lfloor z^{\mathrm{LP}} \rfloor$ from the solution z^{LP} of the LP-relaxation of (KP) and set $U := U_{\mathrm{LP}}$. It can be shown that U_{LP} is at most twice as large as the optimal solution value z^*. Initializing $y_0(0) := 0$ and $y_0(q) := c+1$ for $q = 1, \ldots, U$, all other values can be computed for $j = 1, \ldots, n$ and $q = 0, \ldots, U$ by using the recursion

$$y_i(q) := \begin{cases} y_{i-1}(q) & \text{if } q < p_j, \\ \min\left(y_{i-1}(q), \left(y_{i-1}(q)\right)\right) & \text{if } q \geq p_j. \end{cases}$$

The optimal solution value is given by $\max\{q | y_n(q) \leq c\}$ and the running time of DP-Profits is bounded by $O(nU)$.

Theorem 1 (Ibarra, Kim) *There is an FTPAS for (KP) which runs in $O(n \log n + n/\varepsilon^2)$ time.*

Proof The FTPAS is based on appropriate *scaling* of the profit values p_j and then running DP-Profits with the scaled profit values. Scaling means here that the given profit values p_j are replaced by new profits $\tilde{p}_j$ such that $\tilde{p}_j := \left\lfloor \frac{p_j}{K} \right\rfloor$ for an appropriate chosen constant K.

This scaling can be seen as a partitioning of the profit range into intervals of length K with starting points $0, K, 2K, \ldots$. Naturally, for every profit value p_j, there is some integer value $i \geq 0$ such that p_j falls into the interval $[iK, (i+1)K[$. The scaling procedure generates for every p_j the

value $\tilde{p}_j$ as the corresponding index i of the lower interval bound iK.

Running DP-Profits yields a solution set $\tilde{X}$ for the scaled items which will usually be different from the original optimal solution set X^*. Evaluating the original profits of item set $\tilde{X}$ yields the approximate solution value z^H . The difference between z^H and the optimal solution value can be bounded as follows:

$$z^H \geq \sum_{j \in \tilde{X}} K \left\lfloor \frac{p_j}{K} \right\rfloor \geq \sum_{j \in \tilde{X}*} K \left\lfloor \frac{p_j}{K} \right\rfloor$$
$$\geq \sum_{j \in \tilde{X}*} K \left(\frac{p_j}{K} - 1 \right) = z* - |X*| \, K.$$

To get the desired performance guarantee of $1-\varepsilon$, it is sufficient to have

$$\frac{z* - z^H}{z*} \leq \frac{|X*| \, K}{z*} \leq \varepsilon.$$

To ensure this, K has to be chosen such that

$$K \leq \frac{\varepsilon z^*}{|X*|}. \tag{4}$$

Since $n \geq |X^*|$ and $U_{\mathrm{LP}}/2 \leq z^*$, choosing $K := \frac{\varepsilon U_{\mathrm{LP}}}{2n}$ satisfies condition (4) and thus guarantees the performance ratio of $1-\varepsilon$. Substituting U in the $O(nU)$ bound for DP-Profits by U/K yields an overall running time of $O(n^2\varepsilon)$.

A further improvement in the running time is obtained in the following way. Separate the items into *small* items (having profit $\leq \frac{\varepsilon}{2} U_{\mathrm{LP}}$) and *large* items (having profit $> \frac{\varepsilon}{2} U_{\mathrm{LP}}$). Then, perform DP-Profits for the scaled large items only. To each entry q of the obtained dynamic programming array with corresponding weight $y(q)$, the small items are added to a knapsack with residual capacity $c - y(q)$ in a greedy way. The small items shall be sorted in nonincreasing order of their profit to weight ratio. Out of the resulting combined profit values, the highest one is selected. Since every optimal solution contains at most $2/\varepsilon$ large items, $|X^*|$ can be replaced in (4) by $2/\varepsilon$ which results in an overall running time $O(n \log n + n/\varepsilon^2)$. The memory requirement of the algorithm is $O(n + 1/\varepsilon^3)$.

Two important approximation schemes with advanced treatment of items and algorithmic fine-tuning were presented some years later. The classical paper by Lawler [5] gives a refined scaling resp. partitioning of the items and several other algorithmic improvements which results in a running time $O(n \log(1/\varepsilon) + 1/\varepsilon^4)$. A second paper by Magazine and Oguz [6] contains among other features a partitioning and recombination technique to reduce the space requirements of the dynamic programming procedure. The fastest algorithm is due to Kellerer and Pferschy [2, 3] with running time $O(n \min\{\log n, \log(1/\varepsilon)\} + 1/\varepsilon^2 \log(1/\varepsilon) \cdot \min\{n, 1/\varepsilon \log(1/\varepsilon)\})$ and space requirement $O(n + 1/\varepsilon^2)$.

Applications

(KP) is one of the classical problems in combinatorial optimization. Since (KP) has this simple structure and since there are efficient algorithms for solving it, many solution methods of more complex problems employ the knapsack problem (sometimes iteratively) as a subproblem.

A straightforward interpretation of (KP) is an investment problem. A wealthy individual or institutional investor has a certain amount of money c available which he wants to put into profitable business projects. As a basis for his decisions, he compiles a long list of possible investments including for every investment the required amount w_j and the expected net return p_j over a fixed period. The aspect of risk is not explicitly taken into account here. Obviously, the combination of the binary decisions for every investment such that the overall return on investment is as large as possible can be formulated by (KP).

One may also view the (KP) as a "cutting" problem. Assume that a sawmill has to cut a log into shorter pieces. The pieces must however be cut into some predefined standard-lengths w_j , where each length has an associated selling price p_j . In order to maximize the profit of the log, the sawmill can formulate the problem as a (KP) where the length of the log defines the capacity c.

Among the wide range of "real-world" applications shall be mentioned two-dimensional

cutting problems, column generation, separation of cover inequalities, financial decision problems, asset-backed securitization, scheduling problems, knapsack cryptosystems, and most recent combinatorial auctions. For a survey on applications of knapsack problems, the reader is referred to [4].

Recommended Reading

1. Ibarra OH, Kim CE (1975) Fast approximation algorithms for the knapsack and sum of subset problem. J ACM 22:463–468
2. Kellerer H, Pferschy U (1999) A new fully polynomial time approximation scheme for the knapsack problem. J Comb Optim 3:59–71
3. Kellerer H, Pferschy U (2004) Improved dynamic programming in connection with an FPTAS for the knapsack problem. J Comb Optim 8:5–11
4. Kellerer H, Pisinger D, Pferschy U (2004) Knapsack problems. Springer, Berlin
5. Lawler EL (1979) Fast approximation algorithms for knapsack problems. Math Oper Res 4:339–356
6. Magazine MJ, Oguz O (1981) A fully polynomial approximation algorithm for the 0-1 knapsack problem. Eur J Oper Res 8:270–273
7. Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. Wiley, Chichester

Knowledge in Distributed Systems

Yoram Moses
Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa, Israel

Keywords

Common knowledge; Coordinated Attack; Distributed computing; Knowledge; Knowledge gain; Message chain; Potential causality

Years and Authors of Summarized Original Work

1984; Halpern, Moses
1986; Chandy, Misra

Problem Definition

What is the role of knowledge in distributed computing?

Actions taken by a process in a distributed system can only be based on its local information or local *knowledge*. Indeed, in reasoning about distributed protocols, people often talk informally about what processes know about the state of the system and about the progress of the computation. Can the informal reasoning about knowledge in distributed and multi-agent systems be given a rigorous mathematical formulation, and what uses can this have?

Key Results

In [4] Halpern and Moses initiated a theory of knowledge in distributed systems. They suggested that states of knowledge ascribed to groups of processes, especially *common knowledge*, have an important role to play. Knowledge-based analysis of distributed protocols has generalized well-known results and enables the discovery of new ones. These include new efficient solutions to basic problems, tools for relating results in different models, and proving lower bounds and impossibility results. For example, the inability to attain common knowledge when communication is unreliable was established in [4] and shown to imply and generalize the Coordinated Attack problem. Chandy and Misra showed in [1] that in asynchronous systems there is a tight connection between the manner in which knowledge is gained or lost and the message chains that underly Lamport's notion of potential causality.

Modeling Knowledge

In philosophy, knowledge is often modeled by so-called *possible-worlds* semantics. Roughly speaking, at a given "world," an agent will know a given fact to be true precisely if this fact holds at all worlds that the agent "considers possible." In a distributed system, the agents are processes or computing elements. A simple language

for reasoning about knowledge is obtained by starting out with a set $\boldsymbol{\Phi} = \{p, q, p', q' \ldots\}$ of propositions, or basic facts. The facts in Φ will depend on the application we wish to study; they may involve statements such as $x = 0$ or $x > y$ concerning values of variables or about other aspects of the computation (e.g., in the analysis of mutual exclusion, a proposition $\mathsf{CS}(i)$ may be used to state that process i is in the critical section). We obtain a logical language $\mathcal{L}_n^K = \mathcal{L}_n^K(\Phi)$ for knowledge, which is a set of formulas, by the following inductive definition. First, $p \in \mathcal{L}_n^K$ for all propositions $p \in \Phi$. Moreover, for all formulas $\varphi, \psi \in \mathcal{L}_n^K$, the language contains the formulas $\neg\varphi$ (standing for "*not* φ"), $\varphi \wedge \psi$ (standing for "φ *and* ψ"), and $K_i\varphi$ ("*process* i *knows* φ"), for every process $i \in \{1, \ldots, n\}$ (Using the operators "$\neg$" and "$\wedge$," we can express all of the Boolean operators. Thus, $\varphi \vee \psi$ ("φ *or* ψ") can be expressed as $\neg(\neg\varphi \wedge \neg\psi)$, while $\varphi \Rightarrow \psi$ ("φ *implies* ψ") is $\neg\varphi \vee \psi$, etc.). The language $\mathcal{L}_n^K$ is the basis of a propositional logic of knowledge. Using it, we can make formulas such as $K_1\mathsf{CS}(1) \wedge K_1K_2\neg\mathsf{CS}(2)$, which states that "*process 1 knows that it is in the critical section, and it knows that process 2 knows that 2 is not in the critical section*." $\mathcal{L}_n^K$ determines the *syntax* of formulas of the logic. A mathematical definition of what the formulas mean is called its *semantics*.

A process will typically know different things in different computations; even within a computation, its knowledge changes over time as a result of communication and of observing various events. We refer to time t in a run (or computation) r by the pair $\boldsymbol{(r, t)}$, which is called a *point*. Formulas are considered to be true or false at a point (r, t), with respect to a set of runs R (we call R a *system*). The set of points of R is denoted by $\boldsymbol{\mathsf{Pts}(R)}$.

The definition of knowledge is based on the idea that at any given point (r, t), each process has a well-defined *view*, which depends on i's history up to time t in r. This view may consist of all events that i has observed, or on a much more restricted amount of information, which is considered to be available to i at (r, t). In the language of [3], this view can be thought of as being process i's *local state* at (r, t), which we denote by $\boldsymbol{r_i(t)}$. Intuitively, a process is assumed to be able to distinguish two points iff its local state at one is different from its state in the other. In a given system R, the meaning of the propositions in a set Φ needs to be defined explicitly. This is done by way of an *interpretation* $\pi : \Phi \times \mathsf{Pts}(R) \rightarrow \{\mathsf{True}, \mathsf{False}\}$. The pair $\boldsymbol{\mathcal{I} = (R, \pi)}$ is called an *interpreted system*. We denote the fact that φ is satisfied, or true, at a point (r, t) in the system $\mathcal{I}$ by $(\mathcal{I}, r, t) \models \varphi$. Semantics of $\mathcal{L}_n^K(\Phi)$ with respect to an interpreted system $\mathcal{I}$ is given by defining the satisfaction relation "$\models$" defined by induction on the structure of the formulas, as follows:

$(\mathcal{I}, r, t) \models p$ *iff* $\pi\big(p, (r, t)\big) = \mathsf{True}$, for a proposition $p \in \Phi$

$(\mathcal{I}, r, t) \models \neg\varphi$ *iff* $(\mathcal{I}, r, t) \not\models \varphi$

$(\mathcal{I}, r, t) \models \varphi \wedge \psi$ *iff* both $(\mathcal{I}, r, t) \models \varphi$ and $(\mathcal{I}, r, t) \models \psi$

$(\mathcal{I}, r, t) \models K_i\varphi$ *iff* $(\mathcal{I}, r', t') \models \varphi$ whenever $r_i'(t') = r_i(t)$ and $(r', t') \in \mathsf{Pts}(R)$

The fourth clause, which defines satisfaction for knowledge formulas, can be applied repeatedly. This gives meaning to formulas involving knowledge about formulas that themselves involve knowledge, such as $K_2\big(\mathsf{CS}(2) \wedge \neg K_1\neg\mathsf{CS}(2)\big)$. Knowledge here is ascribed to processes. The intuition is that the local state captures all of the information available to the process. If there is a scenario leading to another point at which a fact φ is false and the process has the same state as it has not, then the process does not know φ.

This notion of knowledge has fairly strong properties that distinguish it from what one might

consider a reasonable notion of, say, human knowledge. For example, it does not depend on computation, thoughts, or a derivation of what the process knows. It is purely "information based." Indeed, any fact that holds at all elements of $\mathsf{Pts}(R)$ (e.g., the protocol that processes are following) is automatically known to all processes. Moreover, it is not assumed that a process can report its knowledge or that its knowledge is explicitly recorded in the local state. This notion of knowledge can be thought of as being ascribed to the processes by an external observer and is especially useful for analysis by a protocol designer.

Common Knowledge and Coordinated Attack

A classic example of a problem for which the knowledge terminology can provide insight is Jim Gray's *Coordinated Attack* problem. We present it in the style of [4]:

The Coordinated Attack Problem

Two divisions of an army are camped on two hilltops, and the enemy awaits in the valley below. Neither general will decide to attack unless he is sure that the other will attack with him, because only a simultaneous attack guarantees victory. The divisions do not initially have plans to attack, and one of the commanding generals wishes to coordinate a simultaneous attack (at some time the next day). The generals can only communicate by means of a messenger. Normally, it takes the messenger 1 h to get from one encampment to the other. However, the messenger can lose his way or be captured by the enemy. Fortunately, on this particular night, everything goes smoothly. How long will it take them to coordinate an attack?

It is possible to show by induction that k trips of the messenger do not suffice, for all $k \geq 0$, and hence the generals will be unable to attack. Gray used this example to illustrate the impact of unreliable communication on the ability to consistently update distinct sites of a distributed database. A much stronger result that generalizes this and applies directly to practical problems can be obtained based on a notion called *common knowledge*. Given a group $G \subseteq \{1, \ldots, n\}$ of processes, we define two new logical operators E_G and C_G, corresponding to *everyone (in G) knows* and *is common knowledge in G*, respectively. We shall denote $E_G^1\varphi = E_G\varphi$ and inductively define $E_G^{k+1}\varphi = E_G(E_G^k\varphi)$. Satisfaction for the new operators is given by

$$(\mathcal{I}, r, t) \models E_G\varphi \ \textbf{\textit{iff}} \ (\mathcal{I}, r, t) \models K_i\varphi \text{ holds for all } i \in G$$

$$(\mathcal{I}, r, t) \models C_G\varphi \ \textit{iff} \ (\mathcal{I}, r, t) \models E_G^k\varphi \text{ holds for all } k \geq 1$$

Somewhat surprisingly, common knowledge is not uncommon in practice. People shake hands to signal that they attained common knowledge of an agreement, for example. Similarly, a public announcement to a class or to an audience is considered common knowledge. Indeed, as we now discuss, simultaneous actions can lead to common knowledge.

Returning to the Coordinated Attack problem, consider three propositions attack_A, attack_B, and $\mathsf{delivered}$, corresponding, respectively, to "*general A is attacking*," "*general B is attacking*," and "*at least one message has been delivered*." The fact that the generals do not have a plan to attack can be formalized by saying that at least one of them does not attack unless $\mathsf{delivered}$ is true. Consider a set of runs R consisting of all possible interactions of the generals in the above setting. Suppose that the generals follow the specifications, so they only ever attack simultaneously at points of R. Then, roughly speaking, since the generals' actions depend on their local state, general A knows when attack_A is true. But since they only attack simultaneously and attack_B is true whenever attack_B is true, $K_A\mathsf{attack}_B$ will hold whenever general A attacks. Since B similarly knows when A attacks, $K_BK_A\mathsf{attack}_B$ will hold as well. Indeed, it can be shown that when the generals attack in a system that guarantees that attacks are simultaneous,

they must have common knowledge that they are attacking.

Theorem 1 (Halpern and Moses [4]) *Let R be a system with unreliable communication, let $\mathcal{I} = (R, \pi)$, let $(r, t) \in \mathsf{Pts}(R)$, and assume that $|G| > 1$. Then $(\mathcal{I}, r, t) \models \neg C_G \mathsf{delivered}$.*

As in the case of Coordinated Attack, simultaneous actions must be common knowledge when they are performed. Moreover, in cases in which such actions require a minimal amount of communication to be materialize, $C_G \mathsf{delivered}$ must hold when they are performed. Theorem 1 implies that no such actions can be coordinated when communication is unreliable. One immediate consequence is:

Corollary 1 *Under a protocol that satisfies the constraints of the Coordinated Attack problem, the generals never attack.*

The connection between common knowledge and simultaneous actions goes even deeper. It can be shown that when a fact that is not common knowledge to G becomes common knowledge to G, a state transition must occur simultaneously at all sites in G. If simultaneity cannot be coordinated in the system, common knowledge cannot be attained. This raises some philosophical issues: Events and transitions that are viewed as being simultaneous in a system that is modeled at a particular ("coarse") granularity of time will fail to be simultaneous when time is modeled at a finer granularity. As discussed in [4], this is not quite a paradox, since there are many settings in which it is acceptable, and even desirable, to model interactions at a granularity of time in which simultaneous transitions do occur.

A Hierarchy of States of Knowledge and Common Knowledge

Common knowledge is a much stronger state of knowledge than, say, knowledge of an individual process. Indeed, it is best viewed as a state of knowledge of a group. There is an essential difference between E_G^k (everyone knows that everyone knows, for k levels), even for large k, and C_G (common knowledge). Indeed, for every k, there are examples of tasks that can be achieved if $E_G^{k+1}\varphi$ holds but not if $E_G^k\varphi$ does. This suggests the existence of a hierarchy of states of group knowledge, ranging from $E_G\varphi$ to $C_G\varphi$. But it is also possible to define natural states of knowledge for a group that are weaker than these. One is S_G, where $S_G\varphi$ is true if $\bigvee_{i \in G} K_i\varphi$ – *someone in G knows φ*. Even weaker is *distributed knowledge*, denoted by D_G, which is defined by

$$(\mathcal{I}, r, t) \models D_G\varphi \text{ iff } (\mathcal{I}, r', t') \models \varphi \text{ for all } (r', t') \text{ satisfying } r_i'(t') = r_i(t) \text{ for all } i \in G$$

Roughly speaking, the distributed knowledge of a group corresponds to what follows from the combined information of the group at a given instant. Thus, for example, if all processes start out with initial value 1, they will have distributed knowledge of this fact, even if no single process knows this individually. Halpern and Moses propose a hierarchy of states of group knowledge and suggest that communication can often be seen as the process of moving the state of knowledge up the hierarchy:

$$C_G\varphi \Rightarrow E_G^{k+1}\varphi \Rightarrow E_G^k\varphi \Rightarrow \cdots \Rightarrow E_G\varphi \Rightarrow S_G\varphi \Rightarrow D_G\varphi.$$

Knowledge Gain and Loss in Asynchronous Systems

In asynchronous systems there are no guarantees about the pace at which communication is delivered and no guarantees about the relative rates at which processes operate. This motivated Lamport's definition of the happened-before relation among events. It is based on the intuition that in asynchronous systems only information obtained via message chains can affect the activity at a

given site. A crisp formalization of this intuition was discovered by Chandy and Misra in [1]:

Theorem 2 (Chandy and Misra) *Let $\mathcal{I}$ be an asynchronous interpreted system, let $\varphi \in \mathcal{L}_n^K$, and let $t' > t$. Then*

Knowledge Gain: *If $(\mathcal{I}, r, t) \not\models K_j \varphi$ and $(\mathcal{I}, r, t') \models K_{i_m} K_{i_{m-1}} \cdots K_{i_1} \varphi$, then there is a message chain through processes $\langle i_1, i_2, \ldots, i_m \rangle$ in r between times t and t'.*

Knowledge Loss: *If $(\mathcal{I}, r, t) \models K_{i_m} K_{i_{m-1}} \cdots K_{i_1} \varphi$ and $(\mathcal{I}, r, t') \not\models \varphi$, then there is a message chain through processes $\langle i_m, i_{m-1}, \ldots, i_1 \rangle$ in r between times t and t'.*

Note that the second clause implies that sending messages can cause a process to lose knowledge about other sites. Roughly speaking, the only way a process can know a nontrivial fact about a remote site is if this fact can only be changed by explicit permission from the process.

Applications and Extensions

The knowledge framework has been used in several ways. We have already seen its use for proving impossibility results in the discussion of the Coordinated Attack example. One interesting use of the formalism is as a tool for expressing *knowledge-based* protocols, in which programs can contain tests such as *if* K_i(*msg received*) *then*..., Halpern and Zuck, for example, showed that distinct solutions to the sequence transmission problem under different assumptions regarding communication faults were all implementations of the same knowledge-based protocol [5].

A knowledge-based analysis can lead to the design of efficient, sometimes optimal, distributed protocols. Dwork and Moses analyzed when facts become common knowledge when processes can crash and obtained an efficient and optimal solution to simultaneous consensus in which decisions are taken when initial values become common knowledge [2]. Moses and Tuttle showed that in a slightly harsher failure model, similar optimal solutions exist, but they are not computationally efficient, because computing when values are common knowledge is NP-hard [7]. A thorough exposition of reasoning about knowledge in a variety of fields including distributed systems, game theory, and philosophy appears in [3], while a later discussion of the role of knowledge in coordination, with further references, appears in [6].

Cross-References

- Byzantine Agreement
- Causal Order, Logical Clocks, State Machine Replication
- Distributed Snapshots

Recommended Reading

1. Chandy KM, Misra J (1986) How processes learn. Distrib Comput 1(1):40–52
2. Dwork C, Moses Y (1990) Knowledge and common knowledge in a Byzantine environment: crash failures. Inf Comput 88(2):156–186
3. Fagin R, Halpern JY, Moses Y, Vardi MY (2003) Reasoning about knowledge. MIT, Cambridge, MA
4. Halpern JY, Moses Y (1990) Knowledge and common knowledge in a distributed environment. J ACM 37(3):549–587. A preliminary version appeared in Proceeding of the 3rd ACM symposium on principles of distributed computing, 1984
5. Halpern JY, Zuck LD (1992) A little knowledge goes a long way: knowledge-based derivations and correctness proofs for a family of protocols. J ACM 39(3):449–478
6. Moses Y (2015) Relating knowledge and coordinated action: the knowledge of preconditions principle. In: Proceedings of the 15th TARK conference, Pittsburgh, pp 207–215
7. Moses Y, Tuttle MR (1988) Programming simultaneous actions using common knowledge. Algorithmica 3:121–169

L

Large-Treewidth Graph Decompositions

Julia Chuzhoy
Toyota Technological Institute, Chicago, IL, USA

Keywords

Bidimensionality theory; Excluded grid theorem; Erdős-Pósa; Graph decomposition; Treewidth

Years and Authors of Summarized Original Work

2013; Chekuri, Chuzhoy

Problem Definition

Treewidth is an important and a widely used graph parameter. Informally, the treewidth of a graph measures how close the graph is to being a tree. In particular, low-treewidth graphs often exhibit behavior somewhat similar to that of trees, in that many problems can be solved efficiently on such graphs, often by using dynamic programming. The treewidth of a graph $G = (V, E)$ is typically defined via tree decompositions. A tree decomposition for G consists of a tree $T = (V(T), E(T))$ and a collection of sets $\{X_v \subseteq V\}_{v \in V(T)}$ called *bags*, such that the following two properties are satisfied: (i) for each edge $(a, b) \in E$, there is some node $v \in V(T)$ with both $a, b \in X_v$, and (ii) for each vertex $a \in V$, the set of all nodes of T whose bags contain a form a nonempty (connected) subtree of T. The *width* of a given tree decomposition is $\max_{v \in V(T)}\{|X_v|-1\}$, and the treewidth of a graph G, denoted by $\mathrm{tw}(G)$, is the width of a minimum-width tree decomposition for G.

In large-treewidth graph decompositions, we seek to partition a given graph G into a large number of disjoint subgraphs $G_1, \ldots, G_h$, where each subgraph G_i has a large treewidth. Specifically, if k denotes the treewidth of G, h is the desired number of the subgraphs in the decomposition, and r is the desired lower bound on the treewidth of each subgraph G_i, then we are interested in efficient algorithms that partition any input graph G of treewidth k into h disjoint subgraphs of treewidth at least r each, and in establishing the bounds on h and r in terms of k, for which such a partition exists.

Key Results

The main result of [1] is summarized in the following theorem.

Theorem 1 *There is an efficient algorithm that, given integers $h, r, k \geq 0$, where either $hr^2 \leq k/\operatorname{poly}\log k$ or $h^3 r \leq k/\operatorname{poly}\log k$ holds, and a graph G of treewidth k, computes a partition of G into h disjoint subgraphs of treewidth at least r each.*

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

Applications

While low-treewidth graphs can often be handled well by dynamic programming, the major tool for dealing with large-treewidth graphs so far has been the Excluded Grid Theorem of Robertson and Seymour [11]. The theorem states that there is some function $g : \mathbb{Z}^+ \to \mathbb{Z}^+$, such that for any integer t, every graph of treewidth at least $g(t)$ contains a $(t \times t)$-grid as a minor (we say that a graph H is a minor of G iff we can obtain H from G by a sequence of edge deletions and edge contractions). A long line of work is dedicated to improving the upper and the lower bounds on the function g [2, 6, 7, 9–12]. The best current bounds show that the theorem holds for $g(t) = O(t^{98} \cdot \text{poly}\log(t))$ [2], and the best negative result shows that $g(t) = \Omega(t^2 \log t)$ must hold [12]. Robertson et al. [12] suggest that $g(t) = \Theta(t^2 \log t)$ may be sufficient, and Demaine et al. [5] conjecture that the bound of $g(t) = \Theta(t^3)$ is both necessary and sufficient. Large-treewidth graph decomposition is a tool that allows, in several applications, to bypass the Excluded Grid Theorem while obtaining stronger parameters. Such applications include Erdős-Pósa-type results and fixed-parameter tractable algorithms that rely on the bidimensionality theory. We note that the Excluded Grid Theorem of Robertson and Seymour provides a large-treewidth graph decomposition with weaker bounds. The most recent polynomial bounds for the Excluded Grid Theorem of [2] only ensure that a partition exists for any h, r where $h^{49} r^{98} \leq k / \text{poly}\log k$. Prior to the work of [1], the state-of-the-art bounds for the Grid-Minor Theorem could only guarantee that the partition exists whenever $hr^2 \leq \log k / \log\log k$.

We now provide several examples where the large-treewidth graph decomposition theorem can be used to improve previously known bounds.

Erdős-Pósa-Type Results

A family $\mathcal{F}$ of graphs is said to satisfy the Erdős-Pósa property, iff there is an integer-valued function $f_{\mathcal{F}}$, such that for every graph G, either G contains k disjoint subgraphs isomorphic to members of $\mathcal{F}$, or there is a set S of $f_{\mathcal{F}}(k)$ nodes, such that $G \setminus S$ contains no subgraph isomorphic to a member of $\mathcal{F}$. In other words, S is a cover, or a hitting set, for $\mathcal{F}$ in G. Erdős and Pósa [8] showed such a property when $\mathcal{F}$ is the family of cycles, with $f_{\mathcal{F}}(k) = \Theta(k \log k)$.

The Excluded Grid Theorem has been widely used in proving Erdős-Pósa-type results, where the specific parameters obtained depend on the best known upper bound on the function $g(k)$ in the Excluded Grid Theorem. The parameters in many Erdős-Pósa-type results can be significantly strengthened using Theorem 1, as shown in the following theorem:

Theorem 2 *Let $\mathcal{F}$ be any family of connected graphs and assume that there is an integer r, such that any graph of treewidth at least r is guaranteed to contain a subgraph isomorphic to a member of $\mathcal{F}$. Then $f_{\mathcal{F}}(k) \leq O(kr^2 \text{poly}\log(kr))$.*

Combining Theorem 2 with the best current bound for the Excluded Grid Theorem [2], we obtain the following corollary.

Corollary 1 *Let $\mathcal{F}$ be any family of connected graphs, such that for some integer q, any graph containing a $q \times q$ grid as a minor is guaranteed to contain a subgraph isomorphic to a member of $\mathcal{F}$. Then $f_{\mathcal{F}}(k) \leq O(q^{98} k \text{poly}\log(kq))$.*

For a fixed graph H, let $\mathcal{F}(H)$ be the family of all graphs that contain H as a minor. Robertson and Seymour [11], as one of the applications of their Excluded Grid Theorem, showed that $\mathcal{F}(H)$ has the Erdős-Pósa property iff H is planar. By directly applying Corollary 1, we get the following improved near-linear dependence on k.

Theorem 3 *For any fixed planar graph H, the family $\mathcal{F}(H)$ of graphs has the Erdős-Pósa property with $f_{\mathcal{F}(H)}(k) = O(k \cdot \text{poly}\log(k))$.*

Improved Running Times for Fixed-Parameter Tractability

The theory of bidimensionality [3] is a powerful methodology in the design of fixed-parameter tractable (FPT) algorithms. It led

to sub-exponential (in the parameter k) time FPT algorithms for bidimensional parameters in planar graphs and more generally graphs that exclude a fixed graph H as a minor. The theory is based on the Excluded Grid Theorem. However, in general graphs, the weak bounds of the Excluded Grid Theorem meant that one could only derive FPT algorithms with running time of the form $2^{k^c} n^{O(1)}$, for some large constant c, by using the results of Demaine and Hajiaghayi [4], and the recent polynomial bounds for the Excluded Grid Theorem [2]. Using Theorem 1, we can obtain algorithms with running times of the form $2^{k \text{ poly}\log(k)} n^{O(1)}$ for the same class of problems as in [4].

Open Problems

The authors conjecture that there is an efficient algorithm that, given integers k, r, h with $hr \leq k/\text{poly}\log k$, and any graph G of treewidth k, finds a partition of G into h disjoint subgraphs of treewidth at least r each. This remains an open problem.

Experimental Results

None are reported.

URLs to Code and Data Sets

None are reported.

Recommended Reading

1. Chekuri C, Chuzhoy J (2013) Large-treewidth graph decompositions and applications. In: Proceedings of ACM STOC, Palo Alto, pp 291–300
2. Chekuri C, Chuzhoy J (2014) Polynomial bounds for the grid-minor theorem. In: STOC, New York
3. Demaine E, Hajiaghayi M (2007) The bidimensionality theory and its algorithmic applications. Comput J 51(3):292–302
4. Demaine ED, Hajiaghayi M (2007) Quickly deciding minor-closed parameters in general graphs. Eur J Comb 28(1):311–314
5. Demaine E, Hajiaghayi M, Kawarabayashi Ki (2009) Algorithmic graph minor theory: improved grid minor bounds and Wagner's contraction. Algorithmica 54:142–180. http://dx.doi.org/10.1007/s00453-007-9138-y
6. Diestel R (2012) Graph theory. Graduate texts in mathematics, vol 173, 4th edn. Springer, Berlin
7. Diestel R, Jensen TR, Gorbunov KY, Thomassen C (1999) Highly connected sets and the excluded grid theorem. J Comb Theory Ser B 75(1):61–73
8. Erdos P, Pósa L (1965) On independent circuits contained in a graph. Can J Math 17:347–352
9. Kawarabayashi K, Kobayashi Y (2012) Linear min-max relation between the treewidth of H-minor-free graphs and its largest grid minor. In: Proceedings of STACS, Paris
10. Leaf A, Seymour P (2012) Treewidth and planar minors, manuscript. Available at https://web.math.princeton.edu/~pds/papers/treewidth/paper.pdf
11. Robertson N, Seymour PD (1986) Graph minors. V. Excluding a planar graph. J Comb Theory Ser B 41(1):92–114
12. Robertson N, Seymour P, Thomas R (1994) Quickly excluding a planar graph. J Comb Theory Ser B 62(2):323–348

L

Layout Decomposition for Multiple Patterning

Haitong Tian and Martin D.F. Wong
Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA

Keywords

Double patterning; Layout decomposition; Lithography; Triple patterning

Years and Authors of Summarized Original Work

2012; Haitong Tian

Problem Definition

As the feature size keeps shrinking, there are increasing difficulties to print circuit patterns using

single litho exposure. For 32/22 nm technology nodes, double patterning lithography (DPL) is one of the most promising techniques for the industry. In DPL, a layout is decomposed into two masks, where each feature in the layout is uniquely assigned to one of the masks. By using two masks which go through two separate exposures, better printing resolution can be achieved. For 14/10 nm technology node and beyond, triple patterning lithography (TPL) is one technique to obtain qualified printing results. In TPL, a layout is decomposed into three masks which further enhance the printing resolution. Currently, DPL and TPL are the two most studied multiple patterning techniques for advanced technology nodes [1–7]. Multiple patterning techniques such as quadruple patterning lithography and beyond usually are not investigated because of their increasing mask cost and other technical issues.

For DPL/TPL, there is a minimum coloring distance $d_{\min}$. If the distance of two features is less than $d_{\min}$, they cannot be printed in the same mask. $d_{\min}$ reflects the printing capabilities of current technology and can be redeemed as a constant in the problem. One practical concern of DPL/TPL is feature splitting, in which a feature is split into two or more parts for a legal color assignment. Such a splitting is called a stitch, which increases manufacturing cost and complexity due to additional line ends and more tight overlay control. Therefore, minimizing the number of stitches is a key objective for DPL/TPL decompositions. Other concerns include minimizing design rule violations, maximizing the overlap length, and balancing the usage of different colors. Among these concerns for DPL/TPL, minimizing the number of stitches is the most commonly studied one.

Multiple patterning decomposition is essentially a graph k-coloring problem, where $k = 2$ for DPL and $k = 3$ for TPL. It is well known that 3-coloring problem is NP-Complete, even when the graph is planar. For the general layout, ILP formulations are used in [1–3], and some heuristics are proposed in [5–7]. In reality, many industry designs are based on predesigned standard cells, where the layout is usually in row structures. All the cells are of exactly the same height, with power rails going from the left most of the cell to the right most of it. It is shown that multiple patterning decomposition ($k = 3$) for cell-based row structure layout is polynomial time solvable [4]. The following discussions are based on $k = 3$. The same concept can be easily extended to other multiple patterning techniques such as $k = 2$ or $k > 3$.

Problem: Multiple Patterning Decomposition

Using k colors to represent the k masks, multiple patterning decomposition can be defined as follows:

Input: *Circuit layout and a minimum coloring distance* $d_{\min}$.

Output: *A coloring solution where all features are assigned to one of the k colors.*

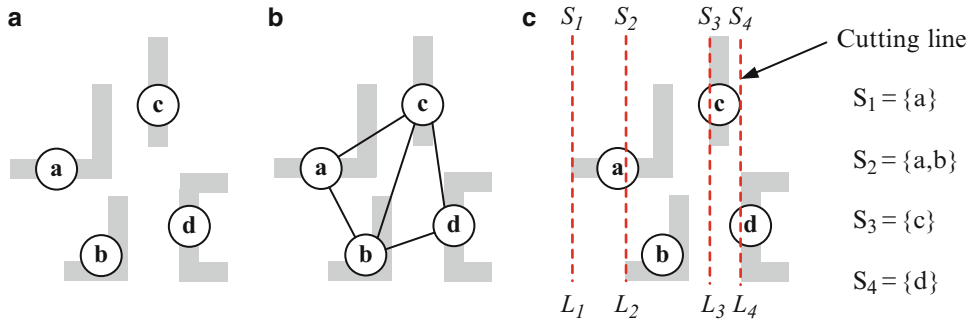
Constraint: *Any two features with the distance less than* $d_{\min}$ *cannot be assigned to the same color.*

Key Results

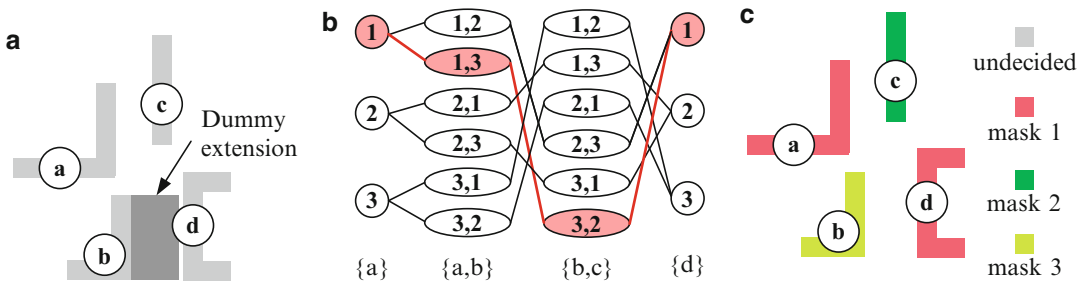
Multiple Patterning Decomposition for Standard Cell Designs

Given a layout, a conflict graph $G = (V, E)$ is constructed, where (I) vertices $V = \{v_1, \ldots, v_n\}$ represent the features in the layout and (II) $E = \{e_1, \ldots, e_m\}$ represent conflicting relationships between the features. A conflict edge exists if the distance of the two features is within $d_{\min}$. Imagine a cutting line that goes vertically across the cell, there are limited number of features that intersect with the cutting line due to the fixed height of the cell. Therefore, the coloring solutions of each cutting can be enumerated in polynomial time. The set of polygons that intersect with the same cutting line is called a cutting line set. An example of conflict graph, cutting line, and cutting line set is shown in Fig. 1.

By using the left boundary of each feature as the cutting line, the solutions of each cut line are computed. Solutions of adjacent cut lines are connected together, which leads to a solution graph. Polygon dummy extension is performed to ensure that the constructed solution graph is legal. For each polygon, its right boundary is



Layout Decomposition for Multiple Patterning, Fig. 1 (**a**) Input layout. (**b**) Conflict graph. (**c**) Cutting lines and the corresponding cutting line sets. There are four cutting lines L_1–L_4 and four cutting line sets S_1–S_4 in this layout



Layout Decomposition for Multiple Patterning, Fig. 2 (**a**) Input layout with polygon dummy extension. (**b**) Solution graph. The highlighted path is a sample decomposition. (**c**) Sample decomposition. Different colors represent different masks

virtually extended to its right most conflicting polygon. After extending the right boundaries of the polygons, it is guaranteed that for any polygon in a cutting line set, all its conflicting polygons (with smaller x coordinates) appear in the previous cutting line set. Therefore, the solution graph can be incrementally constructed and the correctness of the graph is guaranteed.

The solution graph is complete in the sense that it explores all the solution space. It is proven in [4] that every path in the solution graph corresponds to legal TPL decomposition and every legal TPL decomposition corresponds to a path in the solution graph. Figure 2 illustrates the overall flow of their approach.

Minimizing Stitches

The approach can be extended to handle stitches. All legal stitch candidates are computed for the layout, where a polygon feature is decomposed into a set of touching polygons by the stitch candidates. Conflict graph $G = (V, E)$ is constructed to model the rectangular layout, where (I) vertices $V = \{v_1, \ldots, v_n\}$ represent the features in the layout and (II) $E = \{e_1, \ldots, e_m\}$ represent different relationships between the features. There are two types of edges in the graph: conflict edges and stitch edges. A conflict edge exists if the two features do not touch each other and their distance is within $d_{\min}$. A stitch edge exists if the two features touch each other.

A weighted solution graph is constructed, where the weight of an edge denotes the number of stitches needed between the two vertices. A shortest path algorithm is utilized to get the decomposition with optimal number of stitches.

Multiple Patterning Coloring Constraint

In practice, there are additional coloring constraints such as balancing the usage of different masks [4,8] and assigning the same pattern for the same type of cells [9]. For standard cell designs,

coloring balancing can be simply achieved by using three global variables when parsing the solution graph [4]. An efficient SAT formulation with limited number of clauses is used to guarantee that the same type of cells has the same coloring decomposition [9].

Applications

Products using DPL in 22 nm technology node are already available in markets. TPL can be used in 14/10 nm technology node.

Open Problems

None is reported.

Experimental Results

The authors in [1] show that as the minimum coloring distance $d_{\min}$ increases, the number of unsolved conflicts increases. They also observe that the placement utilization has a very small impact on the number of unsolved conflicts. The results in [3] show that the speedup techniques can greatly reduce the overall runtime without adversely affecting the quality of the decomposition. Better results on the same benchmarks are reported in [5–7]. The authors in [4] show that their algorithm is able to solve all TPL decomposable benchmarks. For complex layout with stitch candidates, their approach computes a decomposition with the optimal number of stitches.

URLs to Code and Data Sets

The NanGate Open Cell Library can be obtained online for free.

Cross-References

▸ Graph Coloring
▸ Layout Decomposition for Triple Patterning

Recommended Reading

1. Kahng AB, Xu X, Park C-H, Yao H (2008) Layout decomposition for double patterning lithography. In: IEEE/ACM international conference on computer-aided design, San Jose
2. Yuan K, Yang J-S, Pan DZ (2010) Double patterning layout decomposition for simultaneous conflict and stitch minimization. IEEE Trans Comput-Aided Des Integr Circuits Syst 29:185–196
3. Yu B, Yuan K, Zhang B, Ding D, Pan DZ (2011) Layout decomposition for triple patterning lithography. In: IEEE/ACM international conference on computer-aided design, San Jose
4. Tian H, Zhang H, Ma Q, Xiao Z, Wong MDF (2012) A polynomial time triple patterning algorithm for cell based row-structure layout. In: IEEE/ACM international conference on computer-aided design, San Jose
5. Fang S-Y, Chang Y-W, Chen W-Y (2012) A novel layout decomposition algorithm for triple patterning lithography. In: IEEE/ACM proceedings of design automation conference, San Francisco
6. Kuang J, Yang EFY (2013) An efficient layout decomposition approach for triple patterning lithography. In: IEEE/ACM proceedings of design automation conference, Austin
7. Zhang Y, Luk W-S, Zhou H, Yan C, Zeng X (2013) Layout decomposition with pairwise coloring for multiple patterning lithography. In: IEEE/ACM international conference on computer-aided design, San Jose
8. Yu B, Lin Y-H, Luk-Pat G, Ding D, Lucas K, Pan DZ (2013) A high-performance triple patterning layout decomposer with balanced density. In: IEEE/ACM international conference on computer-aided design, San Jose
9. Tian H, Zhang H, Du Y, Xiao Z, Wong MDF (2013) Constrained pattern assignment for standard cell based triple patterning lithography. In: IEEE/ACM international conference on computer-aided design, San Jose

Layout Decomposition for Triple Patterning

Bei Yu and David Z. Pan
Department of Electrical and Computer Engineering, University of Texas, Austin, TX, USA

Keywords

Graph coloring; Layout decomposition; Lithography; Mathematical programming

Years and Authors of Summarized Original Work

2011; Yu, Yuan, Zhang, Ding, Pan

Problem Definition

Layout decomposition is a key stage in triple patterning lithography manufacturing process, where the original designed layout is divided into three masks. There will be three exposure/etching steps, through which the circuit layout can be produced. When the distance between two input features is less than certain minimum distance min_s, they need to be assigned to different masks (colors) to avoid coloring conflict. Sometimes coloring conflict can be resolved by splitting a pattern into two different masks. However, this introduces stitches, which lead to yield loss because of overlay error. Therefore, two of the main objectives in layout decomposition are conflict minimization and stitch minimization. An example of triple patterning layout decomposition is shown in Fig. 1, where all features are divided into three masks without any conflict and one stitch is introduced.

Given an input layout, a *conflict graph* is constructed to transfer initial geometrical relationship into an undirected graph with a set of vertices V and two sets of edges, which are the *conflict edges* (CE) and *stitch edges* (SE), respectively. V has one or more vertices for each polygonal shape and each vertex is associated with a polygonal shape. An edge is in CE iff the two corresponding vertices are within minimum coloring distance min_s. An edge is in SE iff there is a stitch candidate between the two vertices which are associated with the same polygonal shape.

Problem 1 (Layout Decomposition for Triple Patterning)

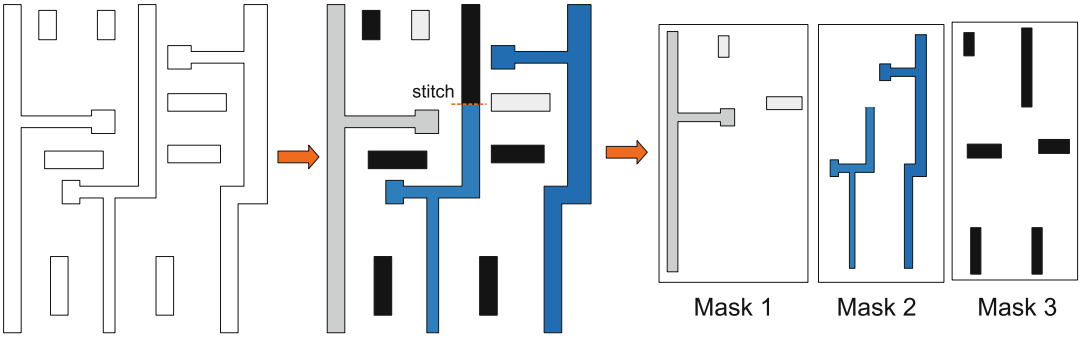
INPUT: *The decomposition graph where each vertex represents one polygonal shape, and all possible conflicts and stitches are in the conflict edge set CE and the stitch edge set SE, respectively.*

OUTPUT: *A three-color assignment to the conflict graph, such that the weighted cost of conflicts and stitches are minimized. The additional constraints may include color balancing, overlay control, and color preference.*

Key Results

Given an input layout, the conflict graph is constructed. Based on the conflict graph, the layout decomposition for triple patterning can be formulated as an integer linear programming (ILP) formulation [5]. As shown in (1), the objective function in the ILP formulation is to minimize the weighted cost function of conflict and stitch numbers simultaneously:

$$\min \sum_{e_{ij} \in CE} c_{ij} + \alpha \sum_{e_{ij} \in SE} s_{ij} \qquad (1)$$



Layout Decomposition for Triple Patterning, Fig. 1 Layout decomposition for triple patterning lithography (TPL)

where α is a parameter for assigning relative cost of stitch versus conflict. Typically, α is much smaller than 1, for example, 0.1, as resolving conflict is the most important objective during layout decomposition. Although the ILP formulation can solve the above layout decomposition problem optimally, it is not scalable to deal with large layouts in modern VLSI designs as the ILP problem is NP-complete.

In [5], a semidefinite programming (SDP)-based algorithm was proposed to achieve good runtime and solution quality. Instead of using a two binary variables to represent three masks, three unit vectors $(1,0), \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)$, and $\left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)$ are proposed to represent them. Note that the angle between any two vectors of the same color is 0, while the angle between any two vectors with different colors is $2\pi/3$. The inner product of two m-dimension vectors $\mathbf{v_i}$ and $\mathbf{v_j}$ is defined as $\mathbf{v_i} \cdot \mathbf{v_j} = \sum_k v_{ik} v_{jk}$. Then for any two vectors $\mathbf{v_i}, \mathbf{v_j} \in \left\{(1,0), \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right), \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\right\}$, the following property holds:

$$\mathbf{v_i} \cdot \mathbf{v_j} = \begin{cases} 1, & \mathbf{v_i} = \mathbf{v_j} \\ -\frac{1}{2}, & \mathbf{v_i} \neq \mathbf{v_j} \end{cases}$$

Based on the vector representation, the layout decomposition for triple patterning problem can be written as the following vector programming:

$$\min \sum_{e_{ij} \in CE} \frac{2}{3}\left(\mathbf{v_i} \cdot \mathbf{v_j} + \frac{1}{2}\right) + \frac{2\alpha}{3} \sum_{e_{ij} \in SE} \left(1 - \mathbf{v_i} \cdot \mathbf{v_j}\right) \tag{2}$$

$$\text{s.t. } \mathbf{v_i} \in \left\{(1,0), \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right), \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\right\} \tag{2a}$$

It shall be noted that $\mathbf{v_i}$ here is discrete, which is very expensive to solve. Then the discrete vector program is relaxed to the corresponding continuous formulation, which can be solved as a standard semidefinite programming (SDP), as shown below:

$$\min \sum_{e_{ij} \in CE} \frac{2}{3}\left(\mathbf{y_i} \cdot \mathbf{y_j} + \frac{1}{2}\right) + \frac{2\alpha}{3} \sum_{e_{ij} \in SE} \left(1 - \mathbf{y_i} \cdot \mathbf{y_j}\right) \tag{3}$$

$$\text{s.t. } \mathbf{y_i} \cdot \mathbf{y_i} = 1, \quad \forall i \in V \tag{3a}$$

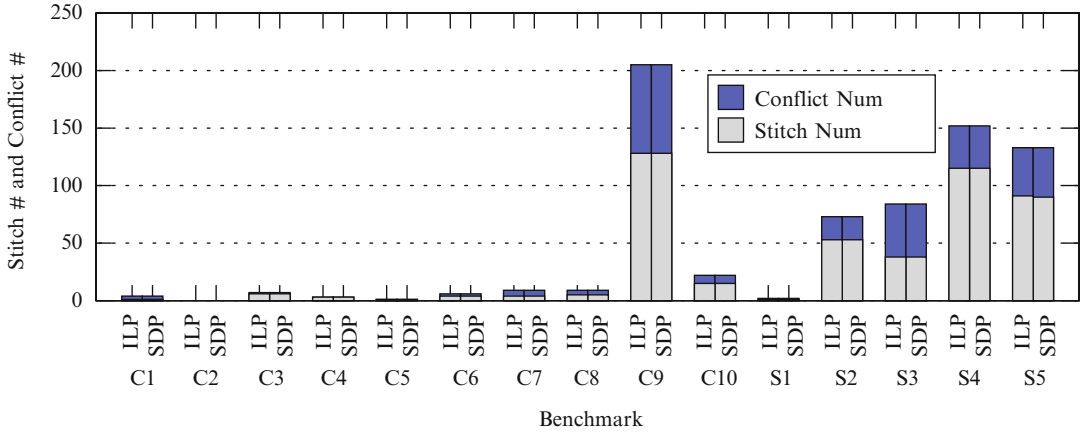
$$-\frac{1}{2} \leq \mathbf{y_i} \cdot \mathbf{y_j}, \quad \forall e_{ij} \in CE \tag{3b}$$

$$Y \succeq 0 \tag{3c}$$

The resulting matrix Y, where $y_{ij} = \mathbf{y_i} \cdot \mathbf{y_j}$, essentially provides the relative coloring guidance between two layout features (nodes in the conflict graph). It will be used to guide the final color assignment. If y_{ij} is close to 1, nodes i and j should be in the same mask; if y_{ij} is close to -0.5, nodes i and j tend to be in different masks. The results show that with reasonable threshold such as $0.9 < y_{ij} \leq 1$ for the same mask and $-0.5 \leq y_{ij} < -0.4$ for different masks, more than 80 % of nodes/polygons are decided by the global SDP. For the rest values, heuristic mapping algorithms will be performed to assign all vertices to their final colors.

A set of graph simplification techniques have been proposed to achieve speedup [1,2,5,8]. For example, one technique is called *iterative vertex removal*, where all vertices with degree less than or equal to two are detected and removed temporarily from the conflict graph. After each vertex removal, the degrees of other vertices would be updated. This removing process will continue until all the vertices have degree three or more. All the vertices that are temporarily removed are stored in a stack S. After the color assignment of the remained conflict graph is solved, the removed vertices in S are added back for coloring assignment. For row-based structure layout, specific graph-based algorithms are proposed to provide fast layout decomposition solutions [3,7].

Triple patterning layout decomposition has been actively studied in the last few years, with many interesting results reported. In [5], the performances between ILP- and SDP-based methods were compared. As shown in Fig. 2, SDP-based method can achieve the same optimal solutions



Layout Decomposition for Triple Patterning, Fig. 2 For ISCAS benchmark suite, the results of ILP- and SDP-based methods are very comparable

as obtained by ILP for 14 out of 15 test cases. However, the runtime of ILP-based algorithm is prohibitive when the problem size is big and the layout is dense. Graph simplification techniques are very effective to speed up the layout decomposition process as that can effectively reduce the ILP and SDP problem size. The coloring density balance was integrated into the SDP formulation in [6]. In [4], the SDP framework was further extended to handle quadruple patterning or more general multiple patterning lithography with new vector definition and linear runtime heuristic algorithms.

URLs to Code and Data Sets

Programs and benchmark suites can be found through http://www.cerc.utexas.edu/utda/download/MPLD/.

Cross-References

- Graph Coloring
- Greedy Approximation Algorithms

Recommended Reading

1. Fang SY, Chen WY, Chang YW (2012) A novel layout decomposition algorithm for triple patterning lithography. In: IEEE/ACM design automation conference (DAC), San Francisco, pp 1185–1190
2. Kuang J, Young EF (2013) An efficient layout decomposition approach for triple patterning lithography. In: IEEE/ACM design automation conference (DAC), Austin, pp 69:1–69:6
3. Tian H, Zhang H, Ma Q, Xiao Z, Wong M (2012) A polynomial time triple patterning algorithm for cell based row-structure layout. In: IEEE/ACM international conference on computer-aided design (ICCAD), San Jose, pp 57–64
4. Yu B, Pan DZ (2014) Layout decomposition for quadruple patterning lithography and beyond. In: IEEE/ACM design automation conference (DAC), San Francisco
5. Yu B, Yuan K, Zhang B, Ding D, Pan DZ (2011) Layout decomposition for triple patterning lithography. In: IEEE/ACM international conference on computer-aided design (ICCAD), San Jose, pp 1–8
6. Yu B, Lin YH, Luk-Pat G, Ding D, Lucas K, Pan DZ (2013) A high-performance triple patterning layout decomposer with balanced density. In: IEEE/ACM International conference on computer-aided design (ICCAD), San Jose, pp 163–169
7. Yu B, Xu X, Gao JR, Pan DZ (2013) Methodology for standard cell compliance and detailed placement for triple patterning lithography. In: IEEE/ACM international conference on computer-aided design (ICCAD), San Jose, pp 349–356
8. Zhang Y, Luk WS, Zhou H, Yan C, Zeng X (2013) Layout decomposition with pairwise coloring for multiple patterning lithography. In: IEEE/ACM international conference on computer-aided design (ICCAD), San Jose, pp 170–177

Learning Automata

Stefano Varricchio
Department of Computer Science, University of Roma, Rome, Italy

Keywords

Boolean formulae; Computational learning; Formal series; Machine learning; Multiplicity automata; Multivariate polynomials

Years and Authors of Summarized Original Work

2000; Beimel, Bergadano, Bshouty, Kushilevitz, Varricchio

Problem Definition

This problem is concerned with the learnability of *multiplicity automata* in Angluin's *exact learning model* and applications to the learnability of functions represented by small multiplicity automata.

The Learning Model

It is the *exact learning* model [2]: Let f be a *target* function. A learning algorithm may propose to an oracle, in each step, two kinds of queries: *membership queries* (MQ) and *equivalence queries* (EQ). In a MQ it may query for the value of the function f on a particular assignment z. The response to such a query is the value $f(z)$. (If f is Boolean, this is the standard membership query.) In an EQ it may propose to the oracle a hypothesis function h. If h is equivalent to f on all input assignments, then the answer to the query is YES and the learning algorithm succeeds and halts. Otherwise, the answer to the equivalence query is NO and the algorithm receives a *counterexample*, i.e., an assignment z such that $f(z) \neq h(z)$. One says that the learner *learns* a class of functions $\mathcal{C}$, if for every function $f \in \mathcal{C}$ the learner outputs a hypothesis h that is equivalent to f and does so in time polynomial in the "size" of a shortest representation of f and the length of the longest counterexample. The exact learning model is strictly related to the *Probably Approximately Correct* (PAC) model of Valiant [19]. In fact, every equivalence query can be easily simulated by a sample of random examples. Therefore, learnability in the exact learning model also implies learnability in the PAC model with membership queries [2, 19].

Multiplicity Automata

Let $\mathcal{K}$ be a field, Σ be an alphabet, and ϵ be the empty string. A *multiplicity automaton* (MA) A of size r consists of $|\Sigma|$ matrices $\{\mu_\sigma : \sigma \in \Sigma\}$, each of which is an $r \times r$ matrix of elements from $\mathcal{K}$ and an r-tuple $\vec{\gamma} = (\gamma_1, \ldots, \gamma_r) \in \mathcal{K}^r$. The automaton A defines a function $f_A : \Sigma^* \to \mathcal{K}$ as follows. First, define a mapping μ, which associates with every string in Σ^* an $r \times r$ matrix over $\mathcal{K}$, by $\mu(\epsilon) \triangleq \mathrm{ID}$ where ID denotes the *identity matrix*, and for a string $w = \sigma_1\sigma_2 \ldots \sigma_n$, let $\mu(\omega) \triangleq \mu_{\sigma_1} \cdot \mu_{\sigma_2} \ldots \mu_{\sigma_n}$. A simple property of μ is that $\mu(x \circ y) = \mu(x) \cdot \mu(y)$, where $\circ$ denotes concatenation. Now, $f_A(\omega) \triangleq [\mu(\omega)]1 \cdot \vec{\gamma}$ (where $[\mu(w)]_i$ denotes the ith row of the matrix $\mu(w)$). Let $f : \Sigma^* \to \mathcal{K}$ be a function. Associate with f an infinite matrix F, where each of its rows is indexed by a string $x \in \Sigma^*$ and each of its columns is indexed by a string $y \in \Sigma^*$. The (x, y) entry of F contains the value $f(x \circ y)$. The matrix F is called the *Hankel Matrix* of f. The xth row of F is denoted by F_x. The (x, y) entry of F may be therefore denoted as $F_x(y)$ and as $F_{x,y}$. The following result relates the size of the minimal MA for f to the rank of F (cf. [4] and references therein).

Theorem 1 *Let $f : \Sigma^* \to \mathcal{K}$ such that $f \not\equiv 0$ and let F be its Hankel matrix. Then, the size r of the smallest multiplicity automaton A such that $f_A \equiv f$ satisfies $r = \mathrm{rank}(F)$ (over the field $\mathcal{K}$).*

Key Results

The learnability of multiplicity automata has been proved in [7] and, independently, in [17]. In what follows, let $\mathcal{K}$ be a field, $f : \Sigma^* \to \mathcal{K}$

Stefano Varricchio: deceased

be a function, and F its Hankel matrix such that $r = \text{rank}(F)$ (over $\mathcal{K}$).

Theorem 2 ([4]) *The function f is learnable by an algorithm in time $O(|\Sigma| \cdot r \cdot \text{M}(r) + m \cdot r^3)$ using r equivalence queries and $O((|\Sigma| + \log m) r^2)$ membership queries, where m is the size of the longest counterexample obtained during the execution of the algorithm and $\text{M}(r)$ is the complexity of multiplying two $r \times r$ matrices.*

Some extensions of the above result can be found in [8, 13, 16]. In many cases of interest, the domain of the target function f is not Σ^* but rather Σ^n for some value n, i.e., $f : \Sigma^n \to \mathcal{K}$. The length of counterexamples, in this case, is always n and so $m = n$. Denote by F^d the submatrix of F whose rows are strings in Σ^d and whose columns are strings in Σ^{n-d} and let $r_{\max} = \max_{d=0}^{n} \text{rank}(F^d)$ (where rank is taken over $\mathcal{K}$).

Theorem 3 ([4]) *The function f is learnable by an algorithm in time $O(|\Sigma| rn \cdot M(r_{\max}))$ using $O(r)$ equivalence queries and $O((|\Sigma| + \log n) r \cdot r_{\max})$ membership queries.*

The time complexity of the two above results has been recently further improved [9].

Applications

The results of this section can be found in [3–6]. They show the learnability of various classes of functions as a consequence of Theorems 2 and 3. This can be done by proving that for every function f in the class in question, the corresponding Hankel matrix F has low rank. As is well known, any nondeterministic automaton can be regarded as a multiplicity automaton, whose associated function returns the number of accepting paths of the nondeterministic automaton on w. Therefore, the learnability of multiplicity automata gives a new algorithm for learning deterministic automata and unambiguous automata. (A nondeterministic automata is *unambiguous* if for every $w \in \Sigma^*$, there is at most one accepting path.) The learnability of deterministic automata has been proved in [1]. By [14], the class of deterministic automata contains the class of $O(\log n)$-term DNF, i.e., DNF formulae over n Boolean variables with $O(\log n)$ number of terms. Hence, this class can be learned using multiplicity automata.

Classes of Polynomials

Theorem 4 *Let $p_{i,j} : \Sigma \to \mathcal{K}$ be arbitrary functions of a single variable ($1 \le i \le t$, $1 \le j \le n$). Let $g_i : \Sigma^n \to \mathcal{K}$ be defined by $\prod_{j=1}^{n} p_{i,j}(z_j)$. Finally, let $f : \Sigma^n \to \mathcal{K}$ be defined by $f = \sum_{i=1}^{t} g_i$. Let F be the Hankel matrix corresponding to f and F^d the submatrices defined in the previous section. Then, for every $0 \le d \le n$, $\text{rank}(F^d) \le t$.*

Corollary 1 *The class of functions that can be expressed as functions over $\text{GF}(p)$ with t summands, where each summand T_i is a product of the form $p_{i,1}(x_1) \cdots p_{i,n}(x_n)$ (and $p_{i,j} : \text{GF}(p) \to \text{GF}(p)$ are arbitrary functions), is learnable in time $\text{poly}(n, t, p)$.*

The above corollary implies as a special case the learnability of polynomials over $\text{GF}(p)$. This extends the result of [18] from multi-linear polynomials to arbitrary polynomials. The algorithm of Theorem 3, for polynomials with n variables and t terms, uses $O(nt)$ equivalence queries and $O(t^2 n \log n)$ membership queries. The special case of the above class – the class of polynomials over GF(2) – was known to be learnable before [18]. Their algorithm uses $O(nt)$ equivalence queries and $O(t^3 n)$ membership queries. The following theorem extends the latter result to *infinite* fields, assuming that the functions $p_{i,j}$ are bounded-degree polynomials.

Theorem 5 *The class of functions over a field $\mathcal{K}$ that can be expressed as t summands, where each summand T_i is of the form $p_{i,1}(x_1) \cdots p_{i,n}(x_n)$ and $p_{i,j} : \mathcal{K} \to \mathcal{K}$ are univariate polynomials of degree at most k, is learnable in time $\text{poly}(n, t, k)$. Furthermore, if $|\mathcal{K}| \ge nk + 1$, then this class is learnable from membership queries only in time $\text{poly}(n, t, k)$ (with small probability of error).*

Classes of Boxes

Let $[\ell]$ denote the set $\{0, 1, \ldots, \ell - 1\}$. A box in $[\ell]^n$ is defined by two corners $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ (in $[\ell]^n$) as follows:

$$B_{a_1,\ldots,a_n,b_1,\ldots,b_n} = \{(x_1, \ldots, x_n) : \forall i,\ a_i \leq x_i \leq b_i\}.$$

A box can be represented by its characteristic function in $[\ell]^n$. The following result concerns a more general class of functions.

Theorem 6 *Let $p_{i,j} : \Sigma \to \{0, 1\}$ be arbitrary functions of a single variable $(1 \leq i \leq t,\ 1 \leq j \leq n)$. Let $g_i : \Sigma^n \to \{0, 1\}$ be defined by $\prod_{j=1}^{n} p_{i,j}(z_j)$. Assume that there is no point $x \in \Sigma^n$ such that $g_i(x) = 1$ for more than s functions g_i. Finally, let $f : \Sigma^n \to \{0, 1\}$ be defined by $f = \bigvee_{i=1}^{t} g_i$. Let F be the Hankel matrix corresponding to f. Then, for every field $\mathcal{K}$ and for every $0 \leq d \leq n$, $\operatorname{rank}(F^d) \leq \sum_{i=1}^{s} \binom{t}{i}$.*

Corollary 2 *The class of unions of disjoint boxes can be learned in time $\operatorname{poly}(n, t, \ell)$ (where t is the number of boxes in the target function). The class of unions of $O(\log n)$ boxes can be learned in time $\operatorname{poly}(n, \ell)$.*

Classes of DNF Formulae

The learnability of DNF formulae has been widely investigated. The following special case of Corollary 1 solves an open problem of [18]:

Corollary 3 *The class of functions that can be expressed as exclusive OR of t (not necessarily monotone) monomials is learnable in time $\operatorname{poly}(n, t)$.*

While Corollary 3 does not refer to a subclass of DNF, it already implies the learnability of disjoint (i.e., satisfy-1) DNF. Since DNF is a special case of union of boxes (with $\ell = 2$), one obtains also the learnability of disjoint DNF from Corollary 2. Positive results for satisfy-s DNF (i.e., DNF formulae in which each assignment satisfies at most s terms) can be obtained, with larger values of s. The following two important corollaries follow from Theorem 6. Note that Theorem 6 holds in any field. For convenience (and efficiency), let $\mathcal{K} = \mathrm{GF}(2)$.

Theorem 7 *Let $f = T_1 \vee T_2 \vee \ldots \vee T_t$ be a satisfy-s DNF (i.e., each T_i is a monomial). Let F be the Hankel matrix corresponding to f. Then, $\operatorname{rank}(F^d) \leq \sum_{i=1}^{s} \binom{t}{i} \leq t^s$.*

Corollary 4 *The class of satisfy-s DNF formulae, for $s = O(1)$, is learnable in time $\operatorname{poly}(n, t)$.*

Corollary 5 *The class of satisfy-s, t-term DNF formulae is learnable in time $\operatorname{poly}(n)$ for the following choices of s and t : (1) $t = O(\log n)$, (2) $t = \operatorname{poly}\log(n)$ and $s = O(\log n / \log \log n)$, (3) $t = 2^{O(\log n / \log \log n)}$ and $s = O(\log \log n)$.*

Classes of Decision Trees

The algorithm of Theorem 3 efficiently learns the class of disjoint DNF formulae. This includes the class of decision trees. Therefore, decision trees of size t on n variables are learnable using $O(tn)$ equivalence queries and $O(t^2 n \log n)$ membership queries. This is better than the best known algorithm for decision trees [11] (which uses $O(t^2)$ equivalence queries and $O(t^2 n^2)$ membership queries). The following results concern more general classes of decision trees.

Corollary 6 *Consider the class of decision trees that compute functions $f : \mathrm{GF}(p)^n \to \mathrm{GF}(p)$ as follows: each node v contains a query of the form "$x_i \in S_v$?" for some $S_v \subseteq \mathrm{GF}(p)$. If $x_i \in S_v$, then the computation proceeds to the left child of v, and if $x_i \notin S_v$ the computation proceeds to the right child. Each leaf ℓ of the tree is marked by a value $\gamma\ell \in \mathrm{GF}(p)$ which is the output on all the assignments which reach this leaf. Then, this class is learnable in time $\operatorname{poly}(n, |L|, p)$, where L is the set of leaves.*

The above result implies the learnability of decision trees with "greater-than" queries in the nodes, solving a problem of [11]. Every decision tree with "greater-than" queries that computes a Boolean function can be expressed as the union of disjoint boxes. Hence, this case can also be

derived from Corollary 2. The next theorem will be used to learn more classes of decision trees.

Theorem 8 *Let* $g_i : \Sigma^n \to \mathcal{K}$ *be arbitrary functions* $(1 \le i \le \ell)$. *Let* $f : \Sigma^n \to \mathcal{K}$ *be defined by* $f = \prod_{i=1}^{\ell} g_i$. *Let* F *be the Hankel matrix corresponding to* f, *and* G_i *be the Hankel matrix corresponding to* g_i. *Then,* $\text{rank}(F^d) \le \prod_{i=1}^{\ell} \text{rank}(G_i^d)$.

This theorem has some interesting applications. The first application states that arithmetic circuits of depth two with multiplication gate of fan-in $O(\log n)$ at the top level and addition gates with unbounded fan-in in the bottom level are learnable.

Corollary 7 *Let* $\mathcal{C}$ *be the class of functions that can be expressed in the following way: Let* $p_{i,j} : \Sigma \to \mathcal{K}$ *be arbitrary functions of a single variable* $(1 \le i \le \ell,\ 1 \le j \le n)$. *Let* $\ell = O(\log n)$ *and* $g_i : \Sigma^n \to \mathcal{K} (1 \le i \le \ell)$ *be defined by* $\Sigma_{j=1}^{n} p_{i,j}(z_j)$. *Finally, let* $f : \Sigma^n \to \mathcal{K}$ *be defined by* $f = \prod_{i=1}^{\ell} g_i$. *Then,* $\mathcal{C}$ *is learnable in time* $\text{poly}(n, |\Sigma|)$.

Corollary 8 *Consider the class of decision trees of depth* s, *where the query at each node* v *is a Boolean function* f_v *with* $r_{\max} \le t$ *(as defined in section "Key Results") such that* $(t+1)^s = \text{poly}(n)$. *Then, this class is learnable in time* $\text{poly}(n, |\Sigma|)$.

The above class contains, for example, all the decision trees of depth $O(\log n)$ that contain in each node a term or a XOR of a subset of variables as defined in [15] (in this case $r_{\max} \le 2$).

Negative Results

In [4] some limitation of the learnability via the automaton representation has been proved. One can show that the main algorithm does not efficiently learn several important classes of functions. More precisely, these classes contain functions f that have no "small" automaton, i.e., by Theorem 1, the corresponding Hankel matrix F is "large" over every field $\mathcal{K}$.

Theorem 9 *The following classes are not learnable in time polynomial in* n *and the formula size using multiplicity automata (over any field* $\mathcal{K}$*): DNF, monotone DNF; 2-DNF; read-once DNF;* k*-term DNF, for* $k = \omega(\log n)$*; satisfy-*s *DNF, for* $s = \omega(1)$*; and read-*j *satisfy-*s *DNF, for* $j = \omega(1)$ *and* $s = \Omega(\log n)$.

Some of these classes are known to be learnable by other methods, some are natural generalizations of classes known to be learnable as automata ($O(\log n)$-term DNF [11, 12, 14], and satisfy-s DNF for $s = O(1)$ (Corollary 4)) or by other methods (read-j satisfy-s for $js = O(\log n / \log \log n)$ [10]), and the learnability of some of the others is still an open problem.

Cross-References

- Learning Constant-Depth Circuits
- Learning DNF Formulas

Recommended Reading

1. Angluin D (1987) Learning regular sets from queries and counterexamples. Inf Comput 75: 87–106
2. Angluin D (1988) Queries and concept learning. Mach Learn 2(4):319–342
3. Beimel A, Bergadano F, Bshouty NH, Kushilevitz E, Varricchio S (1996) On the applications of multiplicity automata in learning. In: Proceedings of the 37th annual IEEE symposium on foundations of computer science, Burlington, Vermont, USA. IEEE Computer Society, Los Alamitos, pp 349–358
4. Beimel A, Bergadano F, Bshouty NH, Kushilevitz E, Varricchio S (2000) Learning functions represented as multiplicity automata. J ACM 47: 506–530
5. Beimel A, Kushilevitz E (1997) Learning boxes in high dimension. In: Ben-David S (ed) 3rd European conference on computational learning theory (EuroCOLT '97), Jerusalem, Israel. Lecture notes in artificial intelligence, vol 1208. Springer, Berlin, pp 3–15. Journal version: Algorithmica 22:76–90 (1998)
6. Bergadano F, Catalano D, Varricchio S (1996) Learning sat-k-DNF formulas from membership queries. In: Proceedings of the 28th annual ACM symposium on the theory of computing, Philadelphia, Pennsylvania, USA. ACM, New York, pp 126–130

7. Bergadano F, Varricchio S (1994) Learning behaviors of automata from multiplicity and equivalence queries. In: Proceedings of 2nd Italian conference on algorithms and complexity, Rome, Italy. Lecture notes in computer science, vol 778. Springer, Berlin, pp 54–62. Journal version: SIAM J Comput 25(6):1268–1280 (1996)
8. Bergadano F, Varricchio S (1996) Learning behaviors of automata from shortest counterexamples. In: EuroCOLT '95, Barcelona, Spain. Lecture notes in artificial intelligence, vol 904. Springer, Berlin, pp 380–391
9. Bisht L, Bshouty NH, Mazzawi H (2006) On optimal learning algorithms for multiplicity automata. In: Proceedings of 19th annual ACM conference on computational learning theory, Pittsburgh, Pennsylvania, USA. Lecture notes in computer science, vol 4005. Springer, Berlin, pp 184–198
10. Blum A, Khardon R, Kushilevitz E, Pitt L, Roth D (1994) On learning read-k-satisfy-j DNF. In: Proceedings of the 7th annual ACM conference on computational learning theory, New Brunswick, New Jersey, USA. ACM, New York, pp 110–117
11. Bshouty NH (1993) Exact learning via the monotone theory. In: Proceedings of the 34th annual IEEE symposium on foundations of computer science, Palo Alto, California, USA. IEEE Computer Society, Los Alamitos, pp 302–311. Journal version: Inf Comput 123(1):146–153 (1995)
12. Bshouty NH (1995) Simple learning algorithms using divide and conquer. In: Proceedings of 8th annual ACM conference on computational learning theory, Santa Cruz, California, USA. ACM, New York, pp 447–453. Journal version: Comput Complex 6:174–194 (1997)
13. Bshouty NH, Tamon C, Wilson DK (1998) Learning matrix functions over rings. Algorithmica 22(1/2):91–111
14. Kushilevitz E (1996) A simple algorithm for learning $O(\log n)$-term DNF. In: Proceedings of 9th annual ACM conference on computational learning theory, Desenzano del Garda, Italy. ACM, New York, pp 266–269. Journal version: Inf Process Lett 61(6):289–292 (1997)
15. Kushilevitz E, Mansour Y (1993) Learning decision trees using the fourier spectrum. SIAM J Comput 22(6):1331–1348
16. Melideo G, Varricchio S (1998) Learning unary output two-tape automata from multiplicity and equivalence queries. In: ALT '98, Otzenhausen, Germany. Lecture notes in computer science, vol 1501. Springer, Berlin, pp 87–102
17. Ohnishi H, Seki H, Kasami T (1994) A polynomial time learning algorithm for recognizable series. IEICE Trans Inf Syst E77-D(10)(5):1077–1085
18. Schapire RE, Sellie LM (1996) Learning sparse multivariate polynomials over a field with queries and counterexamples. J Comput Syst Sci 52(2):201–213
19. Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142

Learning Constant-Depth Circuits

Rocco A. Servedio
Computer Science, Columbia University, New York, NY, USA

Keywords

Boolean Fourier analysis; Learning AC^0 circuits; PAC learning; Uniform-distribution learning

Years and Authors of Summarized Original Work

1993; Linial, Mansour, Nisan

Problem Definition

This problem deals with learning "simple" Boolean functions $f : \{0,1\}^n \to \{-1,1\}$ from uniform random labeled examples. In the basic uniform-distribution PAC framework, the learning algorithm is given access to a *uniform random example oracle* $EX(f,U)$ which, when queried, provides a labeled random example $(x, f(x))$ where x is drawn from the uniform distribution U over the Boolean cube $\{0,1\}^n$. Successive calls to the $EX(f,U)$ oracle yield independent uniform random examples. The goal of the learning algorithm is to output a representation of a hypothesis function $h : \{0,1\}^n \to \{-1,1\}$ which with high probability has high accuracy; formally, for any $\epsilon, \delta > 0$, given ϵ and δ the learning algorithm should output an h which with probability at least $1-\delta$ has $\Pr_{x \in U}[h(x) \neq f(x)] \leq \epsilon$.

Many variants of the basic framework described above have been considered. In the *distribution-independent* PAC learning model, the random example oracle is $EX(f,\mathcal{D})$ where $\mathcal{D}$ is an arbitrary (and unknown to the learner) distribution over $\{0,1\}^n$; the hypothesis h should now have high accuracy with respect to $\mathcal{D}$, i.e., with probability $1-\delta$, it must satisfy

$\Pr_{x\in\mathcal{D}}[h(x) \neq f(x)] \leq \epsilon$. Another variant that has been considered is when the distribution $\mathcal{D}$ is assumed to be an unknown *product distribution*; such a distribution is defined by n parameters $0 \leq p_1, \ldots, p_n \leq 1$, and a draw from $\mathcal{D}$ is obtained by independently setting each bit x_i to 1 with probability p_i. Yet another variant is to consider learning with the help of a *membership oracle*: this is a "black box" oracle $MQ(f)$ for f which, when queried on an input $x \in \{0,1\}^n$, returns the value of $f(x)$. The model of uniform-distribution learning with a membership oracle has been well studied; see e.g. [7, 15].

There are many ways to make precise the notion of a "simple" Boolean function; one common approach is to stipulate that the function be computed by a Boolean circuit of some restricted form. A circuit of *size s and depth d* consists of s AND and OR gates (of unbounded fanin) in which the longest path from any input literal $x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n$ to the output node is of length d. Note that a circuit of size s and depth 2 is simply a CNF formula or DNF formula. The complexity class consisting of those Boolean functions computed by poly(n)-size, $O(1)$-depth circuits is known as *nonuniform* AC^0.

Key Results

Positive Results

Linial et al. [16] showed that almost all of the Fourier weight of any constant-depth circuit is on "low-order" Fourier coefficients:

Lemma 1 *Let $f : \{0,1\}^n \to \{-1,1\}$ be a Boolean function that is computed by an AND/OR/NOT circuit of size s and depth d. Then for any integer $t \geq 0$,*

$$\sum_{S\subset\{1,\ldots,n\},|S|>t} \hat{f}(S)^2 \leq 2s2^{-t^{1/d}/20}.$$

(Hastad [6] has given a refined version of Lemma 1 with slightly sharper bounds; see also [21] for a streamlined proof.) They also showed that any Boolean function can be well approximated by approximating its Fourier spectrum.

Lemma 2 *Let $f : \{0,1\}^n \to \{-1,1\}$ be any Boolean Function and let $g : \{0,1\}^n \to \mathbf{R}$ be an arbitrary function such that $\sum_{S\subseteq\{1,\ldots,n\}}(\hat{f}(S) - \hat{g}(S))^2 \leq \epsilon$. Then $\Pr_{x\in U}[f(x) \neq \text{sign}(g(x))] \leq \epsilon$.*

Using the above two results together with a procedure that estimates all the "low-order" Fourier coefficients, they obtained a quasipolynomial-time algorithm for learning constant-depth circuits:

Theorem 1 *There is an $n^{O((\log^d(n/\epsilon)))}$-time algorithm that learns any poly(n)-size, depth-d Boolean AND/OR/NOT circuit to accuracy ϵ with respect to the uniform distribution, using uniform random examples only.*

Furst et al. [3] extended this result to learning under constant-bounded product distributions. A product distribution $\mathcal{D}$ is said to be *constant bounded* if each of its n parameters $p_1, \ldots, p_n$ is bounded away from 0 and 1, i.e., satisfies $\min\{p_i, 1-p_i\} = \Theta(1)$.

Theorem 2 *There is an $n^{O((\log^d(n/\epsilon)))}$-time algorithm that learns any poly(n)-size, depth-d Boolean AND/OR/NOT circuit to accuracy ϵ given random examples drawn from any constant-bounded product distribution.*

By combining the Fourier arguments of Linial et al. with hypothesis boosting, Jackson et al. [8] were able to extend Theorem 1 to a broader class of circuits, namely, constant-depth AND/OR/NOT circuits that additionally contain (a limited number of) *majority gates*. A majority gate over r Boolean inputs is a binary gate which outputs "true" if and only if at least half of its r Boolean inputs are set to "true."

Theorem 3 *There is an $n^{\log^{O(1)}(n/\epsilon)}$-time algorithm that learns any poly(n)-size, constant-depth Boolean AND/OR/NOT circuit that additionally contains polylog(n) many majority gates to accuracy ϵ with respect to the uniform distribution, using uniform random examples only.*

A *threshold gate* over r Boolean inputs is a binary gate defined by r real weights $w_1, \ldots, w_r$ and a real threshold θ; on input $(x_1, \ldots, x_r)$

the value of the threshold gate is 1 if and only if $w_1x_1 + \cdots + w_rx_r \geq \theta$. Gopalan and Servedio [4] observed that a conjecture of Gotsman and Linial [5] bounding the average sensitivity of low-degree polynomial threshold functions implies Fourier concentration – and hence quasipolynomial time learnability using the original Linial et al. [16] algorithm – for Boolean functions computed by polynomial-size constant-depth AND/OR/NOT circuits augmented with a threshold gate as the topmost (output) gate. Combining this observation with upper bounds on the noise sensitivity of low-degree polynomial threshold functions given in [2], they obtained unconditional sub-exponential time learning for these circuits. Subsequent improvements of these noise sensitivity results, nearly resolving the Gotsman-Linial conjecture and giving stronger unconditional running times for learning these circuits, were given by Kane [10]; see [2, 4, 10] for detailed statements of these results.

Negative Results

Kharitonov [11] showed that under a strong but plausible cryptographic assumption, the algorithmic result of Theorem 1 is essentially optimal. A *Blum integer* is an integer $N = P \cdot Q$ where both P and Q are congruent to 3 modulo 4. Kharitonov proved that if the problem of factoring a randomly chosen n-bit Blum integer is 2^{n^ϵ}-hard for some fixed $\epsilon > 0$, then any algorithm that (even weakly) learns polynomial-size depth-d circuits must run in time $2^{\log^{\Omega(d)} n}$, even if it is only required to learn under the uniform distribution and can use a membership oracle. This implies that there is no polynomial-time algorithm for learning polynomial-size, depth-d circuits (for d larger than some absolute constant).

Using a cryptographic construction of Naor and Reingold [18], Jackson et al. [8] proved a related result for circuits with majority gates. They showed that under Kharitonov's assumption, any algorithm that (even weakly) learns depth-5 circuits consisting of $\log^k n$ many majority gates must run in time $2^{\log^{\Omega(k)} n}$ time, even if it is only required to learn under the uniform distribution and can use a membership oracle.

Applications

The technique of learning by approximating most of the Fourier spectrum (Lemma 2 above) has found many applications in subsequent work on uniform-distribution learning. It is a crucial ingredient in the current state-of-the-art algorithms for learning monotone DNF formulas [20], monotone decision trees [19], and intersections of half-spaces [12] from uniform random examples only. Combined with a membership oracle-based procedure for identifying large Fourier coefficients, this technique is at the heart of an algorithm for learning decision trees [15]; this algorithm in turn plays a crucial role in the celebrated polynomial-time algorithm of Jackson [7] for learning polynomial-size depth-2 circuits under the uniform distribution.

The ideas of Linial et al. have also been applied for the difficult problem of *agnostic learning*. In the agnostic learning framework, there is a joint distribution $\mathcal{D}$ over example-label pairs $\{0,1\}^n \times \{-1,1\}$; the goal of an agnostic learning algorithm for a class $\mathcal{C}$ of functions is to construct a hypothesis h such that $\Pr_{(x,y)\in\mathcal{D}}[h(x) \neq y] \leq \min_{f\in\mathcal{C}} \Pr_{(x,y)\in\mathcal{D}}[f(x) \neq y] + \epsilon$. Kalai et al. [9] gave agnostic learning algorithms for half-spaces and related classes via an algorithm which may be viewed as a generalization of Linial et al.'s algorithm to a broader class of distributions.

Finally, there has been some applied work on learning using Fourier representations as well [17].

Open Problems

Perhaps the most outstanding open question related to this work is whether polynomial-size circuits of depth two – i.e., DNF formulas – can be learned in polynomial time from uniform random examples only. Blum [1] has offered a cash prize for a solution to a restricted version

of this problem. A hardness result for learning DNF would also be of great interest; recent work of Klivans and Sherstov [14] gives a hardness result for learning ANDs of majority gates, but hardness for DNF (ANDs of ORs) remains an open question.

Another open question is whether the quasipolynomial-time algorithms for learning constant-depth circuits under uniform distributions and product distributions can be extended to the general distribution-independent model. Known results in complexity theory imply that quasipolynomial-time distribution-independent learning algorithms for constant-depth circuits would follow from the existence of efficient linear threshold learning algorithms with a sufficiently high level of tolerance to "malicious" noise. Currently no nontrivial distribution-independent algorithms are known for learning circuits of depth 3; for depth-2 circuits the best known running time in the distribution-independent setting is the $2^{\tilde{O}(n^{1/3})}$-time algorithm of Klivans and Servedio [13].

A third direction for future work is to extend the results of [8] to a broader class of circuits. Can constant-depth circuits augmented with MOD_p gates be learned in quasipolynomial time? Jackson et al. [8] discusses the limitations of current techniques to address these extensions.

Experimental Results

None to report.

Data Sets

None to report.

URL to Code

None to report.

Cross-References

- Cryptographic Hardness of Learning
- Learning DNF Formulas
- PAC Learning
- Statistical Query Learning

Recommended Reading

1. Blum A (2003) Learning a function of r relevant variables (open problem). In: Proceedings of the 16th annual COLT, Washington, DC, pp 731–733
2. Diakonikolas I, Harsha P, Klivans A, Meka R, Raghavendra P, Servedio RA, Tan L-Y (2010) Bounding the average sensitivity and noise sensitivity of polynomial threshold functions. In: Proceedings of the 42nd annual ACM symposium on theory of computing (STOC), Cambridge, pp 533–542
3. Furst M, Jackson J, Smith S (1991) Improved learning of AC^0 functions. In: Proceedings of the 4th annual conference on learning theory (COLT), Santa Cruz, pp 317–325
4. Gopalan P, Servedio R (2010) Learning and lower bounds for AC^0 with threshold gates. In: Proceedings of the 14th international workshop on randomization and computation (RANDOM), Barcelona, pp 588–601
5. Gotsman C, Linial N (1994) Spectral properties of threshold functions. Combinatorica 14(1): 35–50
6. Håstad J (2001) A slight sharpening of LMN. J Comput Syst Sci 63(3):498–508
7. Jackson J (1997) An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J Comput Syst Sci 55: 414–440
8. Jackson J, Klivans A, Servedio R (2002) Learnability beyond AC^0. In: Proceedings of the 34th annual ACM symposium on theory of computing (STOC), Montréal, pp 776–784
9. Kalai A, Klivans A, Mansour Y, Servedio R (2005) Agnostically learning halfspaces. In: Proceedings of the 46th IEEE symposium on foundations of computer science (FOCS), Pittsburgh, pp 11–20
10. Kane DM (2014) The correct exponent for the Gotsman-Linial conjecture. Comput Complex 23:151–175
11. Kharitonov M (1993) Cryptographic hardness of distribution-specific learning. In: Proceedings of the twenty-fifth annual symposium on theory of computing (STOC), San Diego, pp 372–381
12. Klivans A, O'Donnell R, Servedio R (2004) Learning intersections and thresholds of halfspaces. J Comput Syst Sci 68(4):808–840

13. Klivans A, Servedio R (2004) Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. J Comput Syst Sci 68(2):303–318
14. Klivans A, Sherstov A (2006) Cryptographic hardness results for learning intersections of halfspaces. In: Proceedings of the 46th IEEE symposium on foundations of computer science (FOCS), Berkeley
15. Kushilevitz E, Mansour Y (1993) Learning decision trees using the Fourier spectrum. SIAM J Comput 22(6):1331–1348
16. Linial N, Mansour Y, Nisan N (1993) Constant depth circuits, Fourier transform and learnability. J ACM 40(3):607–620
17. Mansour Y, Sahar S (2000) Implementation issues in the Fourier transform algorithm. Mach Learn 40(1):5–33
18. Naor M, Reingold O (1997) Number-theoretic constructions of efficient pseudo-random functions. In: Proceedings of the thirty-eighth annual symposium on foundations of computer science (FOCS), Miami Beach, pp 458–467
19. O'Donnell R, Servedio R (2006) Learning monotone decision trees in polynomial time. In: Proceedings of the 21st conference on computational complexity (CCC), Prague, pp 213–225
20. Servedio R (2004) On learning monotone DNF under product distributions. Inf Comput 193(1):57–74
21. Stefankovic D (2002) Fourier transforms in computer science. PhD thesis, University of Chicago. Masters thesis, TR-2002-03

Learning DNF Formulas

Jeffrey C. Jackson
Department of Mathematics and Computer Science, Duquesne University, Pittsburgh, PA, USA

Keywords

Disjunctive normal form; DNF; Membership oracle; PAC learning; Random walk learning; Smoothed analysis

Years and Authors of Summarized Original Work

1997; Jackson
2005; Bshouty, Mossel, O'Donnell, Servedio
2009; Kalai, Samorodnitsky, Teng

Problem Definition

A disjunctive normal form (DNF) expression is a Boolean expression written as a disjunction of *terms*, where each term is the conjunction of Boolean variables that may or may not be negated. For example, $(v_1 \wedge \overline{v_2}) \vee (v_2 \wedge v_3)$ is a two-term DNF expression over three variables. DNF expressions occur frequently in digital circuit design, where DNF is often referred to as sum of products notation. From a learning perspective, DNF expressions are of interest because they provide a natural representation for certain types of expert knowledge. For example, the conditions under which complex tax rules apply can often be readily represented as DNFs. Another nice property of DNF expressions is their universality: every n-bit Boolean function (the type of function considered in this entry unless otherwise noted) $f : \{0,1\}^n \to \{0,1\}$ can be represented as a DNF expression F over at most n variables.

In the basic *probably-approximately correct (PAC)* learning model [24], n is a fixed positive integer and a *target distribution* $\mathcal{D}$ over $\{0,1\}^n$ is assumed fixed. The learner will have black-box access to an unknown arbitrary Boolean f through an *example oracle* $EX(f,\mathcal{D})$ which, when queried, selects $x \in \{0,1\}^n$ at random according to $\mathcal{D}$ and returns the pair $\langle x, f(x)\rangle$. The DNF learning problem is then to design an algorithm provably meeting the following specifications.

Problem 1 (PAC-DNF)

INPUT: Positive integer n; $\epsilon, \delta > 0$; oracle $EX(f,\mathcal{D})$ for $f : \{0,1\}^n \to \{0,1\}$ expressible as DNF having at most s terms and for $\mathcal{D}$ an arbitrary distribution over $\{0,1\}^n$.

OUTPUT: With probability at least $1-\delta$ over the random choices made by $EX(f,\mathcal{D})$ and the algorithm (if any), a function $h : \{0,1\}^n \to \{0,1\}$ such that $\Pr_{x\sim\mathcal{D}}[h(x) \neq f(x)] < \epsilon$ and such that $h(x)$ is computable in time polynomial in n and s for each $x \in \{0,1\}^n$.

RUN TIME: Polynomial in n, s, $1/\epsilon$, and $1/\delta$.

The PAC-DNF problem has not been resolved at the time of this writing, and many believe that no algorithm can solve this problem (see, e.g., [2]). However, DNF has been shown to be learnable in several models that relax the PAC assumptions in various ways. In particular, all polynomial-time learning results to date have limited the choice of the target distribution $\mathcal{D}$ to (at most) the class of *constant-bounded product distributions*. For a constant $c \in (0, \frac{1}{2}]$, a c-bounded product distribution $\mathcal{D}_\mu$ is defined by fixing a vector $\mu \in [c, 1-c]^n$ and having $\mathcal{D}_\mu$ correspond to selecting each bit x_i of x independently so that the mean value of x_i is μ_i; mathematically, this distribution function can be written as $\mathcal{D}_\mu(x) \equiv \prod_{i=1}^{n} (x_i \mu_i + (1-x_i)(1-\mu_i))$. In most learning models, the target distribution is assumed to be selected by an unknown, even adversarial, process from among the allowed distributions; thus, the limitation relative to PAC learning is on the class of allowed distributions, not on how a distribution is chosen. However, in an alternative model of learning from *smoothed product distributions* [17], the mechanism used to choose the target distribution is also constrained as follows. A constant $c \in (0, \frac{1}{2}]$ and an arbitrary vector $\mu \in [2c, 1-2c]^n$ are fixed, a perturbation $\Delta \in [-c, c]^n$ is chosen uniformly at random, and the target distribution is taken to be the c-bounded product distribution $\mathcal{D}_{\mu'}$ such that $\mu' = \mu + \Delta$. The learning algorithm now needs to succeed with only high probability over the choice of Δ (as well as over the usual choices, and in the same run time as for PAC-DNF).

Problem 2 (SMOOTHED-DNF)

INPUT: Same as PAC-DNF except that oracle $EX(f, \mathcal{D}_{\mu'})$ has $\mathcal{D}_{\mu'}$ a smoothed product distribution.

OUTPUT: With probability at least $1-\delta$ over the random choice of μ' and the random choices made by $EX(f, \mathcal{D}_{\mu'})$ and the algorithm (if any), a function $h : \{0,1\}^n \to \{0,1\}$ such that $\Pr_{x \sim \mathcal{D}_{\mu'}}[h(x) \neq f(x)] < \epsilon$.

Various more-informative oracles have also been studied, and we will consider two in particular. A *membership oracle* $MEM(f)$, given x, returns $f(x)$. A *product random walk oracle* $PRW(f, \mathcal{D}_\mu)$ [15] is initialized by selecting an internal state vector x at random according to a fixed arbitrary constant-bounded product distribution $\mathcal{D}_\mu$. Then, on each call to $PRW(f, \mathcal{D}_\mu)$, an $i \in \{1, 2, \ldots, n\}$ is chosen uniformly at random and bit x_i in the internal vector x is replaced with a bit b effectively chosen by flipping a μ_i-biased coin, so that x_i will be 1 with probability μ_i. The oracle then returns the triple $\langle i, x', f(x') \rangle$ consisting of the selected i, the resulting new internal state vector x', and the value that f assigns to this vector. These oracles are used to define two additional DNF learning problems.

Problem 3 (PMEM-DNF)

INPUT: Same as PAC-DNF except that the oracle supplied is $MEM(f)$ and the target distribution is a constant-bounded product distribution $\mathcal{D}_\mu$.

OUTPUT: With probability at least $1-\delta$ over the random choices made by the algorithm, a function h such that $\Pr_{x \sim \mathcal{D}_\mu}[h(x) \neq f(x)] < \epsilon$.

Problem 4 (PRW-DNF)

INPUT: Same as PAC-DNF except that the oracle supplied is $PRW(f, \mathcal{D}_\mu)$ for a constant-bounded product distribution $\mathcal{D}_\mu$.

OUTPUT: With probability at least $1-\delta$ over the random choices made by $PRW(f, \mathcal{D}_\mu)$ and the algorithm (if any), a function h such that $\Pr_{x \sim \mathcal{D}_\mu}[h(x) \neq f(x)] < \epsilon$.

Certain other DNF learning problems and associated results are mentioned briefly in the next section.

Key Results

The first algorithm for efficiently learning arbitrary functions in time polynomial in their DNF size was the Harmonic Sieve [13], which solved the PMEM-DNF problem. This algorithm, like all algorithms for learning arbitrary DNF functions to date, relies in large part on Fourier analysis for its proof of correctness. In particular, a key component of the Sieve involves finding *heavy* (large magnitude) Fourier coefficients of

certain functions using a variation on an algorithm discovered by Goldreich and Levin [11] and first employed to obtain a learning result by Kushilevitz and Mansour [22]. The original Sieve also depends on a certain hypothesis boosting algorithm [10]. Subsequent work on the PMEM-DNF problem [5,7,8,18,21] has produced simpler and/or faster algorithms. In the case of [18], the result is somewhat stronger as the approximator h is *reliable*: it rarely produces false positives. The best run time for the PMEM-DNF problem obtained thus far, $\tilde{O}(ns^4/\epsilon)$, is due to Feldman [7].

Using a membership oracle is a form of *active learning* in which the learning algorithm is able to influence the oracle, as opposed to *passive learning*—exemplified by learning from an example oracle—where the learning algorithm merely accepts the information provided by the oracle. Thus, an apparently significant step in the direction of a solution to PAC-DNF occurred when Bshouty et al. [6] showed that PRW-DNF, which is obviously a passive learning problem, can be solved when the product distribution is constrained to be uniform (the distribution is c-bounded for $c = \frac{1}{2}$). Noise sensitivity analysis plays a key role in Bshouty et al.'s result. Jackson and Wimmer [15] subsequently defined the product random walk model and extended the result of [6] to show that PRW-DNF can be solved in general, not merely for uniform random walks. Both results still rely to some extent on the Harmonic Sieve, or more precisely on a slightly modified version called the Bounded Sieve [3].

More recently, the SMOOTHED-DNF problem has been defined and solved by Kalai, Samorodnitsky, and Teng [17]. As an example oracle is used by the algorithm solving this problem, this result can be viewed as a partial solution to PAC-DNF that applies when the target distribution is chosen in a somewhat "friendly," rather than adversarial, way. Their algorithm avoids the Sieve's need for boosting by combining a form of gradient projection optimization [12] with reliable DNF learning [18] in order to produce a good approximator h, given only the heavy Fourier coefficients of the function f to be approximated. Feldman [9] subsequently discovered a simpler algorithm for this problem.

Algorithms for efficiently learning DNF in a few, less studied, models are mentioned only briefly here due to space constraints. These results include uniform-distribution learning of DNF from a quantum example oracle [4] and from two different types of extended statistical queries [3, 14].

Finally, note that although the focus of this entry is on learning arbitrary functions in time polynomial in their DNF size, there is also a substantial literature on polynomial-time learning of restricted classes of functions representable by constrained forms of DNF, such as monotone DNF (functions expressible as a DNF having no negated variables). For the most part, this work predates the algorithms described here. [19] provides a good summary of many early restricted-DNF results. See also Sellie's algorithm [23] that, roughly speaking, efficiently learns with respect to the uniform target distribution most—but not necessarily all—functions representable by polynomial-size DNF given uniform random examples of each function.

Applications

DNF learning algorithms have proven useful as a basis for learning more expressive function-representation classes. In fact, the Harmonic Sieve, without alteration, can be used to learn with respect to the uniform distribution arbitrary functions in time polynomial in their size as a majority of parity functions (see [13] for a definition of the TOP class and a discussion of its superior expressiveness relative to DNF). In another generalization direction, a DNF expression can be viewed as a union of rectangles of the Boolean hypercube. Atici and Servedio [1] have given a generalized version of the Harmonic Sieve that can, among other things, efficiently learn with respect to uniform an interesting subset of the class of unions of rectangles over $\{0, 1, \ldots, b-1\}^n$ for non-constant b. Both of the preceding results use membership queries. A few quasipolynomial-time passive learning results

for classes more expressive than DNF in various learning models have also been obtained [15, 16] by building on techniques employed originally in DNF learning algorithms.

Open Problems

As indicated at the outset, a resolution, either positively or negatively, of the PAC-DNF question would be a major step forward. Several other DNF questions are also of particular interest. In the problem of *agnostic learning* [20] of DNF, the goal, roughly, is to efficiently find a function h that approximates arbitrary function $f : \{0,1\}^n$ nearly as well as the best s-term DNF, for any fixed s. Is DNF agnostically learnable in any reasonable model? Also welcome would be the discovery of efficient algorithms for learning DNF with respect to interesting classes of distributions beyond product distributions. Finally, although monotone DNF is not a universal function class, it should be mentioned that an efficient example-oracle algorithm for monotone DNF, even if restricted to the uniform target distribution, would be considered a breakthrough.

Cross-References

- ▸ Learning Constant-Depth Circuits
- ▸ Learning Heavy Fourier Coefficients of Boolean Functions
- ▸ PAC Learning

Recommended Reading

1. Atici A, Servedio RA (2008) Learning unions of $\omega(1)$-dimensional rectangles. Theoretical Computer Science 405(3):209 – 222
2. Blum A, Furst M, Kearns M, Lipton R (1994) Cryptographic primitives based on hard learning problems. In: Stinson D (ed) Advances in cryptology—CRYPTO '93. Lecture notes in computer science, vol 773, pp 278–291. Springer, Berlin/Heidelberg
3. Bshouty NH, Feldman V (2002) On using extended statistical queries to avoid membership queries. J Mach Learn Res 2:359–395
4. Bshouty NH, Jackson JC (1999) Learning DNF over the uniform distribution using a quantum example oracle. SIAM J Comput 28(3):1136–1153
5. Bshouty NH, Jackson JC, Tamon C (2004) More efficient PAC-learning of DNF with membership queries under the uniform distribution. J Comput Syst Sci 68(1):205–234
6. Bshouty NH, Mossel E, O'Donnell R, Servedio RA (2005) Learning DNF from random walks. J Comput Syst Sci 71(3):250–265
7. Feldman V (2007) Attribute-efficient and non-adaptive learning of parities and DNF expressions. J Mach Learn Res 8:1431–1460
8. Feldman V (2010) Distribution-specific agnostic boosting. In: Proceedings of innovations in computer science, Tsinghua University, Beijing, pp 241–250
9. Feldman V (2012) Learning DNF expressions from Fourier spectrum. In: COLT 2012—the 25th annual conference on learning theory, Edinburgh, pp 17.1–17.19
10. Freund Y (1995) Boosting a weak learning algorithm by majority. Inf Comput 121(2):256–285
11. Goldreich O, Levin LA (1989) A hard-core predicate for all one-way functions. In: Proceedings of the twenty first annual ACM symposium on theory of computing, Seattle, pp 25–32
12. Gopalan P, Kalai AT, Klivans AR (2008) Agnostically learning decision trees. In: Proceedings of the 40th annual ACM symposium on theory of computing, STOC '08, Victoria, pp 527–536. ACM, New York
13. Jackson J (1997) An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J Comput Syst Sci 55(3):414–440
14. Jackson J, Shamir E, Shwartzman C (1999) Learning with queries corrupted by classification noise. Discret Appl Math 92(2–3):157–175
15. Jackson JC, Wimmer K (2009) New results for random walk learning. In: Dasgupta S, Klivans A (eds) Proceedings of the 22nd annual conference on learning theory, Montreal, pp 267–276. Omnipress, Madison
16. Jackson JC, Klivans AR, Servedio RA (2002) Learnability beyond AC^0. In: STOC '02: Proceedings of the thiry-fourth annual ACM symposium on theory of computing, Montréal, pp 776–784. ACM, New York
17. Kalai AT, Samorodnitsky A, Teng SH (2009) Learning and smoothed analysis. In: Proceedings of the 50th IEEE symposium on foundations of computer science (FOCS), Atlanta, pp 395–404
18. Kalai AT, Kanade V, Mansour Y (2012) Reliable agnostic learning. J Comput Syst Sci 78(5):1481–1495. {JCSS} Special Issue: Cloud Computing 2011
19. Kearns MJ, Vazirani UV (1994) An introduction to computational learning theory. MIT, Cambridge
20. Kearns MJ, Schapire RE, Sellie LM (1994) Toward efficient agnostic learning. Mach Learn 17(2–3):115–141
21. Klivans AR, Servedio RA (2003) Boosting and hard-core set construction. Mach Learn 51(3):217–238
22. Kushilevitz E, Mansour Y (1993) Learning decision trees using the Fourier spectrum. SIAM J Comput 22(6):1331–1348

23. Sellie L (2009) Exact learning of random DNF over the uniform distribution. In: Proceedings of the 41st annual ACM symposium on theory of computing, STOC '09, Bethesda, pp 45–54. ACM, New York
24. Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142

Learning Heavy Fourier Coefficients of Boolean Functions

Luca Trevisan
Department of Computer Science, University of California, Berkeley, CA, USA

Keywords

Error-control codes

Years and Authors of Summarized Original Work

1989; Goldreich, Levin

Problem Definition

The Hamming distance $d_H(y, z)$ between two binary strings y and z of the same length is the number of entries in which y and z disagree. A binary error-correcting code of minimum distance d is a mapping $C : \{0,1\}^k \to \{0,1\}^n$ such that for every two distinct inputs $x, x' \in \{0,1\}^k$, the encodings $C(x)$ and $C(x')$ have Hamming distance at least d. Error-correcting codes are employed to transmit information over noisy channels. If a sender transmits an encoding $C(x)$ of a message x via a noisy channel, and the recipient receives a corrupt bit string $y \neq C(x)$, then, provided that y differs from $C(x)$ in at most $(d-1)/2$ locations, the recipient can recover y from $C(x)$. The recipient can do so by searching for the string x that minimizes the Hamming distance between $C(x)$ and y: there can be no other string x' such that $C(x')$ has Hamming distance $(d-1)/2$ or smaller from y, otherwise $C(x)$ and $C(x')$ would be within Hamming distance $d-1$ or smaller, contradicting the above definition. The problem of recovering the message x from the corrupted encoding y is the *unique decoding* problem for the error-correcting code C. For the above-described scheme to be feasible, the decoding problem must be solvable via an efficient algorithm. These notions are due to Hamming [4].

Suppose that C is a code of minimum distance d, and such that there are pairs of encodings $C(x)$, $C(x')$ whose distance is exactly d. Furthermore, suppose that a communication channel is used that could make a number of errors larger than $(d-1)/2$. Then, if the sender transmits an encoded message using C, it is no longer possible for the recipient to uniquely reconstruct the message. If the sender, for example, transmits $C(x)$, and the recipient receives a string y that is at distance $d/2$ from $C(x)$ and at distance $d/2$ from $C(x')$, then, from the perspective of the recipient, it is equally likely that the original message was x or x'. If the recipient knows an upper bound e on the number of entries that the channel has corrupted, then, given the received string y, the recipient can at least compute the list of all strings x such that $C(x)$ and y differ in at most e locations. An error-correcting code $C : \{0,1\}^k \to \{0,1\}^n$ is (e, L)-list decodable if, for every string $y \in \{0,1\}^n$, the set $\{x \in \{0,1\}^k : d_H(C(x), y) \leq e\}$ has cardinality at most L. The problem of reconstructing the list given y and e is the *list-decoding problem* for the code C. Again, one is interested in efficient algorithms for this problem. The notion of list-decoding is due to Elias [1].

A code $C : \{0,1\}^k \to \{0,1\}^n$ is a *Hadamard code* if every two encodings $C(x)$, $C(x')$ differ in precisely $n/2$ locations. In the Computer Science literature, it is common to use the term Hadamard code for a specific construction (the *Reed–Muller code of order 2*) that satisfies the above property. For a string $a \in \{0,1\}^k$, define the function $\ell_a : \{0,1\}^k \to \{0,1\}$ as

$$\ell_a(x) := \sum_i a_i x_i \bmod 2 .$$

Observe that, for $a \neq b$, the two functions ℓ_a and ℓ_b differ on precisely $(2^k)/2$ inputs. For $n = 2^k$,

the code $H : \{0,1\}^k \to \{0,1\}^n$ maps a message $a \in \{0,1\}^k$ into the n-bit string which is the *truth-table of the function* ℓ_a. That is, if $b_1, \ldots, b_n$ is an enumeration of the $n = 2^k$ elements of $\{0,1\}^k$, and $a \in \{0,1\}^k$ is a message, then the encoding $H(a)$ is the n-bit string that contains the value $\ell_a(b_i)$ in the i-th entry. Note that any two encodings $H(x)$, $H(x')$ differ in precisely $n/2$ entries, and so what was just defined is a Hadamard code. From now on, the term *Hadamard code* will refer exclusively to this construction.

It is known that the Hadamard code $H : \{0,1\}^k \to \{0,1\}^{2^k}$ is $(\frac{1}{2} - \epsilon, \frac{1}{4\epsilon^2})$-list decodable for every $\epsilon > 0$. The Goldreich–Levin results provide efficient list-decoding algorithm.

The following definition of the *Fourier spectrum* of a boolean function will be needed later to state an application of the Goldreich–Levin results to computational learning theory. For a string $a \in \{0,1\}^k$, define the function $\chi_a : \{0,1\}^k \to \{-1,+1\}$ as $\chi_a(x) := (-1)^{\ell_a(x)}$. Equivalently, $\chi_a(x) = (-1)^{\sum_i a_i x_i}$. For two functions $f, g : \{0,1\}^k \to \mathbb{R}$, define their *inner product* as

$$\langle f, g \rangle := \frac{1}{2^k} \sum_x f(x) \cdot g(x) .$$

Then it is easy to see that, for every $a \neq b$, $\langle \chi_a, \chi_b \rangle = 0$, and $\langle \chi_a, \chi_a \rangle = 1$. This means that the functions $\{\chi_a\}_{a \in \{0,1\}^k}$ form an orthonormal basis for the set of all functions $f : \{0,1\}^k \to \mathbb{R}$. In particular, every such function f can be written as a linear combination

$$f(x) = \sum_a \hat{f}(a) \chi_a(x)$$

where the coefficients $\hat{f}(a)$ satisfy $\hat{f}(a) = \langle f, \chi_a \rangle$. The coefficients $\hat{f}(a)$ are called the *Fourier coefficients* of the function f.

Key Results

Theorem 1 *There is a randomized algorithm GL that, given in input an integer k and a parameter $\epsilon > 0$, and given oracle access to a function $f : \{0,1\}^k \to \{0,1\}$, runs in time polynomial in $1/\epsilon$ and in k and outputs, with high probability over its internal coin tosses, a set $S \subseteq \{0,1\}^k$ that contains all the strings $a \in \{0,1\}^k$ such that ℓ_a and f agree on at least a $1/2 + \epsilon$ fraction of inputs.*

Theorem 1 is proved by Goldreich and Levin [3]. The result can be seen as a list-decoding for the Hadamard code $H : \{0,1\}^k \to \{0,1\}^{2^k}$; remarkably, the algorithm runs in time polynomial in k, which is poly-logarithmic in the length of the given corrupted encoding.

Theorem 2 *There is a randomized algorithm KM that given in input an integer k and parameters $\epsilon, \delta > 0$, and given oracle access to a function $f : \{0,1\}^k \to \{0,1\}$, runs in time polynomial in $1/\epsilon$, in $1/\delta$, and in k and outputs a set $S \subseteq \{0,1\}^k$ and a value g(a) for each $a \in S$.*

With high probability over the internal coin tosses of the algorithm,

1 *S contains all the strings $a \in \{0,1\}^k$ such that $|\hat{f}(a)| \geq \epsilon$, and*
2 *For every $a \in S$, $|\hat{f}(a) - g(a)| \leq \delta$.*

Theorem 2 is proved by Kushilevitz and Mansour [5]; it is an easy consequence of the Goldreich–Levin algorithm.

Applications

There are two key applications of the Goldreich–Levin algorithm: one is to cryptography and the other is to computational learning theory.

Application in Cryptography

In cryptography, a *one-way permutation* is a family of functions $\{p_n\}_{n \geq 1}$ such that: (i) for every n, $p_n : \{0,1\}^n \to \{0,1\}^n$ is bijective, (ii) there is a polynomial time algorithm that, given $x \in \{0,1\}^n$, computes $p_n(x)$, and (iii) for every polynomial time algorithm A and polynomial q, and for every sufficiently large n,

$$\mathbb{P}_{x\sim\{0,1\}^n}[A(p_n(x)) = x] \le \frac{1}{q(n)}\,.$$

That is, even though computing $p_n(x)$ given x is doable in polynomial time, the task of computing x given $p_n(x)$ is intractable. A *hard core predicate* for a one-way permutation $\{p_n\}$ is a family of functions $\{B_n\}_{n\ge1}$ such that: (i) for every n, $B_n : \{0,1\}^n \to \{0,1\}$, (ii) there is a polynomial time algorithm that, given $x \in \{0,1\}^n$, computes $B_n(x)$, and (iii) for every polynomial time algorithm A and polynomial q, and for every sufficiently large n,

$$\mathbb{P}_{x\sim\{0,1\}^n}[A(p_n(x)) = B_n(x)] \le \frac{1}{2} + \frac{1}{q(n)}\,.$$

That is, even though computing $B_n(x)$ given x is doable in polynomial time, the task of computing $B_n(x)$ given $p_n(x)$ is intractable.

Goldreich and Levin [3] use their algorithm to show that every one-way permutation has a hard-core predicate, as stated in the next theorem.

Theorem 3 *Let $\{p_n\}$ be a one-way permutation; define $\{p'_n\}$ such that $p'_{2n}(x,y) := p_n(x), y$ and let $B_{2n}(x,y) := \sum_i x_i y_i \bmod 2$. (For odd indices, let $p'_{2n+1}(z,b) := p'_{2n}(z)$ and $B_{2n+1}(z,b) := B_{2n}(z)$.)*

Then $\{p'_n\}$ is a one-way permutation and $\{B_n\}$ is a hard-core predicate for $\{p'_n\}$.

This result is used in efficient constructions of pseudorandom generators, pseudorandom functions, and private-key encryption schemes based on one-way permutations. The interested reader is referred to Chapter 3 in Goldreich's monograph [2] for more details.

There are also related applications in computational complexity theory, especially in the study of average-case complexity. See [7] for an overview.

Application in Computational Learning Theory

Loosely speaking, in computational learning theory one is given an unknown function $f : \{0,1\}^k \to \{0,1\}$ and one wants to compute, via an efficient randomized algorithm, a representation of a function $g : \{0,1\}^k \to \{0,1\}$ that agrees with f on most inputs. In the *PAC learning* model, one has access to f only via randomly sampled pairs $(x, f(x))$; in the model of *learning with queries*, instead, one can evaluate f at points of one's choice. Kushilevitz and Mansour [5] suggest the following algorithm: using the algorithm of Theorem 2, find a set S of large coefficients and approximations $g(a)$ of the coefficients $\hat{f}(a)$ for $a \in S$. Then define the function $g(x) = \sum_{a\in S} g(a)\chi_a(x)$. If the error caused by the absence of the smaller coefficients and the imprecision in the larger coefficient is not too large, g and f will agree on most inputs. (A technical point is that g as defined above is not necessarily a boolean function, but it can be easily "rounded" to be boolean.) Kushilevitz and Mansour show that such an approach works well for the class of functions f for which $\sum_a |\hat{f}(a)|$ is bounded, and they observe that functions of small *decision tree complexity* fall into this class. In particular, they derive the following result.

Theorem 4 *There is a randomized algorithm that, given input parameters k, m, ε and δ, and given oracle access to a function $f : \{0,1\}^k \to \{0,1\}$ of decision tree complexity at most m, runs in time polynomial in k, m, $1/\epsilon$ and $\log 1/\delta$ and, with probability at least $1-\delta$ over its internal coin tosses, outputs a circuit computing a function $g : \{0,1\}^k \to \{0,1\}$ that agrees with f on at least a $1-\epsilon$ fraction of inputs.*

Another application of the Kushilevitz–Mansour technique is due to Linial, Mansour, and Nisan [6].

Cross-References

▸ Decoding Reed-Solomon Codes

Recommended Reading

1. Elias P (1957) List decoding for noisy channels. Technical report 335. Research Laboratory of Electronics, MIT, Cambridge

2. Goldreich O (2001) The foundations of cryptography, vol 1. Cambridge University Press, Cambridge
3. Goldreich O, Levin L (1989) A hard-core predicate for all one-way functions. In: Proceedings of the 21st ACM symposium on theory of computing, Seattle, 14–17 May 1989, pp 25–32
4. Hamming R (1950) Error detecting and error correcting codes. Bell Syst Tech J 29:147–160
5. Kushilevitz E, Mansour Y (1993) Learning decision trees using the Fourier spectrum. SIAM J Comput 22(6):1331–1348
6. Linial N, Mansour Y, Nisan N (1993) Constant depth circuits, Fourier transform and learnability. J ACM 40(3):607–620
7. Trevisan L (2004) Some applications of coding theory in computational complexity. Quad Mat 13:347–424, arXiv:cs.CC/0409044

Learning Significant Fourier Coefficients over Finite Abelian Groups

Adi Akavia
Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, USA

Keywords

Finding heavy fourier coefficients; Learning heavy fourier coefficients

Years and Authors of Summarized Original Work

2003; Akavia, Goldwasser, Safra

Problem Definition

Fourier transform is among the most widely used tools in computer science. Computing the Fourier transform of a signal of length N may be done in time $\Theta(N \log N)$ using the Fast Fourier Transform (FFT) algorithm. This time bound clearly cannot be improved below $\Theta(N)$, because the output itself is of length N. Nonetheless, it turns out that in many applications it suffices to find only the *significant Fourier coefficients*, i.e., Fourier coefficients occupying, say, at least 1 % of the energy of the signal. This motivates the problem discussed in this entry: the problem of efficiently finding and approximating the significant Fourier coefficients of a given signal (SFT, in short). A naive solution for SFT is to first compute the entire Fourier transform of the given signal and then to output only the significant Fourier coefficients; thus yielding no complexity improvement over algorithms computing the entire Fourier transform. In contrast, SFT can be solved far more efficiently in running time $\widetilde{\Theta}(\log N)$ and while reading at most $\widetilde{\Theta}(\log N)$ out of the N signal's entries [2]. This fast algorithm for SFT opens the way to applications taken from diverse areas including computational learning, error correcting codes, cryptography, and algorithms.

It is now possible to formally define the SFT problem, restricting our attention to discrete signals. Use functional notation where a signal is a function $f: G \to \mathbb{C}$ over a finite Abelian group G, its energy is $\|f\|_2^2 \stackrel{\text{def}}{=} 1/|G| \sum_{x \in G} |f(x)|^2$, and its maximal amplitude is $\|f\|_\infty \stackrel{\text{def}}{=} \max\{|f(x)| \; | x \in G\}$. (For readers more accustomed to vector notation, the authors remark that there is a simple correspondence between vector and functional notation. For example, a one-dimensional signal $(v_1, \ldots, v_N) \in \mathbb{C}^{\mathbb{N}}$ corresponds to the function $f: \mathbb{Z}_N \to \mathbb{C}$ defined by $f(i) = v_i$ for all $i = 1, \ldots, N$. Likewise, a two-dimensional signal $M \in \mathbb{C}^{N_1 \times N_2}$ corresponds to the function $f: \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2} \to \mathbb{C}$ defined by $f(i, j) = M_{ij}$ for all $i = 1, \ldots, N_1$ and $j = 1, \ldots, N_2$.) For ease of presentation assume without loss of generality that $G = \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2} \times \cdots \times \mathbb{Z}_{N_k}$ for $N_1, \ldots, N_k \in \mathbb{Z}^+$ (i.e., positive integers), and for $\mathbb{Z}_N$ is the additive group of integers modulo N.

The *Fourier transform* of f is the function $\widehat{f}: G \to \mathbb{C}$ defined for each $\alpha = (\alpha_1, \ldots, \alpha_k) \in G$ by

$$\widehat{f}(\alpha) \stackrel{\text{def}}{=} \frac{1}{|G|} \sum_{(x_1, \ldots, x_k) \in G} \left[f(x_1, \ldots, x_k) \prod_{j=1}^{k} \omega_{N_j}^{\alpha_j x_j} \right],$$

where $\omega_{N_j} = \exp(i2\pi/N_j)$ is a primitive root of unity of order N_j. For any $\alpha \in G$, $val_\alpha \in \mathbb{C}$ and $\tau, \varepsilon \in [0,1]$, say that α is a *τ-significant Fourier coefficient* iff $|\widehat{f}(\alpha)|^2 \geq \tau \|f\|_2^2$, and say that val_α is an *ε-approximation for* $\widehat{f}(\alpha)$ iff $|val_\alpha - \widehat{f}(\alpha)| < \varepsilon$.

Problem 1 (SFT)
INPUT: Integers $N_1, \ldots, N_k \geq 2$ specifying the group $G = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$, a threshold $\tau \in (0,1)$, an approximation parameter $\varepsilon \in (0,1)$, and oracle access (Say that an algorithm is given *oracle access* to a function f over G, if it can request and receive the value $f(x)$ for any $x \in G$ in unit time.) to $f : G \to \mathbb{C}$.
OUTPUT: A list of all τ-significant Fourier coefficients of f along with ε-approximations for them.

Key Results

The key result of this entry is an algorithm solving the SFT problem which is much faster than algorithms for computing the entire Fourier transform. For example, for f a Boolean function over $\mathbb{Z}_N$, the running time of this algorithm is $\log N \cdot poly(\log\log N, 1/\tau, 1/\varepsilon)$, in contrast to the $\Theta(N \log N)$ running time of the FFT algorithm. This algorithm is named the SFT algorithm.

Theorem 1 (SFT algorithm [2]) *There is an algorithm solving the SFT problem with running time* $\log|G| \cdot poly(\log\log|G|, \|f\|_\infty/\|f\|_2, 1/\tau, 1/\varepsilon)$ *for* $|G| = \prod_{j=1}^{k} N_j$ *the cardinality of* G.

Remarks

1. The above result extends to functions f over any finite Abelian group G, as long as the algorithm is given a description of G by its generators and their orders [2].
2. The SFT algorithm reads at most $\log|G| \cdot poly(\log\log|G|, \|f\|_\infty/\|f\|_2, 1/\tau, 1/\varepsilon)$ out of the $|G|$ values of the signal.
3. The SFT algorithm is non adaptive, that is, oracle queries to f are independent of the algorithm's progress.
4. The SFT algorithm is a probabilistic algorithm having a small error probability, where probability is taken over the internal coin tosses of the algorithm. The error probability can be made arbitrarily small by standard amplification techniques.

The SFT algorithm follows a line of works solving the SFT problem for restricted function classes. Goldreich and Levin [9] gave an algorithm for Boolean functions over the group $\mathbb{Z}_2^k = \{0,1\}^k$. The running time of their algorithm is polynomial in $k, 1/\tau$ and $1/\varepsilon$. Mansour [10] gave an algorithm for complex functions over groups $G = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$ provided that $N_1, \ldots, N_k$ are powers of two. The running time of his algorithm is polynomial in $\log|G|, \log(\max_{\alpha\in G}|\widehat{f}(\alpha)|), 1/\tau$ and $1/\varepsilon$. Gilbert et al. [6] gave an algorithm for complex functions over the group $\mathbb{Z}_N$ for any positive integer N. The running time of their algorithm is polynomial in $\log N$, $\log(\max_{x\in\mathbb{Z}_N} f(x)/\min_{x\in\mathbb{Z}_N} f(x)), 1/\tau$ and $1/\varepsilon$. Akavia et al. [2] gave an algorithm for complex functions over any finite Abelian group. The latter [2] improves on [6] even when restricted to functions over $\mathbb{Z}_N$ in achieving $\log N \cdot poly(\log\log N)$ rather than $poly(\log N)$ dependency on N. Subsequent works [7] improved the dependency of [6] on τ and ε.

Applications

Next, the paper surveys applications of the SFT algorithm [2] in the areas of computational learning theory, coding theory, cryptography, and algorithms.

Applications in Computational Learning Theory

A common task in computational learning is to find a hypothesis h approximating a function f, when given only samples of the function f. Samples may be given in a variety of forms, e.g.,

via oracle access to f. We consider the following variant of this learning problem: f and h are complex functions over a finite Abelian group $G = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$, the goal is to find h such that $\|f - h\|_2^2 \leq \gamma \|f\|_2^2$ for $\gamma > 0$ an approximation parameter, and samples of f are given via oracle access.

A straightforward application of the SFT algorithm gives an efficient solution to the above learning problem, provided that there is a small set $\Gamma \subseteq G$ s.t. $\sum_{\alpha \in \Gamma} |\widehat{f}(\alpha)|^2 > (1 - \gamma/3)\|f\|_2^2$. The learning algorithm operates as follows. It first runs the SFT algorithm to find all $\alpha = (\alpha_1, \ldots, \alpha_k) \in G$ that are $\gamma/|\Gamma|$-significant Fourier coefficients of f along with their $\gamma/|\Gamma|\|f\|_\infty$-approximations val_α, and then returns the hypothesis

$$h(x_1, \ldots, x_k) \stackrel{\text{def}}{=} \sum_{\alpha \text{ is } \gamma/|\Gamma| - \text{significant}} val_\alpha \cdot \prod_{j=1}^{k} \omega_{N_j}^{\alpha_j x_j}.$$

This hypothesis h satisfies that $\|f - h\|_2^2 \leq \gamma \|f\|_2^2$. The running time of this learning algorithm and the number of oracle queries it makes is polynomially bounded by $\log |G|$, $\|f\|_\infty / \|f\|_2$, $|\Gamma| \|f\|_\infty / \gamma$.

Theorem 2 *Let $f: G \to \mathbb{C}$ be a function over $G = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$, and $\gamma > 0$ an approximation parameter. Denote $t = \min\{|\Gamma| \ |\Gamma \subseteq G \text{ s.t. } \sum_{\alpha \in \Gamma} |\widehat{f}(\alpha)|^2 > (1 - \gamma/3)\|f\|_2^2\}$. There is an algorithm that given $N_1, \ldots, N_k$, γ, and oracle access to f, outputs a (short) description of $h: G \to \mathbb{C}$ s.t. $\|f - h\|_2^2 < \gamma \|f\|_2^2$. The running time of this algorithm is $\log |G| \cdot poly(\log \log |G|, \|f\|_\infty / \|f\|_2, t\|f\|_\infty / \gamma)$.*

More examples of function classes that can be efficiently learned using our SFT algorithm are given in [3].

Applications in Coding Theory

Error correcting codes encode messages in a way that allows *decoding*, that is, recovery of the original message, even in the presence of noise. When the noise is very high, unique decoding may be infeasible, nevertheless it may still be possible to *list decode*, that is, to find a short list of messages containing the original message. Codes equipped with an efficient list decoding algorithm have found many applications (see [11] for a survey).

Formally, a binary code is a subset $C \subseteq \{0,1\}^*$ of *codewords* each encoding some message. Denote by $C_{N,x} \in \{0,1\}^N$ a codeword of length N encoding a message x. The *normalized Hamming distance* between a codeword $C_{N,x}$ and a received word $w \in \{0,1\}^N$ is $\Delta(C_{N,x}, w) \stackrel{\text{def}}{=} 1/N |\{i \in \mathbb{Z}_N \ | C_{N,x}(i) \neq w(i)\}|$ where $C_{N,x}(i)$ and $w(i)$ are the ith bits of $C_{N,x}$ and w, respectively. Given $w \in \{0,1\}^N$ and a noise parameter $\eta > 0$, the *list decoding* task is to find a list of all messages x such that $\Delta(C_{N,x}, w) < \eta$. The received word w may be given explicitly or implicitly; we focus on the latter where oracle access to w is given. Goldreich and Levin [9] give a list decoding algorithm for Hadamard codes, using in a crucial way their algorithm solving the SFT problem for functions over the Boolean cube.

The SFT algorithm for functions over ZZ_N is a key component in a list decoding algorithm given by Akavia et al. [2]. This list decoding algorithm is applicable to a large class of codes. For example, it is applicable to the code $C^{msb} = \{C_{N,x}: \mathbb{Z}_N \to \{0,1\}\}_{x \in \mathbb{Z}_N^*, N \in \mathbb{Z}^+}$ whose codewords are $C_{N,x}(j) = msb_N(j \cdot x \mod N)$ for $msb_N(y) = 1$ iff $y \geq N/2$ and $msb_N(y) = 0$ otherwise. More generally, this list decoding algorithm is applicable to any *Multiplication code* $C^{\mathcal{P}}$ for $\mathcal{P}$ a family of *balanced* and *well concentrated* functions, as defined below. The running time of this list decoding algorithm is polynomial in $\log N$ and $1/(1 - 2\eta)$, as long as $\eta < \frac{1}{2}$.

Abstractly, the list decoding algorithm of [2] is applicable to any code that is "balanced," "(well) concentrated," and "recoverable," as defined next (and those Fourier coefficients have small greatest common divisor (GCD) with N). A code is *balanced* if $\Pr_{j \in \mathbb{Z}_N}[C_{N,x}(j) = 0] = \Pr_{j \in \mathbb{Z}_N}[C_{N,x}(j) = 1]$ for every codeword $C_{N,x}$. A code is *(well) concentrated* if its codewords can be approximated

by a small number of significant coefficients in their Fourier representation (and those Fourier coefficients have small greatest common divisor (GCD) with N). A code is *recoverable* if there is an efficient algorithm mapping each Fourier coefficient α to a short list of codewords for which α is a significant Fourier coefficient. The key property of concentrated codes is that received words w share a significant Fourier coefficient with all close codewords $C_{N,x}$. The high level structure of the list decoding algorithm of [2] is therefore as follows. First it runs the SFT algorithm to find all significant Fourier coefficients α of the received word w. Second for each such α, it runs the recovery algorithm to find all codewords $C_{N,x}$ for which α is significant. Finally, it outputs all those codewords $C_{N,x}$.

Definition 1 (Multiplication codes [2]) Let $\mathcal{P} = \{P_N : \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ be a family of functions. Say that $C^{\mathcal{P}} = \{C_{N,x} : \mathbb{Z}_N \to \{0,1\}\}_{x\in\mathbb{Z}_N^*, N\in\mathbb{Z}^+}$ is a *multiplication code for* $\mathcal{P}$ if for every $N \in \mathbb{Z}^+$ and $x \in \mathbb{Z}_N^*$, the encoding $C_{N,x} : \mathbb{Z}_N \to \{0,1\}$ of x is defined by

$$C_{N,x}(j) = P(j \cdot x \bmod N) .$$

Definition 2 (Well concentrated [2]) Let $\mathcal{P} = \{P_N : \mathbb{Z}_N \to \mathbb{C}\}_{N\in\mathbb{Z}^+}$ be a family of functions. Say that $\mathcal{P}$ is *well concentrated* if $\forall N \in \mathbb{Z}^+, \gamma > 0$, $\exists \Gamma \subseteq \mathbb{Z}_N$ s.t. (i) $|\Gamma| \leq poly(\log N/\gamma)$, (ii) $\sum_{\alpha\in\Gamma} |\widehat{P_N}(\alpha)|^2 \geq (1-\gamma)\|P_N\|_2^2$, and (iii) for all $\alpha \in \Gamma$, $gcd(\alpha, N) \leq poly(\log N/\gamma)$ (where $gcd(\alpha, N)$ is the greatest common divisor of α and N).

Theorem 3 (List decoding [2]) *Let $\mathcal{P} = \{P_N : \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ be a family of efficiently computable ($\mathcal{P} = \{P_N : \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ is a family of efficiently computable functions if there is an algorithm that given any $N \in \mathbb{Z}^+$ and $x \in \mathbb{Z}_N$ outputs $P_N(x)$ in time $poly(\log N)$.), well concentrated, and balanced functions. Let $C^{\mathcal{P}} = \{C_{N,x} : \mathbb{Z}_N \to \{0,1\}\}_{x\in\mathbb{Z}_N^*, N\in\mathbb{Z}^+}$ be the multiplication code for $\mathcal{P}$. Then there is an algorithm that, given $N \in \mathbb{Z}_N^+$, $\eta < \frac{1}{2}$ and oracle access to $w : \mathbb{Z}_N \to \{0,1\}$, outputs all $x \in \mathbb{Z}_N^*$ for which $\Delta(C_{N,x}, w) < \eta$. The running time of this algorithm is polynomial in* $\log N$ *and* $1/(1-2\eta)$.

Remarks

1. The requirement that $\mathcal{P}$ is a family of *efficiently computable* functions can be relaxed. It suffices to require that the list decoding algorithm receives or computes a set $\Gamma \subseteq \mathbb{Z}_N$ with properties as specified in Definition 2.
2. The requirement that $\mathcal{P}$ is a family of *balanced* functions can be relaxed. Denote $bias(\mathcal{P}) = \min_{b\in\{0,1\}} \inf_{N\in\mathbb{Z}^+} \Pr_{j\in\mathbb{Z}_N}[P_N(j) = b]$. Then the list decoding algorithm of [2] is applicable to $C^{\mathcal{P}}$ even when $bias(\mathcal{P}) \neq \frac{1}{2}$, as long as $\eta < bias(\mathcal{P})$.

Applications in Cryptography

Hard-core predicates for one-way functions are a fundamental cryptographic primitive, which is central for many cryptographic applications such as pseudo-random number generators, semantic secure encryption, and cryptographic protocols. Informally speaking, a Boolean predicate P is a *hard-core predicate* for a function f if $P(x)$ is easy to compute when given x, but hard to guess with a non-negligible advantage beyond 50% when given only $f(x)$. The notion of hard-core predicates was introduced by Blum and Micali [2]. Goldreich and Levin [9] showed a randomized hardcore predicate for any one-way function, using in a crucial way their algorithm solving the SFT problem for functions over the Boolean cube.

Akavia et al. [2] introduce a unifying framework for proving that a predicate P is hard-core for a one-way function f. Applying their framework they prove for a wide class of predicates – *segment predicates* – that they are hard-core predicates for various well-known candidate one-way functions. Thus showing new hard-core predicates for well-known one-way function candidates as well as reproving old results in an entirely different way.

Elaborating on the above, a *segment predicate* is any assignment of Boolean values to an arbitrary partition of $\mathbb{Z}_N$ into $poly(\log N)$ segments,

or dilations of such an assignment. Akavia et al. [2] prove that any segment predicate is hard-core for any one-way function f defined over $\mathbb{Z}_N$ for which, for a non-negligible fraction of the x's in $\mathbb{Z}_N$, given $f(x)$ and y, one can efficiently compute $f(xy)$ (where xy is multiplication in $\mathbb{Z}_N$). This includes the following functions: the *exponentiation* function $EXP_{p,g}: \mathbb{Z}_p \to \mathbb{Z}_p^*$ defined by $EXP_{p,g}(x) = g^x \bmod p$ for each prime p and a generator g of the group $\mathbb{Z}_p^*$; the *RSA* function $RSA: \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by $RSA(x) = \mathrm{e}^x \bmod N$ for each $N = pq$ a product of two primes p, q, and e co-prime to N; the *Rabin* function $Rabin: \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by $Rabin(x) = x^2 \bmod N$ for each $N = pq$ a product of two primes p, q; and the *elliptic curve log* function defined by $ECL_{a,b,p,Q} = xQ$ for each elliptic curve $E_{a,b,p}(Z_p)$ and Q a point of high order on the curve.

The SFT algorithm is a central tool in the framework of [2]: Akavia et al. take a list decoding methodology, where computing a hard-core predicate corresponds to computing an entry in some error correcting code, predicting a predicate corresponds to access to an entry in a corrupted codeword, and the task of inverting a one-way function corresponds to the task of list decoding a corrupted codeword. The codes emerging in [2] are multiplication codes (see Definition 1 above), which are list decoded using the SFT algorithm.

Definition 3 (Segment predicates [2]) Let $\mathcal{P} = \{P_N: \mathbb{Z}_N \to \{0,1\}\}_{N \in \mathbb{Z}^+}$ be a family of predicates that are non-negligibly far from constant (A family of functions $\mathcal{P} = \{P_N: \mathbb{Z}_N \to \{0,1\}\}_{N \in \mathbb{Z}^+}$ is *non-negligibly far from constant* if $\forall N \in \mathbb{Z}^+$ and $b \in \{0,1\}$, $\Pr_{j \in \mathbb{Z}_N}[P_N(j) = b] \leq 1 - poly(1/\log N)$).

- It can be sayed that P_N is a basic t-segment predicate if $P_N(x+1) \neq P_N(x)$ for at most t x's in $\mathbb{Z}_N$.
- It can be sayed that P_N is a t-segment predicate if there exist a basic t-segment predicate P' and $a \in \mathbb{Z}_N$ which is co-prime to N s.t. $\forall x \in \mathbb{Z}_N$, $P_N(x) = P'(x/a)$.
- It can be sayed that $\mathcal{P}$ is a family of segment predicates if $\forall N \in \mathbb{Z}^+$, P_N is a $t(N)$-segment predicate for $t(N) \leq poly(\log N)$.

Theorem 4 (Hardcore predicates [2]) *Let $\mathcal{P}$ be a family of segment predicates. Then, $\mathcal{P}$ is hard-core for RSA, Rabin, EXP, ECL, under the assumption that these are one-way functions.*

Application in Algorithms

Our modern times are characterized by information explosion incurring a need for faster and faster algorithms. Even algorithms classically regarded as efficient – such as the FFT algorithm with its $\Theta(N \log N)$ complexity – are often too slow for data-intensive applications, and linear or even sub-linear algorithms are imperative. Despite the vast variety of fields and applications where algorithmic challenges arise, some basic algorithmic building blocks emerge in many of the existing algorithmic solutions. Accelerating such building blocks can therefore accelerate many existing algorithms. One of these recurring building blocks is the Fast Fourier Transform (FFT) algorithm. The SFT algorithm offers a great efficiency improvement over the FFT algorithm for applications where it suffices to deal only with the significant Fourier coefficients. In such applications, replacing the FFT building block with the SFT algorithm accelerates the $\Theta(N \log N)$ complexity in each application of the FFT algorithm to $poly(\log N)$ complexity [1]. Lossy compression is an example of such an application [1, 5, 8]. To elaborate, central component in several transform compression methods (e.g., JPEG) is to first apply Fourier (or Cosine) transform to the signal, and then discard many of its coefficients. To accelerate such algorithms – instead of computing the entire Fourier (or Cosine) transform – the SFT algorithm can be used to directly approximate only the significant Fourier coefficients. Such an accelerated algorithm achieves compression guarantee as good as the original algorithm (and possibly better), but with running time improved to $poly(\log N)$ in place of the former $\Theta(N \log N)$.

Cross-References

- Abelian Hidden Subgroup Problem
- Learning Constant-Depth Circuits
- Learning DNF Formulas
- Learning Heavy Fourier Coefficients of Boolean Functions
- Learning with Malicious Noise
- List Decoding near Capacity: Folded RS Codes
- PAC Learning
- Statistical Query Learning

Recommended Reading

1. Akavia A, Goldwasser S (2005) Manuscript submitted as an NSF grant, awarded CCF-0514167
2. Akavia A, Goldwasser S, Safra S (2003) Proving hard-core predicates using list decoding. In: Proceedings of the 44th symposium on foundations of computer science (FOCS'03). IEEE Computer Society, pp 146–157
3. Atici A, Servedio RA (2006) Learning unions of $\omega(1)$-dimensional rectangles. ALT, pp 32–47
4. Blum M, Micali S (1984) How to generate cryptographically strong sequences of pseudo-random bits. SIAM J Comput 4(13):850–864
5. Cormode G, Muthukrishnan S (2006) Combinatorial algorithms for compressed sensing. In: 13th International colloquium on structural information and communication complexity (SIROCCO), Chester, 2–5 July 2006, pp 280–294
6. Gilbert AC, Guha S, Indyk P, Muthukrishnan S, Strauss M (2002) Near-optimal sparse fourier representations via sampling. In: Proceedings of the thiry-fourth annual ACM symposium on theory of computing (STOC'02). ACM, pp 152–161
7. Gilbert AC, Muthukrishnan S, Strauss MJ (2005) Improved time bounds for near-optimal sparse Fourier representation via sampling. In: Proceedings of SPIE Wavelets XI, San Diego
8. Gilbert AC, Strauss MJ, Tropp JA, Vershynin R (2007) One sketch for all: fast algorithms for compressed sensing. In: Proceedings of the 39th ACM symposium on theory of computing (STOC'07)
9. Goldreich O, Levin L (1989) A hard-core predicate for all one-way functions. In: Proceedings of the 27th ACM symposium on theory of computing (STOC'89)
10. Mansour Y (1995) Randomized interpolation and approximation of sparse polynomials. SIAM J Comput 24:357–368
11. Sudan M (2000) List decoding: algorithms and applications. SIGART Newsl 31:16–27

Learning with Malicious Noise

Peter Auer
Chair for Information Technology,
Montanuniversitaet Leoben, Leoben, Austria

Keywords

Agnostic learning; Boosting; Clustering; Computational learning theory; Efficient learning; Malicious noise; PAC (probably approximately correct) learning; Sample complexity; Statistical query learning

Years and Authors of Summarized Original Work

1993; Kearns, Li
1999; Cesa-Bianchi, Dichterman, Fischer, Shamir, Simon

Problem Definition

This problem is concerned with PAC learning of concept classes when training examples are affected by malicious errors. The PAC (probably approximately correct) model of learning (also known as the distribution-free model of learning) was introduced by Valiant [13]. This model makes the idealized assumption that error-free training examples are generated from the same distribution which is then used to evaluate the learned hypothesis. In many environments, however, there is some chance that an erroneous example is given to the learning algorithm. The malicious noise model – again introduced by Valiant [14] – extends the PAC model by allowing example errors of any kind: it makes no assumptions on the nature of the errors that occur. In this sense the malicious noise model is a worst-case model of errors, in which errors may be generated by an adversary whose goal is to foil the learning algorithm. Kearns and Li [8,9] study the maximal malicious error rate such that learning is still possible. They also provide a canonical method

to transform any standard learning algorithm into an algorithm which is robust against malicious noise.

Notations Let X be a set of instances. The goal of a learning algorithm is to infer an unknown subset $C \subseteq X$ of instances which exhibit a certain property. Such subsets are called concepts. It is known to the learning algorithm that the correct concept C is from a concept class $\mathcal{C} \subseteq 2^X$, $C \in \mathcal{C}$. Let $C(x)=1$ if $x \in C$ and $C(x)=0$ if $x \notin C$. As input the learning algorithm receives an accuracy parameter $\varepsilon > 0$, a confidence parameter $\delta > 0$, and the malicious noise rate $\beta \geq 0$. The learning algorithm may request a sample of labeled instances $S = \langle (x_1, \ell_1), \ldots, (x_m, \ell_m) \rangle$, $x_i \in X$, and $\ell_i \in \{0, 1\}$ and produces a hypothesis $H \subseteq X$. Let $\mathcal{D}$ be the unknown distribution of instances in X. Learning is successful if H misclassifies an example with probability less than ε, $\mathrm{err}_{\mathcal{D}}(C, H) := \mathcal{D}\{x \in X : C(x) \neq H(x)\} < \varepsilon$. A learning algorithm is required to be successful with probability $1 - \delta$. The error of a hypothesis H in respect to a sample S of labeled instances is defined as $\mathrm{err}(S, H) := |\{(x, \ell) \in S : H(x) \neq \ell\}|/|S|$.

The VC dimension $\mathrm{VC}(\mathcal{C})$ of a concept class $\mathcal{C}$ is the maximal number of instances $x_1, \ldots, x_d$ such that $\{(C(x_1), \ldots, C(x_d)) : C \in \mathcal{C}\} = \{0, 1\}^d$. The VC dimension is a measure of the difficulty to learn concept class $\mathcal{C}$ [4].

To investigate the computational complexity of learning algorithms, sequences of concept classes with increasing complexity $(X_n, \mathcal{C}_n)_n = \langle (X_1, \mathcal{C}_1), (X_2, \mathcal{C}_2), \ldots \rangle$ are considered. In this case the learning algorithm receives also a complexity parameter n as input.

Generation of Examples In the malicious noise model, the labeled instances (x_i, ℓ_i) are generated independently from each other by the following random process:

(a) Correct examples: with probability $1 - \beta$, an instance x_i is drawn from distribution $\mathcal{D}$ and labeled by the correct concept C, $\ell_i = C(x_i)$.

(b) Noisy examples: with probability β, an arbitrary example (x_i, ℓ_i) is generated, possibly by an adversary.

Problem 1 (Malicious Noise Learning of $(X, \mathcal{C})$)

INPUT: Reals $\varepsilon, \delta > 0$, $\beta \geq 0$
OUTPUT: A hypothesis $H \subseteq X$

For any distribution $\mathcal{D}$ on X and any concept $C \in \mathcal{C}$, the algorithm needs to produce with probability $1 - \delta$ a hypothesis H such that $\mathrm{err}_{\mathcal{D}}(C, H) < \varepsilon$. The probability $1 - \delta$ is taken in respect to the random sample $(x_1, \ell_1), \ldots, (x_m, \ell_m)$ requested by the algorithm. The examples (x_i, ℓ_i) are generated as defined above.

Problem 2 (Polynomial Malicious Noise Learning of $(X_n, \mathcal{C}_n)_n$)

INPUT: Reals $\varepsilon, \delta > 0$, $\beta \geq 0$, integer $n \geq 1$
OUTPUT: A hypothesis $H \subseteq X_n$

For any distribution $\mathcal{D}$ on X_n and any concept $C \in \mathcal{C}_n$, the algorithm needs to produce with probability $1 - \delta$ a hypothesis H such that $\mathrm{err}_{\mathcal{D}}(C, H) < \varepsilon$. The computational complexity of the algorithm must be bounded by a polynomial in $1/\varepsilon$, $1/\delta$, and n.

Key Results

Theorem 1 ([9]) *Let $\mathcal{C}$ be a nontrivial concept class with two concepts $C_1, C_2 \in \mathcal{C}$ that are equal on an instance x_1 and differ on another instance x_2, $C_1(x_1) = C_2(x_1)$, and $C_1(x_2) \neq C_2(x_2)$. Then no algorithm can learn $\mathcal{C}$ with malicious noise rate $\beta \geq \frac{\varepsilon}{1+\varepsilon}$.*

Theorem 2 *Let $\Delta > 0$ and $d = \mathrm{VC}(\mathcal{C})$. For a suitable constant κ, any algorithm which requests a sample S of $m \geq \kappa \frac{\varepsilon d \log 1/(\Delta\delta)}{\Delta^2}$ labeled examples and returns a hypothesis $H \in \mathcal{C}$ which minimizes $\mathrm{err}(S, H)$ learns the concept class $\mathcal{C}$ with malicious noise rate $\beta \leq \frac{\varepsilon}{1+\varepsilon} - \Delta$.*

Lower bounds on the number of examples necessary for learning with malicious noise were derived by Cesa-Bianchi et al.

Theorem 3 ([7]) *Let $\Delta > 0$ and $d = \mathrm{VC}(\mathcal{C}) \geq 3$. There is a constant κ such that any algorithm which learns $\mathcal{C}$ with malicious noise rate $\beta = \frac{\varepsilon}{1+\varepsilon} - \Delta$ by requesting a sample and returning a hypothesis $H \in \mathcal{C}$ which minimizes $\mathrm{err}(S, H)$ needs a sample of size at least $m \geq \kappa \frac{\varepsilon d}{\Delta^2}$.*

A general conversion of a learning algorithm for the noise-free model into an algorithm for the malicious noise model was given by Kearns and Li.

Theorem 4 ([9]) *Let $\mathcal{A}$ be a (polynomial-time) learning algorithm which learns concept classes $\mathcal{C}_n$ from $m(\varepsilon, \delta, n)$ noise-free examples, i.e., $\beta = 0$. Then $\mathcal{A}$ can be converted into a (polynomial-time) learning algorithm for $\mathcal{C}_n$ for any malicious noise rate $\beta \leq \frac{\log m(\varepsilon/8,1/2,n)}{m(\varepsilon/8,1/2,n)}$.*

The next theorem relates learning with malicious noise to a type of combinatorial optimization problems.

Theorem 5 ([9]) *Let $r \geq 1$ and $\alpha > 0$.*

1. *Let $\mathcal{A}$ be a polynomial-time algorithm which, for any sample S, returns a hypothesis $H \in \mathcal{C}$ with $\mathrm{err}(S, H) \leq r \cdot \min_{C \in \mathcal{C}} \mathrm{err}(S, C)$. Then $\mathcal{A}$ learns concept class $\mathcal{C}$ for any malicious noise rate $\beta \leq \frac{\varepsilon}{(1+\alpha)(1+\varepsilon)r}$ in time polynomial in $1/\epsilon$, $\log 1/\delta$, $\mathrm{VC}(\mathcal{C})$, and $1/\alpha$.*
2. *Let $\mathcal{A}$ be a polynomial-time learning algorithm for concept classes $\mathcal{C}_n$ which tolerates a malicious noise rate $\beta = \frac{\varepsilon}{r}$ and returns a hypothesis $H \in \mathcal{C}_n$. Then $\mathcal{A}$ can be converted into a polynomial-time algorithm which for any sample S, with high probability, returns a hypothesis $H \in \mathcal{C}_n$ such that $\mathrm{err}(S, H) \leq (1 + \alpha)r \cdot \min_{C \in \mathcal{C}} \mathrm{err}(S, C)$.*

The computational hardness of several such related combinatorial optimization problems was shown by Ben-David, Eiron, and Long [3]. Some particular concept classes for which learning with malicious noise has been considered are monomials, CNF and DNF formulas [9, 14], symmetric functions and decision lists [9], multiple intervals on the real line [7], and halfspaces [11].

Applications

Several extensions of the learning model with malicious noise have been proposed, in particular the agnostic learning model [10] and the statistical query model [1]. The following relations between these models and the malicious noise model have been established:

Theorem 6 ([10]) *If concept class $\mathcal{C}$ is polynomial-time learnable in the agnostic model, then $\mathcal{C}$ is polynomial-time learnable with any malicious noise rate $\beta \leq \varepsilon/2$.*

Theorem 7 ([1]) *If $\mathcal{C}$ is learnable from (relative error) statistical queries, then $\mathcal{C}$ is learnable with any malicious noise rate $\beta \leq \varepsilon/\log^p(1/\varepsilon)$ for a suitable large p independent of $\mathcal{C}$.*

Another learning model related to the malicious noise model is learning with nasty noise [6]. In this model examples affected by malicious noise are not chosen at random with probability β, but an adversary might manipulate an arbitrary fraction of βm examples out of a given sample of size m. The malicious noise model was also considered in the context of online learning [2] and boosting [12]. A variant of the malicious noise model for unsupervised learning has been investigated in [5]. In this model, noisy data points are again replaced by arbitrary points possibly generated by an adversary. Still, a correct clustering of the points can learned if the noise rate is moderate.

Cross-References

- Performance-Driven Clustering
- Learning DNF Formulas
- Online Learning and Optimization
- PAC Learning
- Statistical Query Learning

Recommended Reading

1. Aslam JA, Decatur SE (1998) Specification and simulation of statistical query algorithms for efficiency and noise tolerance. J Comput Syst Sci 56: 191–208

2. Auer P, Cesa-Bianchi N (1998) On-line learning with malicious noise and the closure algorithm. Ann Math Artif Intell 23:83–99
3. Ben-David S, Eiron N, Long P (2003) On the difficulty of approximately maximizing agreements. J Comput Syst Sci 66:496–514
4. Blumer A, Ehrenfeucht A, Haussler D, Warmuth M (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM 36:929–965
5. Brubaker SC (2009) Robust PCA and clustering in noisy mixtures. In: Proceedings of 20th annual ACM-SIAM symposium on discrete algorithms, New York, pp 1078–1087
6. Bshouty N, Eiron N, Kushilevitz E (2002) PAC learning with nasty noise. Theor Comput Sci 288: 255–275
7. Cesa-Bianchi N, Dichterman E, Fischer P, Shamir E, Simon HU (1999) Sample-efficient strategies for learning in the presence of noise. J ACM 46: 684–719
8. Kearns M, Li M (1988) Learning in the presence of malicious errors. In: Proceedings of 20th ACM symposium on theory of computing, Chicago, pp 267–280
9. Kearns M, Li M (1993) Learning in the presence of malicious errors. SIAM J Comput 22:807–837
10. Kearns M, Schapire R, Sellie L (1994) Toward efficient agnostic learning. Mach Learn 17: 115–141
11. Klivans AR, Long PM, Servedio RA (2009) Learning halfspaces with malicious noise. Journal of Mach Learn Res 10:2715–2740
12. Servedio RA (2003) Smooth boosting and learning with malicious noise. Journal of Mach Learn Res 4:633–648
13. Valiant L (1984) A theory of the learnable. Commun ACM 27:1134–1142
14. Valiant L Learning (1985) disjunctions of conjunctions. In: Proceedings of 9th international joint conference on artificial intelligence, Los Angeles, pp 560–566

Learning with the Aid of an Oracle

Christino Tamon
Department of Computer Science, Clarkson University, Potsdam, NY, USA

Keywords

Boolean circuits; Disjunctive normal form; Exact learning via queries

Years and Authors of Summarized Original Work

1996; Bshouty, Cleve, Gavaldà, Kannan, Tamon

Problem Definition

In the exact learning model of Angluin [2], a learning algorithm A must discover an unknown function $\mathsf{f} : \{0,1\}^n \to \{0,1\}$ that is a member of a known class C of Boolean functions. The learning algorithm can make at least one of the following types of queries about f:

- Equivalence query $\mathsf{EQ_f(g)}$, for a candidate function g:
  The reply is either "yes," if $\mathsf{g} \Leftrightarrow \mathsf{f}$, or a counterexample a with $\mathsf{g}(a) \neq \mathsf{f}(a)$, otherwise.
- Membership query $\mathsf{MQ_f}(a)$, for some $a \in \{0,1\}^n$:
  The reply is the Boolean value $\mathsf{f}(a)$.
- Subset query $\mathsf{SubQ_f(g)}$, for a candidate function g:
  The reply is "yes," if $\mathsf{g} \Rightarrow \mathsf{f}$, or a counterexample a with $\mathsf{f}(a) < \mathsf{g}(a)$, otherwise.
- Superset query $\mathsf{SupQ_f(g)}$, for a candidate function g:
  The reply is "yes," if $\mathsf{f} \Rightarrow \mathsf{g}$, or a counterexample a with $\mathsf{g}(a) < \mathsf{f}(a)$, otherwise.

A disjunctive normal formula (DNF) is a depth-2 OR-AND circuit whose size is given by the number of its AND gates. Likewise, a conjunctive normal formula (CNF) is a depth-2 AND-OR circuit whose size is given by the number of its OR gates. Any Boolean function can be represented as both a DNF or a CNF formula. A k-DNF is a DNF where each AND gate has a fan-in of at most k; similarly, we may define a k-CNF.

Problem For a given class C of Boolean functions, such as polynomial-size Boolean circuits or disjunctive normal form (DNF) formulas, the goal is to design polynomial-time

learning algorithms for any unknown $\mathsf{f} \in \mathsf{C}$ and ask a polynomial number of queries. The output of the learning algorithm should be a function g of polynomial size satisfying $g \Leftrightarrow f$. The polynomial functions bounding the running time, query complexity, and output size are defined in terms of the number of inputs n and the size of the smallest representation (Boolean circuit or DNF) of the unknown function f.

Key Results

One of the main results proved in [5] is that Boolean circuits and disjunctive normal formulas are exactly learnable using equivalence queries and access to an NP oracle.

Theorem 1 *The following tasks can be accomplished with probabilistic polynomial-time algorithms that have access to an* NP *oracle and make polynomially many equivalence queries:*

- *Learning* DNF *formulas of size s using equivalence queries that are depth-3* AND-OR-AND *formulas of size $O(sn^2/\log^2 n)$.*
- *Learning Boolean circuits of size s using equivalence queries that are circuits of size $O(sn + n\log n)$.*

The idea behind this result is simple. Any class C of Boolean functions is exactly learnable with equivalence queries using the Halving algorithm of Littlestone [11]. This algorithm asks equivalence queries that are the *majority* of candidate functions from C. These are functions in C that are consistent with the counterexamples obtained so far by the learning algorithm. Since each such majority query eliminates at least half of the candidate functions, $\log_2 |\mathsf{C}|$ equivalence queries are sufficient to learn any function in C. A problem with using the Halving algorithm here is that the majority query has exponential size. But, it can be shown that a majority of a polynomial number of uniformly random candidate functions is a good enough approximator to the majority of all candidate functions. Moreover, with access to an NP oracle, there is a randomized polynomial time algorithm for generating random uniform candidate functions due to Jerrum, Valiant, and Vazirani [7]. This yields the result.

The next observation is that subset and superset queries are apparently powerful enough to simulate both equivalence queries and the NP oracle. This is easy to see since the tautology test $\mathsf{g} \Leftrightarrow 1$ is equivalent to $\mathsf{SubQ_f}(\overline{\mathsf{g}}) \wedge \mathsf{SubQ_f}(\mathsf{g})$, for any unknown function f; and, $\mathsf{EQ_f}(\mathsf{g})$ is equivalent to $\mathsf{SubQ_f}(\mathsf{g}) \wedge \mathsf{SupQ_f}(\mathsf{g})$. Thus, the following generalization of Theorem 1 is obtained.

Theorem 2 *The following tasks can be accomplished with probabilistic polynomial-time algorithms that make polynomially many subset and superset queries:*

- *Learning* DNF *formulas of size s using equivalence queries that are depth-3* AND-OR-AND *formulas of size $O(sn^2/\log^2 n)$.*
- *Learning Boolean circuits of size s using equivalence queries that are circuits of size $O(sn + n\log n)$.*

Stronger deterministic results are obtained by allowing more powerful complexity-theoretic oracles. The first of these results employ techniques developed by Sipser and Stockmeyer [12, 13].

Theorem 3 *The following tasks can be accomplished with deterministic polynomial-time algorithms that have access to an Σ_3^p oracle and make polynomially many equivalence queries:*

- *Learning* DNF *formulas of size s using equivalence queries that are depth-3* AND-OR-AND *formulas of size $O(sn^2/\log^2 n)$.*
- *Learning Boolean circuits of size s using equivalence queries that are circuits of size $O(sn + n\log n)$.*

In the following result, C is an infinite class of functions containing functions of the form $\mathsf{f} : \{0,1\}^\star \to \{0,1\}$. The class C is p-evaluatable if the following tasks can be performed in polynomial time:

- Given y, is y a valid representation for any function $\mathsf{f}_y \in \mathsf{C}$?
- Given a valid representation y and $x \in \{0,1\}^\star$, is $\mathsf{f}_y(x) = 1$?

Theorem 4 *Let* C *be any p-evaluatable class. The following statements are equivalent:*

- C *is learnable from polynomially many equivalence queries of polynomial size (and unlimited computational power).*
- C *is learnable in deterministic polynomial time with equivalence queries and access to a* Σ_5^p *oracle.*

For exact learning with membership queries, the following results are proved.

Theorem 5 *The following tasks can be accomplished with deterministic polynomial-time algorithms that have access to an* NP *oracle and make polynomially many membership queries (in n,* DNF *and* CNF *sizes of* f*, where* f *is the unknown function):*

- *Learning monotone Boolean functions.*
- *Learning* $O(\log n)$-CNF $\bigcap$ $O(\log n)$-DNF.

The ideas behind the above result use techniques from [2,4]. For a monotone Boolean function f, the standard closure algorithm uses both equivalence and membership queries to learn f using candidate functions g satisfying $\mathsf{g} \Rightarrow \mathsf{f}$. The need for membership can be removed using the following observation. Viewing $\neg\mathsf{f}$ as a monotone function on the inverted lattice, we can learn f and $\neg\mathsf{f}$ simultaneously using candidate functions g, h, respectively, that satisfy $\mathsf{g} \Rightarrow \mathsf{h}$. The NP oracle is used to obtain an example a that either helps in learning f or in learning $\neg\mathsf{f}$; when no such example can be found, we have learned f.

Theorem 6 *Any class* C *of Boolean functions that is exactly learnable using a polynomial number of membership queries (and unlimited computational power) is exactly learnable in expected polynomial time using a polynomial number of membership queries and access to an* NP *oracle.*

Moreover, any p-evaluatable class C *that is exactly learnable from a polynomial number of membership queries (and unlimited computational power) is also learnable in deterministic polynomial time using a polynomial number of membership queries and access to a* Σ_5^p *oracle.*

Theorems 4 and 6 showed that information-theoretic learnability using equivalence and membership queries can be transformed into computational learnability at the expense of using the Σ_5^p and NP oracles, respectively.

Applications

The learning algorithm for Boolean circuits using equivalence queries and access to an NP oracle has found an application in complexity theory. Watanabe (see [10]) showed an improvement on a known theorem of Karp and Lipton [8]: if NP has polynomial-size circuits, then the polynomial-time hierarchy PH collapses to $\mathsf{ZPP}^{\mathsf{NP}}$. Subsequently, Aaronson (see [1]) showed that queries to the NP oracle used in the learning algorithm (for Boolean circuits) cannot be parallelized by any relativizing techniques.

Some techniques developed in Theorem 5 for exact learning using membership queries of monotone Boolean functions have found applications in data mining [6].

Open Problems

It is unknown if there are polynomial-time learning algorithms for Boolean circuits and DNF formulas using equivalence queries (without complexity-theoretic oracles). There are strong cryptographic evidence that Boolean circuits are not learnable in polynomial-time

(see [3] and the references therein). The best running time for learning DNF formulas is $2^{\tilde{O}(n^{1/3})}$ as given by Klivans and Servedio [9]. It is unclear if membership queries help in this case.

Cross-References

For related learning results, see
▸ Learning DNF Formulas
▸ Learning Automata

Recommended Reading

1. Aaronson S (2006) Oracles are subtle but not malicious. In: Proceedings of the 21st annual IEEE conference on computational complexity (CCC'06), Prague, pp 340–354
2. Angluin D (1988) Queries and concept learning. Mach Learn 2:319–342
3. Angluin D, Kharitonov M (1995) When Won't Membership Queries Help? J Comput Syst Sci 50:336–355
4. Bshouty NH (1995) Exact learning boolean function via the monotone theory. Inf Comput 123:146–153
5. Bshouty NH, Cleve R, Gavaldà R, Kannan S, Tamon C (1996) Oracles and queries that are sufficient for exact learning. J Comput Syst Sci 52(3):421–433
6. Gunopolous D, Khardon R, Mannila H, Saluja S, Toivonen H, Sharma RS (2003) Discovering all most specific sentences. ACM Trans Database Syst 28:140–174
7. Jerrum MR, Valiant LG, Vazirani VV (1986) Random generation of combinatorial structures from a uniform distribution. Theor Comput Sci 43:169–188
8. Karp RM, Lipton RJ (1980) Some connections between nonuniform and uniform complexity classes. In: Proceedings of the 12th annual ACM symposium on theory of computing, Los Angeles, pp 302–309
9. Klivans AR, Servedio RA (2004) Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. J Comput Syst Sci 68:303–318
10. Köbler J, Watanabe O (1998) New collapse consequences of np having small circuits. SIAM J Comput 28:311–324
11. Littlestone N Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. Mach Learn 2:285–318 (1987)
12. Sipser M (1983) A complexity theoretic approach to randomness. In: Proceedings of the 15th annual ACM symposium on theory of computing, Boston, pp 330–334
13. Stockmeyer LJ (1985) On approximation algorithms for $\#P$. SIAM J Comput 14:849–861

LEDA: a Library of Efficient Algorithms

Christos Zaroliagis
Department of Computer Engineering and Informatics, University of Patras, Patras, Greece

Keywords

LEDA platform for combinatorial and geometric computing

Years and Authors of Summarized Original Work

1995; Mehlhorn, Näher

Problem Definition

In the last forty years, there has been a tremendous progress in the field of computer algorithms, especially within the core area known as *combinatorial algorithms*. Combinatorial algorithms deal with objects such as lists, stacks, queues, sequences, dictionaries, trees, graphs, paths, points, segments, lines, convex hulls, etc, and constitute the basis for several application areas including network optimization, scheduling, transport optimization, CAD, VLSI design, and graphics. For over thirty years, asymptotic analysis has been the main model for designing and assessing the efficiency of combinatorial algorithms, leading to major algorithmic advances.

Despite so many breakthroughs, however, very little had been done (at least until 15 years ago) about the practical utility and assessment of this wealth of theoretical work. The main reason for this lack was the absence of a standard *algorithm library*, that is, of a software library that contains a systematic collection of robust and efficient implementations of algorithms and data structures, upon which other algorithms and data structures can be easily built.

The lack of an algorithm library limits severely the great impact which combinatorial algorithms can have. The continuous

re-implementation of basic algorithms and data structures slows down progress and typically discourages people to make the (additional) effort to use an efficient solution, especially if such a solution cannot be re-used. This makes the migration of scientific discoveries into practice a very slow process.

The major difficulty in building a library of combinatorial algorithms stems from the fact that such algorithms are based on complex data types, which are typically not encountered in programming languages (i.e., they are not built-in types). This is in sharp contrast with other computing areas such as statistics, numerical analysis, and linear programming.

Key Results

The currently most successful algorithm library is LEDA (Library for Efficient Data types and Algorithms) [4, 5]. It contains a very large collection of advanced data structures and algorithms for combinatorial and geometric computing. The development of LEDA started in the early 1990s, it reached a very mature state in the late 1990s, and it continues to grow. LEDA has been written in C++ and has benefited considerably from the object-oriented paradigm.

Four major goals have been set in the design of LEDA.

1. *Ease of use*: LEDA provides a sizable collection of data types and algorithms in a form that they can be readily used by non-experts. It gives a precise and readable specification for each data type and algorithm, which is short, general and abstract (to hide the details of implementation). Most data types in LEDA are parameterized (e.g., the dictionary data type works for arbitrary key and information type). To access the objects of a data structure by position, LEDA has invented the *item concept* that casts positions into an abstract form.
2. *Extensibility*: LEDA is easily extensible by means of parametric polymorphism and can be used as a platform for further software development. Advanced data types are built on top of basic ones, which in turn rest on a uniform conceptual framework and solid implementation principles. The main mechanism to extend LEDA is through the so-called LEDA extension packages (LEPs). A LEP extends LEDA into a particular application domain and/or area of algorithms that is not covered by the core system. Currently, there are 15 such LEPs; for details see [1].
3. *Correctness*: In LEDA, programs should give sufficient justification (proof) for their answers to allow the user of a program to easily assess its correctness. Many algorithms in LEDA are accompanied by *program checkers*. A program checker C for a program P is a (typically very simple) program that takes as input the input of P, the output of P, and perhaps additional information provided by P, and verifies that the answer of P in indeed the correct one.
4. *Efficiency*: The implementations in LEDA are usually based on the asymptotically most efficient algorithms and data structures that are known for a problem. Quite often, these implementations have been fine-tuned and enhanced with heuristics that considerably improve running times. This makes LEDA not only the most comprehensive platform for combinatorial and geometric computing, but also a library that contains the currently fastest implementations.

Since 1995, LEDA is maintained by the Algorithmic Solutions Software GmbH [1] which is responsible for its distribution in academia and industry.

Other efforts for algorithm libraries include the Standard Template Library (STL) [7], the Boost Graph Library [2, 6], and the Computational Geometry Algorithms Library (CGAL) [3].

STL [7] (introduced in 1994) is a library of interchangeable components for solving many fundamental problems on sequences of elements, which has been adopted into the C++ standard. It contributed the *iterator concept* which provides an interface between *containers* (an object that stores other objects) and algorithms. Each

algorithm in STL is a function template parameterized by the types of iterators upon which it operates. Any iterator that satisfies a minimum set of requirements can be used regardless of the data structure accessed by the iterator. The systematic approach used in STL to build abstractions and interchangeable components is called *generic programming*.

The Boost Graph Library [2, 6] is a C++ graph library that applies the notions of generic programming to the construction of graph algorithms. Each graph algorithm is written not in terms of a specific data structure, but instead in terms of a graph abstraction that can be easily implemented by many different data structures. This offers the programmer the flexibility to use graph algorithms in a wide variety of applications. The first release of the library became available in September 2000.

The Computational Geometry Algorithms Library [3] is another C++ library that focuses on geometric computing only. Its main goal is to provide easy access to efficient and reliable geometric algorithms to users in industry and academia. The CGAL library started in 1996 and the first release was in April 1998.

Among all libraries mentioned above LEDA is by far the best (both in quality and efficiency of implementations) regarding combinatorial computing. It is worth mentioning that the late versions of LEDA have also incorporated the iterator concept of STL.

Finally, a notable effort concerns the Stony Brook Algorithm Repository [8]. This is not an algorithm library, but a comprehensive collection of algorithm implementations for over seventy problems in combinatorial computing, started in 2001. The repository features implementations coded in different programming languages, including C, C++, Java, Fortran, ADA, Lisp, Mathematic, and Pascal.

Applications

An algorithm library for combinatorial and geometric computing has a wealth of applications in a wide variety of areas, including: network optimization, scheduling, transport optimization and control, VLSI design, computer graphics, scientific visualization, computer aided design and modeling, geographic information systems, text and string processing, text compression, cryptography, molecular biology, medical imaging, robotics and motion planning, and mesh partition and generation.

Open Problems

Algorithm libraries usually do not provide an interactive environment for developing and experimenting with algorithms. An important research direction is to add an interactive environment into algorithm libraries that would facilitate the development, debugging, visualization, and testing of algorithms.

Experimental Results

There are numerous experimental studies based on LEDA, STL, Boost, and CGAL, most of which can be found in the world-wide web. Also, the web sites of some of the libraries contain pointers to experimental work.

URL to Code

The afore mentioned algorithm libraries can be downloaded from their corresponding web sites, the details of which are given in the bibliography (Recommended Reading).

Cross-References

- ▸ Engineering Algorithms for Large Network Applications
- ▸ Experimental Methods for Algorithm Analysis
- ▸ Implementation Challenge for Shortest Paths
- ▸ Shortest Paths Approaches for Timetable Information
- ▸ TSP-Based Curve Reconstruction

Recommended Reading

1. Algorithmic Solutions Software GmbH, http://www.algorithmic-solutions.com/. Accessed Feb 2008
2. Boost C++ Libraries, http://www.boost.org/. Accessed Feb 2008
3. CGAL: Computational Geometry Algorithms Library, http://www.cgal.org/. Accessed Feb 2008
4. Mehlhorn K, Näher S (1995) LEDA: a platform for combinatorial and geometric computing. Commun ACM 38(1):96–102
5. Mehlhorn K, Näher S (1999) LEDA: a platform for combinatorial and geometric computing. Cambridge University Press, Boston
6. Siek J, Lee LQ, Lumsdaine A (2002) The boost graph library. Addison-Wesley, Cambridge
7. Stepanov A, Lee M (1994) The standard template library. In: Technical report X3J16/94–0095, WG21/N0482, ISO Programming Language C++ project. Hewlett-Packard, Palo Alto
8. The Stony Brook Algorithm Repository, http://www.cs.sunysb.edu/~algorith/. Accessed Feb 2008

Lempel-Ziv Compression

Simon J. Puglisi
Department of Computer Science, University of Helsinki, Helsinki, Finland

Keywords

Dictionary-based compression; LZ77; LZ78; LZ compression; LZ parsing

Years and Authors of Summarized Original Work

1976; Lempel, Ziv
1977; Ziv, Lempel
1978; Ziv, Lempel

Problem Definition

Lossless data compression is concerned with compactly representing data in a form that allows the original data to be faithfully recovered. Reduction in space can be achieved by exploiting the presence of repetition in the data.

Many of the main solutions for lossless data compression in the last three decades have been based on techniques first described by Ziv and Lempel [22, 33, 34]. These methods gained popularity in the 1980s via tools like Unix `compress` and the GIF image format, and today they pervade computer software, for example, in the zip, gzip, and lzma compression utilities, in modem compression standards V.42bis and V.44, and as a basis for the compression used in information retrieval systems [8] and Google's BigTable [4] database system. Perhaps the primary reason for the success of Lempel and Ziv's methods is their powerful combination of compression effectiveness and compression/decompression throughput. We refer the reader to [3, 29] for a review of related dictionary-based compression techniques.

Key Results

Let $S[1 \ldots n]$ be a string of n symbols drawn from an alphabet Σ. Lempel-Ziv-based compression algorithms work by parsing S into a sequence of substrings called *phrases* (or factors). To achieve compression, each phrase is replaced by a compact representation, as detailed below.

LZ78

Assume the encoder has already parsed the phrases $S_1, S_2, \ldots, S_{i-1}$, that is, $S = S_1 S_2 \ldots S_{i-1} S'$ for some suffix S' of S. The LZ78 [34] dictionary is the set of strings obtained by adding a single symbol to one of the strings S_j or to the empty string. The next phrase S_i is then the longest prefix of S' that is an element of the dictionary. For example, $S = bbaabbbabbabbaab$ has an LZ78 parsing of b, ba, a, bb, bab, $babb$, aa, b. Clearly, all LZ78 phrases will be distinct, except possibly the final one. Let S_0 denote the empty string. If $S_i = S_j\alpha$, where $0 \leq j < i$ and $\alpha \in \Sigma$, the code word emitted by LZ78 for S_i will be the pair (j, α). Thus, if LZ78 parses the string S into y words, its output is bounded by $y \log y + y \log |\Sigma| + \Theta(y)$ bits.

LZ77

The LZ77 parsing algorithm takes a single positive integer parameter w, called the *window size*. Say the encoder has already parsed the phrases $S_1, S_2, \ldots, S_{i-1}$, that is, $S = S_1 S_2 \ldots S_{i-1} S'$ for some suffix S' of S. The next phrase S_i starts at position $q = |S_1 S_2 \ldots S_{i-1}| + 1$ in S and is the shortest prefix of S' that does not have an occurrence starting at any position $q - w \leq p_i < q$ in S. Thus defined, LZ77 phrases have the form $t\alpha$, where t (possibly empty) is the longest prefix of S' that also has an occurrence starting at some position $q - w \leq p_i < q$ in S and α is the symbol $S[q + |t| + 1]$.

The version of LZ77 described above is often called *sliding window* LZ77: a text window of length w that slides along the string during parsing is used to decide the next phrase. In the so-called infinite window LZ77, we enforce that $q - w$ is always equal to 0 – in other words, throughout parsing the window stretches all the way back to the beginning of the string. Infinite window parsing is more powerful than sliding window parsing, and for the remainder of this entry, the term "LZ77" refers to infinite window LZ77, unless explicitly stated.

For $S = bbaabbbabbabbaab$, the infinite window LZ77 parsing is b, ba, ab, $bbab$, $babbaa$, b. Note that phrases are allowed to overlap their definitions, as is the case with phrase S_5 in our example. Like LZ78, all LZ77 phrases are distinct, with the possible exception of the last phrase. It is easy to see that infinite window LZ77 will always produce a smaller number of phrases than LZ78. If $S_i = t\alpha$ with $\alpha \in \Sigma$, the code word for S_i is the triple (p_i, ℓ_i, α), where p_i is the position of a previous occurrence of t in $S_1 S_2 \ldots S_{i-1}$ and $\ell_i = |t|$.

Finally, it is important to note that for a given phrase $S_i = t\alpha$, there is sometimes more than one previous occurrence of t, leading to a choice of p_i value. If $p_i < q$ is the largest possible for every phrase, then we call the parsing *rightmost*. In their study on the bit complexity of LZ compression, Ferragina et al. [9] showed that the rightmost parsing can lead to encodings asymptotically smaller than what is achievable otherwise.

Compression and Decompression Complexity

The LZ78 parsing for a string of length n can be computed in $O(n)$ time by maintaining the phrases in a try. Doing so allows finding in time proportional to its length the longest prefix of the unparsed portion that is in the dictionary, and so the time overall is linear in n. Compression in sublinear time and space is possible using succinct dynamic tries [16]. Decoding is somewhat symmetric – an explicit try is not needed, just the parent pointers implicitly represented in the encoded pairs.

Recovering the original string S from its LZ77 parsing (i.e., decompression) is very easy: (p_i, ℓ_i, α) is decoded by copying the symbols of the substring $S[p_i \ldots p_i + \ell_i - 1]$ and then appending α. By the definition of the parsing, any symbol we need to copy will have already been decoded (if we copy the strings left to right).

Obtaining the LZ77 parsing in the first place is not as straightforward and has been the subject of intense research since LZ77 was published. Indeed, it seems safe to speculate that LZ78 and sliding window LZ77 were invented primarily because it was initially unclear how infinite window LZ77 could be computed efficiently. Today, parsing is possible in worst-case $O(n)$ time, using little more than $n(\log n + \log |\Sigma|)$ bits of space. Current state-of-the-art methods [14, 15, 18, 19] operate offline, combining the suffix array [24] of the input string with data structures for answering next and previous smaller value queries [10]. For online parsing, the current best algorithm uses $O(n \log n)$ time and $O(n \log |\Sigma|)$ bits of space [32].

Compression Effectiveness

It is well known that LZ converges to the entropy of any ergodic source [6, 30, 31]. However, it is also possible to prove compression bounds on LZ-based schemes without probabilistic assumptions on the input, using the notion of *empirical entropy* [25].

Convergence to Empirical Entropy

For any string S, the kth-order empirical entropy $H_k(S)$ is a lower bound on the compression

achievable by any compressor that assigns code words to symbols based on statistics derived from the k letters preceding each symbol in the string. In particular, the output of LZ78 (and so LZ77) is upper-bounded by $|S|H_k(S) + o(|S| \log |\Sigma|)$ bits [20] for any $k = o(\log_{|\Sigma|} n)$.

Relationship to Grammar Compression

The *smallest grammar problem* is the problem of finding the smallest context-free grammar that generates only a given input string S. The size g^* of the smallest grammar is a rather elegant measure of compressibility, and Charikar et al. [5] established that finding it is NP-hard. They also considered several approximation algorithms, including LZ78. The LZ78 parsing of S can be viewed as a context-free grammar in which for each dictionary word $S_i = S_j\alpha$, there is a production rule $X_i = X_j\alpha$. LZ78's approximation ratio is rather bad: $\Omega(n^{2/3}/\log n)$.

Charikar et al. also showed that g^* is at least the number of phrases z of the LZ77 parse of S and used the phrases of the parsing to derive a new grammar compression algorithm with approximation ratio $O(\log(|S|/g^*))$. The same result was discovered contemporaneously by Rytter [28] and later simplified by Jeż [17].

Greedy Versus Non-greedy Parsing

LZ78 and LZ77 are both greedy algorithms: they select, at each step, the longest prefix of the remaining suffix of the input that is in the dictionary. For LZ77, the greedy strategy is optimal in the sense that it yields the minimum number of code words. However, if variable-length codes are used to represent each element of the code word triple, the greedy strategy does not yield an optimal parsing, as Ferragina, Nitto, and Venturini have recently established [9]. For LZ78, greedy parsing does not always produce the minimum number of phrases. Indeed, in the worst case, greedy parsing can produce a factor $O(\sqrt{n})$ more than a simple non-greedy parsing strategy that, instead of choosing the prefix that gives the longest extension in the current iteration, chooses the prefix that gives the longest extension in the next iteration [26]. There are many, many variants on LZ parsing that relax the greedy condition with the aim of reducing the overall encoding size in practice. Several of these non-greedy methods are covered in textbooks (e.g., [3, 29]).

Applications

As outlined at the start of this entry, the major applications of Lempel and Ziv's methods are in the fields of lossless data compression. However, the deep connections of these methods to string data mean the Lempel-Ziv parsing has also found important applications in string processing: the parsing reveals a great deal of information about the repetitive structure of the underlying string, and this can be used to design efficient algorithms and data structures.

Pattern Matching in Compressed Space

A compressed full-text index for a string $S[1 \ldots n]$ is a data structure that takes space proportional to the entropy of S while simultaneously supporting efficient queries over S. The supported queries can be relatively simple, such as random access to symbols of S, or more complex, such as reporting all the occurrences of a pattern $P[1 \ldots m]$ in S.

Arroyuelo et al. [2] describe a compressed index based on LZ78. For any text S, their index uses $(2 + \epsilon)nH_k(S) + o(n \log |\Sigma|)$ bits of space and reports all c occurrences of P in S in $O(m^2 \log m + (m + c) \log n)$ time. Their approach stores two copies of the LZ78 dictionary represented as tries. One try contains the dictionary phrases, and the other contains the reverse phrases. The main trick to pattern matching is to then split the pattern in two (in all m possible ways) and then check for each half in the tries.

Kreft and Navarro [21] describe a compressed index based on the LZ77 parsing. It requires $3z \log n + O(z \log |\Sigma|) + o(n)$ bits of space and supports extraction of ℓ symbols in $O(\ell h)$ time and pattern matching in $O(m^2 h + m \log z + c \log z)$ time, where $h \leq \sqrt{n}$ is the maximum length of a referencing chain in the parsing (a position is copied from another, that one from another, and so on, h times). More recently, Gagie et al. [12] describe a different LZ77-based

index with improved query bounds that takes $O(z \log n \log(n/z))$ bits of space and supports extraction of ℓ symbols in $O(\ell + \log n)$ time and pattern matching in $O(m \log m + c \log \log n)$ time. A related technique called *indexing by kernelization*, which does not support access and restricts the maximum pattern length that can be searched for, has recently emerged as a promising practical approach applicable to highly repetitive data [11]. This technique uses an LZ parsing to identify a subsequence (called a *kernel*) of the text that is guaranteed to contain at least one occurrence of every pattern the text contains. This kernel string is then further processed to obtain an index capable of searching the original text.

A related problem is the compressed matching problem, in which the text and the pattern are given together and the text is compressed. The task here is to perform pattern matching in the compressed text without decompressing it. For LZ-based compressors, this problem was first considered by Amir, Benson, and Farach [1]. Considerable progress has been made since then, and we refer the reader to [13,27] (and to another encyclopedia entry) for an overview of more recent results.

String Alignment

Crochemore, Landau, and Ziv-Ukelson [7] used LZ78 to accelerate sequence alignment: the problem of finding the lowest-cost sequence of edit operations that transforms one string $S[1 \ldots n]$ into another string $T[1 \ldots n]$. Masek and Paterson proposed an $O(n^2/\log n)$ time algorithm that applies when the costs of the edit operations are rational. Crochemore et al.'s method runs in the same time in the worst case, but allows real-valued costs, and obtains an asymptotic speedup when the underlying texts are compressible.

The textbook solution to the string alignment problem runs in $O(n^2)$, using a straightforward dynamic program that computes a matrix $M[1 \ldots n, 1 \ldots n]$. The approach of the faster algorithms is to break the dynamic program matrix into blocks. Masek and Paterson use blocks of uniform size. Crochemore et al. use blocks delineated by the LZ78 parsing, the idea being that whenever they need to solve a block $M[i \ldots i', j \ldots j']$, they can solve it in $O(i' - i + j' - j)$ time by essentially copying their solutions to the previous blocks $M[i \ldots i' - 1, j \ldots j']$ and $M[i \ldots i', j \ldots j' - 1]$. A similar approach was later used to speed up training of hidden Markov models [23].

Open Problems

Ferragina, Nitto, and Venturini [9] provide an algorithm for computing the rightmost LZ77 parsing that takes $O(n + n \log |\Sigma| / \log \log n)$ time and $O(n)$ words of space to process a string of length n. The existence of an $O(n)$ time algorithm independent of the alphabet size is an open problem.

As mentioned above, the size z of the LZ77 parsing is a lower bound on the size g^* of the smallest grammar for a given string. Proving an asymptotic separation between g^* and z (or, alternatively, finding a way to produce grammars of size z) is a problem of considerable theoretical interest.

URLs to Code and Data Sets

The source code of the gzip tool (based on LZ77) is available at http://www.gzip.org, and the related compression library zlib is available at http://www.zlib.net. Source code for the more efficient compressor LZMA is at: http://www.7-zip.org/sdk.html.

Source code for more recent LZ parsing algorithms (developed in the last 2 years) is available at http://www.cs.helsinki.fi/group/pads/. This code includes the current fastest LZ parsing algorithms for both internal and external memory.

The Pizza&Chili Corpus is a frequently used test data set for LZ parsing algorithms; see http://pizzachili.dcc.uchile.cl/repcorpus.html.

Cross-References

- ▸ Approximate String Matching
- ▸ Compressed Suffix Array
- ▸ Grammar Compression

- ▸ Pattern Matching on Compressed Text
- ▸ Suffix Trees and Arrays

Recommended Reading

1. Amir A, Benson G, Farach M (1996) Let sleeping files lie: pattern matching in Z-compressed files. J Comput Syst Sci 52(2):299–307
2. Arroyuelo D, Navarro G, Sadakane K (2012) Stronger Lempel-Ziv based compressed text indexing. Algorithmica 62(1–2):54–101
3. Bell TC, Cleary JG, Witten IH (1990) Text compression. Prentice-Hall, Upper Saddle River
4. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2008) Bigtable: a distributed storage system for structured data. ACM Trans Comp Sys 26(2):1–26
5. Charikar M, Lehman E, Liu D, Panigrahy R, Prabhakaran M, Sahai A, Shelat A (2005) The smallest grammar problem. IEEE Trans Inform Theory 51(7):2554–2576
6. Cover T, Thomas J (1991) Elements of information theory. Wiley, New York
7. Crochemore M, Landau GM, Ziv-Ukelson M (2003) A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM J Comput 32(6):1654–1673
8. Ferragina P, Manzini G (2010) On compressing the textual web. In: Proceedings of the third international conference on web search and web data mining (WSDM) 2010, New York, 4–6 February 2010. ACM, pp 391–400
9. Ferragina P, Nitto I, Venturini R (2013) On the bit-complexity of lempel-Ziv compression. SIAM J Comput 42(4):1521–1541
10. Fischer J, Heun V (2011) Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM J Comput 40(2):465–492
11. Gagie T, Puglisi SJ (2015) Searching and indexing genomic databases via kernelization. Front Bioeng Biotechnol 3(12). doi:10.3389/fbioe.2015.00012
12. Gagie T, Gawrychowski P, Kärkkäinen J, Nekrich Y, Puglisi SJ (2014) LZ77-based self-indexing with faster pattern matching. In: Proceedings of Latin-American symposium on theoretical informatics (LATIN), Montevideo. Lecture notes in computer science, vol 8392. Springer, pp 731–742
13. Gawrychowski P (2013) Optimal pattern matching in LZW compressed strings. ACM Trans Algorithms 9(3):25
14. Goto K, Bannai H (2013) Simpler and faster Lempel Ziv factorization. In: Proceedings of the 23rd data compression conference (DCC), Snowbird, pp 133–142
15. Goto K, Bannai H (2014) Space efficient linear time Lempel-Ziv factorization for small alphabets. In: Proceedings of the 24th data compression conference (DCC), Snowbird, pp 163–172
16. Jansson J, Sadakane K, Sung W (2007) Compressed dynamic tries with applications to LZ-compression in sublinear time and space. In: Proceedings of 27th FSTTCS, Montevideo. Lecture notes in computer science, vol 4855. Springer, New Delhi, pp 424–435
17. Jez A (2014) A really simple approximation of smallest grammar. In: Kulikov AS, Kuznetsov SO, Pevzner PA (eds) Proceedings of 25th annual symposium combinatorial pattern matching (CPM) 2014, Moscow, 16–18 June 2014. Lecture notes in computer science, vol 8486. Springer, pp 182–191
18. Kärkkäinen J, Kempa D, Puglisi SJ (2013) Linear time Lempel-Ziv factorization: simple, fast, small. In: Proceedings of CPM, Bad Herrenalb. Lecture notes in computer science, vol 7922, pp 189–200
19. Kempa D, Puglisi SJ (2013) Lempel-Ziv factorization: simple, fast, practical. In: Zeh N, Sanders P (eds) Proceedings of ALENEX, New Orleans. SIAM, pp 103–112
20. Kosaraju SR, Manzini G (1999) Compression of low entropy strings with lempel-ziv algorithms. SIAM J Comput 29(3):893–911
21. Kreft S, Navarro G (2013) On compressing and indexing repetitive sequences. Theor Comput Sci 483:115–133
22. Lempel A, Ziv J (1976) On the complexity of finite sequences. IEEE Trans Inform Theory 22(1):75–81
23. Lifshits Y, Mozes S, Weimann O, Ziv-Ukelson M (2009) Speeding up hmm decoding and training by exploiting sequence repetitions. Algorithmica 54(3):379–399
24. Manber U, Myers GW (1993) Suffix arrays: a new method for on-line string searches. SIAM J Comput 22(5):935–948
25. Manzini G (2001) An analysis of the Burrows-Wheeler transform. J ACM 48(3):407–430
26. Matias Y, Sahinalp SC (1999) On the optimality of parsing in dynamic dictionary based data compression. In: Proceedings of the tenth annual ACM-SIAM symposium on discrete algorithms, 17–19 January 1999, Baltimore, pp 943–944
27. Navarro G, Tarhio J (2005) LZgrep: a Boyer-Moore string matching tool for Ziv-Lempel compressed text. Softw Pract Exp 35(12):1107–1130
28. Rytter W (2003) Application of Lempel-Ziv factorization to the approximation of grammar-based compression. Theor Comput Sci 302(1–3):211–222
29. Salomon D (2006) Data compression: the complete reference. Springer, New York/Secaucus
30. Sheinwald D (1994) On the Ziv-Lempel proof and related topics. Proc. IEEE 82:866–871
31. Wyner A, Ziv J (1994) The sliding-window Lempel-Ziv algorithm is asymptotically optimal. Proc IEEE 82:872–877
32. Yamamoto J, I T, Bannai H, Inenaga S, Takeda M (2014) Faster compact on-line Lempel-Ziv factorization. In: Proceedings of 31st international symposium on theoretical aspects of computer science (STACS), Lyon. LIPIcs 25, pp 675–686

33. Ziv J, Lempel A (1977) A universal algorithm for sequential data compression. IEEE Trans Inform Theory 23(3):337–343
34. Ziv J, Lempel A (1978) Compression of individual sequences via variable-rate coding. IEEE Trans Inform Theory 24(5):530–536

Leontief Economy Equilibrium

Yinyu Ye
Department of Management Science and Engineering, Stanford University, Stanford, CA, USA

Keywords

Bimatrix game; Competitive exchange; Computational equilibrium; Leontief utility function

Synonyms

Algorithmic game theory; Arrow-debreu market; Max-min utility; Walras equilibrium

Years and Authors of Summarized Original Work

2005; Codenotti, Saberi, Varadarajan, Ye
2005; Ye

Problem Definition

The Arrow-Debreu exchange market equilibrium problem was first formulated by Léon Walras in 1954 [7]. In this problem, everyone in a population of m traders has an initial endowment of a divisible goods and a utility function for consuming all goods – their own and others'. Every trader sells the entire initial endowment and then uses the revenue to buy a bundle of goods such that his or her utility function is maximized. Walras asked whether prices could be set for everyone's goods such that this is possible. An answer was given by Arrow and Debreu in 1954 [1] who showed that, under mild conditions, such equilibrium would exist if the utility functions were concave. In general, it is unknown whether or not an equilibrium can be computed efficiently; see, e.g., ▸ General Equilibrium.

Consider a special class of Arrow-Debreu's problems, where each of the n traders has exactly one unit of a divisible and distinctive good for trade, and let trader i, $i = 1, \ldots, n$, bring good i, where the class of problems is called the *pairing class*. For given prices p_j on good j, consumer i's maximization problem is

$$\begin{array}{rl} \text{maximize} & u_i(x_{i1}, \ldots, x_{in}) \\ \text{subject to} & \sum_j p_j x_{ij} \le p_i, \\ & x_{ij} \ge 0, \quad \forall j, \end{array} \tag{1}$$

where x_{ij} is the quantity of good j purchased by trader i. Let x_i^* denote a maximal solution vector of (1). Then, vector p is called the Arrow-Debreu price equilibrium if there exists an x_i^* for consumer i, $i = 1, \ldots, n$, to clear the market, that is,

$$\sum_i x_i^* = e,$$

where e is the vector of all ones representing available goods on the exchange market.

The Leontief economy equilibrium problem is the Arrow-Debreu equilibrium problem when the utility functions are in the Leontief form:

$$u_i(x_i) = \min_{j:\, h_{ij} > 0} \left\{ \frac{x_{ij}}{h_{ij}} \right\},$$

where the Leontief coefficient matrix is given by

$$H = \begin{pmatrix} h_{11} & h_{12} & \ldots & h_{1n} \\ h_{21} & h_{22} & \ldots & h_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ h_{n1} & h_{n2} & \ldots & h_{nn} \end{pmatrix}. \tag{2}$$

Here, one may assume that

Assumption 1 *H has no all-zero row, that is, every trader likes at least one good.*

Key Results

Let u_i be the equilibrium utility value of consumer i and p_i be the equilibrium price for good i, $i = 1, \ldots, n$. Also, let U and P be diagonal matrices whose diagonal entries are u_i's and p_i's, respectively. Then, the Leontief economy equilibrium $p \in R^n$, together with $u \in R^n$, must satisfy

$$\begin{aligned} UHp &= p, \\ P(e - H^T u) &= 0, \\ H^T u &\le e, \\ u,\ p &\ge 0, \\ p &\ne 0. \end{aligned} \tag{3}$$

One can prove:

Theorem 1 (Ye [8]) *System (3) always has a solution $(u \ne 0, p)$ under Assumption 1 (i.e., H has no all-zero row). However, a solution to System (3) may not be a Leontief equilibrium, although every Leontief equilibrium satisfies System (3).*

A solution to System (3) is called a quasi-equilibrium. For example,

$$H^T = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

has a quasi-equilibrium $p^T = (1,\ 0,\ 0)$ and $u^T = (1,\ 0,\ 0)$, but it is not an equilibrium. This is because that trader 3, although with zero budget, can still purchase goods 2 and 3 at zero prices. In fact, check if H has an equilibrium that is an NP-hard problem; see discussion later. However, under certain sufficient conditions, e.g., all entries in H are positive, every quasi-equilibrium is an equilibrium.

Theorem 2 (Ye [8]) *Let $B \subset \{1, 2, \ldots, n\}$, $N = \{1, 2, \ldots, n\} \setminus B$, H_{BB} be irreducible, and u_B satisfy the linear system*

$$H_{BB}^T u_B = e, \quad H_{BN}^T u_B \le e, \quad \text{and} \quad u_B > 0.$$

Then the (right) Perron-Frobenius eigenvector p_B of $U_B H_{BB}$ together with $p_N = 0$ will be a solution to System (3). And the converse is also true. Moreover, there is always a rational solution for every such B, that is, the entries of price vector are rational numbers, if the entries of H are rational. Furthermore, the size (bit length) of the solution is bounded by the size (bit length) of H.

The theorem implies that the traders in block B can trade among themselves and keep others goods "free." In particular, if one trader likes his or her own good more than any other good, that is, $h_{ii} \ge h_{ij}$ for all j, then $u_i = 1/h_{ii}$, $p_i = 1$, and $u_j = p_j = 0$ for all $j \ne i$, that is, $B = \{i\}$, makes a Leontief economy equilibrium. The theorem thus establishes, for the first time, a combinatorial algorithm to compute a Leontief economy equilibrium by finding a right block $B \ne \emptyset$, which is actually a nontrivial solution $(u \ne 0)$ to an LCP problem

$$H^T u + v = e,\ u^T v = 0,\ 0 \ne u, v \ge 0. \tag{4}$$

If $H > 0$, then any complementary solution $u \ne 0$, together with its support $B = \{j : u_j > 0\}$, of (4) induce a Leontief economy equilibrium that is the (right) Perron-Frobenius eigenvector of $U_B H_{BB}$, and it can be computed in polynomial time by solving a linear equation. Even if $H \ngtr 0$, any complementary solution $u \ne 0$ and $B = \{j : u_j > 0\}$, as long as H_{BB} is irreducible, induces an equilibrium for System (3). The equivalence between the pairing Leontief economy model and the LCP also implies

Corollary 1 *LCP (4) always has a nontrivial solution $u \ne 0$, where H_{BB} is irreducible with $B = \{j : u_j > 0\}$, under Assumption 1 (i.e., H has no all-zero row).*

If Assumption 1 does not hold, the corollary may not be true; see example below:

$$H^T = \begin{pmatrix} 0 & 2 \\ 0 & 1 \end{pmatrix}.$$

Applications

Given an arbitrary bimatrix game, specified by a pair of $n \times m$ matrices A and B, with positive entries, one can construct a Leontief exchange economy with $n + m$ traders and $n + m$ goods as follows. In words, trader i comes to the market with one unit of good i, for $i = 1, \ldots, n + m$. Traders indexed by any $j \in \{1, \ldots, n\}$ receive some utility only from goods $j \in \{n+1, \ldots, n+m\}$, and this utility is specified by parameters corresponding to the entries of the matrix B. More precisely the proportions in which the j-th trader wants the goods are specified by the entries on the jth row of B. Vice versa, traders indexed by any $j \in \{n + 1, \ldots, n + m\}$ receive some utility only from goods $j \in \{1, \ldots, n\}$. In this case, the proportions in which the j-th trader wants the goods are specified by the entries on the j-th column of A.

In the economy above, one can partition the traders in two groups, which bring to the market disjoint sets of goods and are only interested in the goods brought by the group they do not belong to.

Theorem 3 (Codenotti et al. [4]) *Let (A, B) denote an arbitrary bimatrix game, where one assumes, w.l.o.g., that the entries of the matrices A and B are all positive. Let*

$$H^T = \begin{pmatrix} 0 & A \\ B^T & 0 \end{pmatrix}$$

describe the Leontief utility coefficient matrix of the traders in a Leontief economy. There is a one-to-one correspondence between the Nash equilibria of the game (A, B) and the market equilibria H of the Leontief economy. Furthermore, the correspondence has the property that a strategy is played with positive probability at a Nash equilibrium if and only if the good held by the corresponding trader has a positive price at the corresponding market equilibrium.

The theorem implies that finding an equilibrium for Leontief economies is at least as hard as finding a Nash equilibrium for two-player nonzero sum games, a problem recently proven PPAD-complete (Chen and Deng [3]), where no polynomial time approximation algorithm is known today.

Furthermore, Gilboa and Zemel [6] proved a number of hardness results related to the computation of Nash equilibria (NE) for finite games in normal form. Since the NE for games with more than two players can be irrational, these results have been formulated in terms of NP-hardness for multiplayer games, while they can be expressed in terms of NP-completeness for two-player games. Using a reduction to the NE game, Codenotti et al. proved:

Theorem 4 (Codenotti et al. [4]) *It is NP-hard to decide whether a Leontief economy H has an equilibrium.*

On the positive side, Zhu et al. [9] recently proved the following result:

Theorem 5 *Let the Leontief utility matrix H be symmetric and positive. Then there is a fully polynomial time approximation scheme (FPTAS) for approximating a Leontief equilibrium, although the equilibrium set remains non-convex or non-connected.*

Cross-References

- Approximations of Bimatrix Nash Equilibria
- Complexity of Bimatrix Nash Equilibria
- General Equilibrium
- Non-approximability of Bimatrix Nash Equilibria

Recommended Reading

The reader may want to read Brainard and Scarf [2] on how to compute equilibrium prices in 1891; Chen and Deng [2] on the most recent hardness result of computing the bimatrix game; Cottle et al. [5] for literature on linear complementarity problems; and all references listed in [4] and [8] for the recent literature on computational equilibrium.

1. Arrow KJ, Debreu G (1954) Existence of an equilibrium for competitive economy. Econometrica 22:265–290

2. Brainard WC, Scarf HE (2000) How to compute equilibrium prices in 1891. Cowles Foundation Discussion Paper 1270
3. Chen X, Deng X (2005) Settling the complexity of 2-player Nash-equilibrium. ECCC TR05-140
4. Codenotti B, Saberi A, Varadarajan K, Ye Y (2006) Leontief economies encode nonzero sum two-player games. In: Proceedings SODA, Miami
5. Cottle R, Pang JS, Stone RE (1992) The linear complementarity problem. Academic, Boston
6. Gilboa I, Zemel E (1989) Nash and correlated equilibria: some complexity considerations. Games Econ Behav 1:80–93
7. Walras L (1954) [1877] Elements of pure economics. Irwin. ISBN 0-678-06028-2
8. Ye Y (2005) Exchange market equilibria with Leontief's utility: freedom of pricing leads to rationality. In: Proceedings WINE, Hong Kong
9. Zhu Z, Dang C, Ye Y (2012) A FPTAS for computing a symmetric Leontief competitive economy equilibrium. Math Program 131:113–129

LexBFS, Structure, and Algorithms

Nicolas Trotignon
Laboratoire de l'Informatique du Parallélisme (LIP), CNRS, ENS de Lyon, Lyon, France

Keywords

Classes of graphs; LexBFS; Truemper configurations; Vertex elimination ordering

Years and Authors of Summarized Original Work

2014; Aboulker, Charbit, Trotignon, Vušković

Problem Definition

We provide a general method to prove the existence and compute efficiently elimination orderings in graphs. Our method relies on several tools that were known before but that were not put together so far: the algorithm LexBFS due to Rose, Tarjan, and Lueker, one of its properties discovered by Berry and Bordat, and a local decomposition property of graphs discovered by Maffray, Trotignon, and Vušković.

Terminology

In this paper, all graphs are finite and simple. A graph G *contains* a graph F if F is isomorphic to an induced subgraph of G. A class of graphs is *hereditary* if for every graph G of the class, all induced subgraphs of G belong to the class. A graph G is *F-free* if it does not contain F. When $\mathcal{F}$ is a set of graphs, G is *$\mathcal{F}$-free* if it is F-free for every $F \in \mathcal{F}$. Clearly every hereditary class of graphs is equal to the class of $\mathcal{F}$-free graphs for some $\mathcal{F}$ ($\mathcal{F}$ can be chosen to be the set of all graphs not in the class but all induced subgraphs of which are in the class). The induced subgraph relation is not a well quasi order (contrary, e.g., to the minor relation), so the set $\mathcal{F}$ does not need to be finite.

When $X \subseteq V(G)$, we write $G[X]$ for the subgraph of G induced by X. An ordering $(v_1, \ldots, v_n)$ of the vertices of a graph G is an *$\mathcal{F}$-elimination ordering* if for every $i = 1, \ldots, n$, $N_{G[\{v_1,\ldots,v_i\}]}(v_i)$ is $\mathcal{F}$-free. Note that this is equivalent to the existence, in every induced subgraph of G, of a vertex whose neighborhood is $\mathcal{F}$-free.

Example

Let us illustrate our terminology on a classical example. We denote by S_2 the independent graph on two vertices. A vertex is *simplicial* if its neighborhood is S_2-free, or equivalently is a clique. A graph is *chordal* if it is hole-free, where a *hole* is a chordless cycle of length at least 4.

Theorem 1 (Dirac [6]) *Every chordal graph admits an $\{S_2\}$-elimination ordering.*

Theorem 2 (Rose, Tarjan, and Lueker [14]) *There exists a linear-time algorithm that computes an $\{S_2\}$-elimination ordering of an input chordal graph.*

LexBFS

To explain the results, we need to define LexBFS. It is a linear time algorithm of Rose, Tarjan, and Lueker [14] whose input is any graph G together with a vertex s and whose output is a linear

ordering of the vertices of G starting at s. A linear ordering of the vertices of a graph G is a *LexBFS ordering* if there exists a vertex s of G such that the ordering can be produced by LexBFS when the input is G, s. The order from Theorem 2 is in fact computed by LexBFS. We do not need here to define LexBFS more precisely, because the following result fully characterizes LexBFS orderings.

Theorem 3 (Brandstädt, Dragan, and Nicolai [3]) *An ordering $\prec$ of the vertices of a graph $G = (V, E)$ is a LexBFS ordering if and only if it satisfies the following property: for all $a, b, c \in V$ such that $c \prec b \prec a$, $ca \in E$ and $cb \notin E$ there exists a vertex d in G such that $d \prec c$, $db \in E$ and $da \notin E$.*

Key Results

The following property was introduced by Maffray, Trotignon, and Vušković in [12] (where it was called Property $(\star)$).

Definition 1 Let $\mathcal{F}$ be a set of graphs. A graph G is *locally $\mathcal{F}$-decomposable* if for every vertex v of G, every $F \in \mathcal{F}$ contained in $N(v)$, and every connected component C of $G - N[v]$, there exists $y \in F$ such that y has a non-neighbor in F and no neighbors in C. A class of graphs $\mathcal{C}$ is *locally $\mathcal{F}$-decomposable* if every graph $G \in \mathcal{C}$ is locally $\mathcal{F}$-decomposable.

It is easy to see that if a graph is locally $\mathcal{F}$-decomposable, then so are all its induced subgraphs. Therefore, for all sets of graphs $\mathcal{F}$, the class of graphs that are locally $\mathcal{F}$-decomposable is hereditary. The main result is the following.

Theorem 4 *If $\mathcal{F}$ is a set of non-complete graphs, and G is a locally $\mathcal{F}$-decomposable graph, then every LexBFS ordering of G is an $\mathcal{F}$-elimination ordering.*

First Example of Application

Let us now illustrate how Theorem 4 can be used with the simplest possible set made of non-complete graphs $\mathcal{F} = \{S_2\}$, where S_2 is the independent graph on two vertices. The following is well known and easy to prove.

Lemma 1 *A graph G is locally $\{S_2\}$-decomposable if and only if G is chordal.*

Hence, a proof for Theorems 1 and 2 is easily obtained by using Lemma 1 and Theorem 4.

Sketch of Proof

The proof of Theorem 4 relies mainly on the following.

Theorem 5 (Berry and Bordat [2]) *If G is a non-complete graph and z is the last vertex of a LexBFS ordering of G, then there exists a connected component C of $G - N[z]$ such that for every neighbor x of z, either $N[x] = N[z]$ or $N(x) \cap C \neq \emptyset$.*

Equivalently, if we put z together with its neighbors of the first type, the resultant set of vertices is a clique, a homogeneous set, and its neighborhood is a minimal separator. Such sets are called *moplexes* in [2] and Theorem 5 is stated in term of moplexes in [2]. Note that Theorem 5 can be proved from the following very convenient lemma.

Lemma 2 *Let $\prec$ be a LexBFS ordering of a graph $G = (V, E)$. Let z denote the last vertex in this ordering. Then for all vertices $a, b, c \in V$ such that $c \prec b \prec a$ and $ca \in E$, there exists a path from b to c whose internal vertices are disjoint from $N[z]$.*

Truemper Configurations

To state the next results, we need special types of graphs that are called Truemper configurations. They play an important role in structural graph theory; see [15]. Let us define them. A *3-path configuration* is a graph induced by three internally vertex disjoint paths of length at least 1, $P_1 = x_1 \dots y_1$, $P_2 = x_2 \dots y_2$ and $P_3 = x_3 \dots y_3$, such that either $x_1 = x_2 = x_3$ or x_1, x_2, x_3 are all distinct and pairwise adjacent and either $y_1 = y_2 = y_3$ or y_1, y_2, y_3 are all distinct and pairwise adjacent. Furthermore, the

vertices of $P_i \cup P_j$, $i \neq j$ induce a hole. Note that this last condition in the definition implies the following:

- If x_1, x_2, x_3 are distinct (and therefore pairwise adjacent) and y_1, y_2, y_3 are distinct, then the three paths have length at least 1. In this case, the configuration is called a *prism*.
- If $x_1 = x_2 = x_3$ and $y_1 = y_2 = y_3$, then the three paths have length at least 2 (since a path of length 1 would form a chord of the cycle formed by the two other paths). In this case, the configuration is called a *theta*.
- If $x_1 = x_2 = x_3$ and y_1, y_2, y_3 are distinct, or if x_1, x_2, x_3 are distinct and $y_1 = y_2 = y_3$, then at most one of the three paths has length 1, and the others have length at least 2. In this case, the configuration is called a *pyramid*.

A *wheel* (H, v) is a graph formed by a hole H, called the *rim*, and a vertex v, called the *center*, such that the center has at least three neighbors on the rim. A *Truemper configuration* is a graph that is either a prism, a theta, a pyramid, or a wheel.

A *hole* in a graph is a chordless cycle of length at least 4. It is *even* or *odd* according to the parity of the number of its edges. A graph is *universally signable* if it contains no Truemper configuration.

Speeding Up of Known Algorithms

We now state the previously known optimization algorithms for which we get better complexity by applying our method. In each case, we prove the existence of an elimination ordering, compute it with LexBFS, and take advantage of the ordering to solve the problem. Each time, we improve the previously known complexity by at least a factor of n:

- Maximum weighted clique in even-hole-free graphs in time $O(nm)$
- Maximum weighted clique in universally signable graphs in time $O(n + m)$
- Coloring in universally signable graphs in time $O(n + m)$

New Algorithms

We now apply systematically our method to all possible sets made of non-complete graphs of order 3. For each such set $\mathcal{F}$ (there are seven of them), we provide a class with a $\mathcal{F}$-elimination ordering.

To describe the classes of graphs that we obtain, we need to be more specific about wheels. A wheel is a *1-wheel* if for some consecutive vertices x, y, z of the rim, the center is adjacent to y and nonadjacent to x and z. A wheel is a *2-wheel* if for some consecutive vertices x, y, z of the rim, the center is adjacent to x and y and nonadjacent to z. A wheel is a *3-wheel* if for some consecutive vertices x, y, z of the rim, the center is adjacent to x, y and z. Observe that a wheel can be simultaneously a 1-wheel, a 2-wheel, and a 3-wheel. On the other hand, every wheel is a 1-wheel, a 2-wheel, or a 3-wheel. Also, any 3-wheel is either a 2-wheel or a *universal wheel* (i.e., a wheel whose center is adjacent to all vertices of the rim).

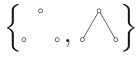
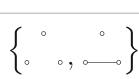
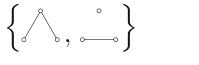
Up to isomorphism, there are four graphs on three vertices, and three of them are not complete. These three graphs (namely, the independent graph on three vertices denoted by S_3, the path of length 2 denoted by P_3, and its complement denoted by $\overline{P_3}$) are studied in the next lemma.

Lemma 3 *For a graph G, the following hold:*

(i) G is locally $\{S_3\}$-decomposable if and only if G is {1-wheel, theta, pyramid}-free.
(ii) G is locally $\{P_3\}$-decomposable if and only if G is 3-wheel-free.
(iii) G is locally $\{\overline{P_3}\}$-decomposable if and only if G is {2-wheel, prism, pyramid}-free.

Applying our method then leads to the next result, that is, a description of eight classes of graphs (one of them is the class of chordal graphs, and one of them is the class of universally signable graphs). They are described in Table 1: the second column describes the forbidden induced subgraphs that define the class and the last column describes the neighborhood of the last vertex of a LexBFS ordering.

LexBFS, Structure, and Algorithms, Table 1 Eight classes of graphs

i	Class $\mathcal{C}_i$	$\mathcal{F}_i$	Neighborhood
1	{1-wheel, theta, pyramid}-free		No stable set of size 3
2	3-wheel-free		Disjoint union of cliques
3	{2-wheel, prism, pyramid}-free		Complete multipartite
4	{1-wheel, 3-wheel, theta, pyramid}-free		Disjoint union of at most two cliques
5	{1-wheel, 2-wheel, prism, theta, pyramid}-free		Stable sets of size at most 2 with all possible edges between them
6	{2-wheel, 3-wheel, prism, pyramid}-free		Clique or stable set
7	{wheel, prism, theta, pyramid}-free		Clique or stable set of size 2
8	hole-free		Clique

LexBFS, Structure, and Algorithms, Table 2 Several properties of classes defined in Table 1

i	Max clique	Coloring
1	NP-hard [13]	NP-hard [9]
2	$O(nm)$ [14]	NP-hard [11]
3	$O(nm)$	NP-hard [11]
4	$O(n+m)$	?
5	$O(nm)$	?
6	$O(n+m)$	NP-hard [11]
7	$O(n+m)$	$O(n+m)$
8	$O(n+m)$ [14]	$O(n+m)$ [14]

Theorem 6 *For $i = 1,\ldots,8$, let $\mathcal{C}_i$ and $\mathcal{F}_i$ be the classes defined as in Table 1. For $i = 1,\ldots,8$, the class $\mathcal{C}_i$ is exactly the class of locally $\mathcal{F}_i$-decomposable graphs.*

For each class C_i, we survey in Table 2 the complexity of the maximum clique problem (for which our method provides sometimes a fast algorithm) and of the coloring problem.

Open Problems

Addario-Berry, Chudnovsky, Havet, Reed, and Seymour [1] proved that every even-hole-free graph admits a vertex whose neighborhood is the union of two cliques. We wonder whether this result can be proved by some search algorithm.

Our work suggests that a linear time algorithm for the maximum clique problem might exist in $\mathcal{C}_2$, but we could not find it.

We are not aware of a polynomial time coloring algorithm for graphs in $\mathcal{C}_4$ or $\mathcal{C}_5$.

Since class $\mathcal{C}_1$ generalizes claw-free graphs, it is natural to ask which of the properties of claw-free graphs it has, such as a structural description (see [4]), a polynomial time algorithm for the maximum stable set (see [7]), approximation algorithms for the chromatic number (see [10]), and a polynomial time algorithm for the induced linkage problem (see [8]).

In [5], an $O(nm)$ time algorithm is described for the maximum weighted stable set problem in $\mathcal{C}_7$. Since the class is a simple generalization of chordal graphs, we wonder whether a linear time algorithm exists.

Recommended Reading

1. Addario-Berry L, Chudnovsky M, Havet F, Reed B, Seymour P (2008) Bisimplicial vertices in even-hole-free graphs. J Comb Theory Ser B 98(6):1119–1164
2. Berry A, Bordat JP (1998) Separability generalizes dirac's theorem. Discret Appl Math 84(1–3):43–53
3. Brandstädt A, Dragan F, Nicolai F (1997) LexBFS-orderings and powers of chordal graphs. Discret Math 171(1–3):27–42
4. Chudnovsky M, Seymour P (2008) Clawfree graphs. IV. Decomposition theorem. J Comb Theory Ser B 98(5):839–938

5. Conforti M, Cornuéjols G, Kapoor A, Vušković K (1997) Universally signable graphs. Combinatorica 17(1):67–77
6. Dirac G (1961) On rigid circuit graphs. Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 25:71–76
7. Faenza Y, Oriolo G, Stauffer G (2011) An algorithmic decomposition of claw-free graphs leading to an $O(n^3)$-algorithm for the weighted stable set problem. In: SODA, San Francisco, pp 630–646
8. Fiala J, Kamiński M, Lidický B, Paulusma D (2012) The k-in-a-path problem for claw-free graphs. Algorithmica 62(1–2):499–519
9. Holyer I (1981) The NP-completeness of some edge-partition problems. SIAM J Comput 10(4): 713–717
10. King A (2009) Claw-free graphs and two conjectures on ω, δ, and χ. PhD thesis, McGill University
11. Maffray F, Preissmann M (1996) On the NP-completeness of the k-colorability problem for triangle-free graphs. Discret Math 162:313–317
12. Maffray F, Trotignon N, Vušković K (2008) Algorithms for square-$3PC(\cdot,\cdot)$-free Berge graphs. SIAM J Discret Math 22(1):51–71
13. Poljak S (1974) A note on the stable sets and coloring of graphs. Commentationes Mathematicae Universitatis Carolinae 15:307–309
14. Rose D, Tarjan R, Lueker G (1976) Algorithmic aspects of vertex elimination on graphs. SIAM J Comput 5:266–283
15. Vušković K (2013) The world of hereditary graph classes viewed through Truemper configurations. In: Blackburn SR, Gerke S, Wildon M (eds) Surveys in combinatorics. London mathematical society lecture note series, vol 409. Cambridge University Press, Cambridge, pp 265–325

Linearity Testing/Testing Hadamard Codes

Sofya Raskhodnikova[1] and Ronitt Rubinfeld[2,3]
[1]Computer Science and Engineering Department, Pennsylvania State University, University Park, State College, PA, USA
[2]Massachusetts Institute of Technology (MIT), Cambridge, MA, USA
[3]Tel Aviv University, Tel Aviv-Yafo, Israel

Keywords

Error-correcting codes; Group homomorphism; Linearity of functions; Property testing; Sublinear-time algorithms

Years and Authors of Summarized Original Work

1993; Blum, Luby, Rubinfeld

Problem Definition

In this article, we discuss the problem of testing linearity of functions and, more generally, testing whether a given function is a group homomorphism. An algorithm for this problem, given by [9], is one of the most celebrated property testing algorithms. It is part of or is a special case of many important property testers for algebraic properties. Originally designed for program checkers and self-correctors, it has found uses in probabilistically checkable proofs (PCPs), which are an essential tool in proving hardness of approximation.

We start by formulating an important special case of the problem, testing the linearity of Boolean functions. A function $f : \{0,1\}^n \to \{0,1\}$ is *linear* if for some $a_1, a_2, \ldots, a_n \in \{0,1\}$,

$$f(x_1, x_2, \ldots, x_n) = a_1 x_1 + a_2 x_2 + \cdots a_n x_n.$$

The operations in this definition are over $\mathbb{F}_2$. That is, given vectors $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$, where $x_1, \ldots, x_n, y_1, \ldots, y_n \in \{0,1\}$, the vector $\mathbf{x} + \mathbf{y} = (x_1 + y_1 \bmod 2, \ldots, x_n + y_n \bmod 2)$. There is another, equivalent definition of linearity of Boolean functions over $\{0,1\}^n$: a function f is *linear* if for all $x, y \in \{0,1\}^n$,

$$f(x) + f(y) = f(x + y).$$

A generalization of a linear function, defined above, is a group homomorphism. Given two finite groups, $(G, \circ)$ and $(H, \star)$, a *group homomorphism* from G to H is a function $f : G \to H$ such that for all elements $x, y \in G$,

$$f(x) \star f(y) = f(x \circ y).$$

We would like to test (approximately) whether a given function is linear or, more generally, is a group homomorphism. Next, we define the property testing framework [12, 23]. Linearity testing was the first problem studied in this framework. The linearity tester of [9] actually preceded the definition of this framework and served as an inspiration for it. Given a proximity parameter $\epsilon \in (0, 1)$, a function is ϵ-far from satisfying a specific property $\mathcal{P}$ (such as being linear or being a group homomorphism) if it has to be modified on at least an ϵ fraction of its domain in order to satisfy $\mathcal{P}$. A function is ϵ-close to $\mathcal{P}$ if it is not ϵ-far from it. A tester for property $\mathcal{P}$ gets a parameter $\epsilon \in (0, 1)$ and an oracle access to a function f. It must accept with probability (The choice of error probability in the definition of the tester is arbitrary. Using standard techniques, a tester with error probability 1/3 can be turned into a tester with error probability $\delta \in (0, 1/3)$ by repeating the original tester $O(\log \frac{1}{\delta})$ times and taking the majority answer.) at least 2/3 if the function f satisfies property $\mathcal{P}$ and reject with probability at least 2/3 if f is ϵ-far from satisfying $\mathcal{P}$. Our goal is to design an efficient tester for group homomorphism.

Alternative Formulation

Another way of viewing the same problem is in terms of error-correcting codes. Given a function $f : G \to H$, we can form a codeword corresponding to f by listing the values of f on all points in the domain. The *homomorphism* code is the set of all codewords that correspond to homomorphisms from G to H. This is an error-correcting code with large distance because, for two different homomorphisms $f, g : G \to H$, the fraction of points $x \in G$ on which $f(x) = g(x)$ is at most 1/2. In the special case when G is $\{0, 1\}^n$ and H is $\{0, 1\}$, we get the Hadamard code. Our goal can be formulated as follows: design an efficient algorithm that tests whether a given string is a codeword of a homomorphism code (or ϵ-far from it).

Key Results

The linearity (homomorphism) tester designed by [9] repeats the following test several times, until the desired success probability is reached, and accepts iff all iterations accept.

Algorithm 1: BLR Linearity (Homomorphism) Test

input : Oracle access to an unknown function $f : G \to H$.

1 Pick $x, y \in G$ uniformly and independently at random.
2 Query f on x, y, and $x + y$ to find out $f(x)$, $f(y)$, and $f(x + y)$.
3 **Accept** if $f(x) + f(y) = f(x + y)$; otherwise, **reject**.

Blum et al. [9] and Ben-Or et al. [7] showed that $O(1/\epsilon)$ iterations of the BLR test suffice to get a property tester for group homomorphism. (The analysis in [9] worked for a special case of the problem, and [7] extended it to all groups). It is not hard to prove that $\Omega(1/\epsilon)$ queries are required to test for linearity and, in fact, any non-trivial property, so the resulting tester is optimal in terms of the query complexity and the running time.

Lots of effort went into understanding the rejection probability of the BLR test for functions that are ϵ-far from homomorphisms over various groups and, especially, for the case $F = \{0, 1\}^n$ (see [17] and references therein). A nice exposition of the analysis for the latter special case, which follows the Fourier-analytic approach of [5], can be found in the book by [21].

Several works [8, 14, 24–26] showed how to reduce the number of random bits required by homomorphism tests. In the natural implementation of the BLR test, $2 \log |G|$ random bits per iteration are used to pick x and y. Shpilka and Wigderson [25] gave a homomorphism test for general groups that needs only $(1+o(1)) \log_2 |G|$ random bits.

The case when G is a subset of an infinite group, f is a real-valued function, and the oracle query to f returns a finite-precision approximation to $f(x)$ has been considered in [2, 10, 11, 19, 20]. These works gave testers with query complexity independent of the domain size (see [18] for a survey).

Applications

Self-Testing/Correcting Programs

The linearity testing problem was motivated in [9] by applications to self-testing and self-correcting of programs. Suppose you are given a program that is known to be correct on most inputs but has not been checked (or, perhaps, is even known to be incorrect) on remaining inputs. A *self-tester* for f is an algorithm that can quickly verify whether a given program that supposedly computes f is correct on most inputs, without the aid of another program for f that has already been verified. A *self-corrector* for f is an algorithm that takes a program that correctly computes f on most inputs and uses it to correctly compute f on all inputs.

Blum et al. [9] used their linearity test to construct self-testers for programs intended to compute various homomorphisms. Such functions include integer, polynomial, matrix, and modular multiplication and division. Once it is verified that a program agrees on most inputs with some homomorphism, the task of determining whether it agrees with the *correct* homomorphism on most inputs becomes much easier.

For programs intended to compute homomorphisms, it is easy to construct self-correctors: Suppose a program outputs $f(x)$ on input x, where f agrees on most inputs with a homomorphism g. Fix a constant c. Consider the algorithm that, on input x, picks $c \log 1/\delta$ values y from the domain G uniformly at random, computes $f(x+y) - f(y)$, and outputs the value that is seen most often, breaking ties arbitrarily. If f is $\frac{1}{8}$-close to g, then, since both y and $x+y$ are uniformly distributed in G, it is the case that for at least $3/4$ of the choices of y, both $g(x+y) = f(x+y)$ and $g(y) = f(y)$, in which case $f(x+y) - f(y) = g(x)$. Thus, it is easy to show that there is a constant c such that if f is $\frac{1}{8}$-close to a homomorphism g, then for all x, the above algorithm outputs $g(x)$ with probability at least $1-\delta$.

Probabilistically Checkable Proofs

We discussed an equivalent formulation of the linearity testing problem in terms of testing whether a given string is a codeword of a Hadamard code. This formulation has been used in proofs of hardness of approximation of some NP-hard problems and to construct PCP systems that can be verified with a few queries (see, e.g., [3, 13]).

The BLR Test as a Building Block

The BLR test has been generalized and extended in many ways, as well as used as a building block in other testers. One generalization, particularly useful in PCP constructions, is to testing if a given function is a polynomial of low degree (see, e.g., [1, 15, 16]). Other generalizations include tests for long codes [6, 13] and tests of linear consistency among multiple functions [4]. An example of an algorithm that uses the BLR test as a building block is a tester by [22] for the *singleton* property of functions $f : \{0,1\}^n \to \{0,1\}$, namely, the property that the function $f(x) = x_i$ for some $i \in [1,n]$.

Open Problems

We discussed that the BLR test can be used to check whether a given string is a Hadamard codeword or far from it. For which other codes can such a check be performed efficiently? In other words, which codes are locally testable? We refer the reader to the entry ▸ Locally Testable Codes.

Which other properties of functions can be efficiently tested in the property testing model? Some examples are given in the entries ▸ Testing Juntas and Related Properties of Boolean Functions and ▸ Monotonicity Testing. Testing properties of graphs is discussed in the entries ▸ Testing Bipartiteness in the Dense-Graph Model and ▸ Testing Bipartiteness of Graphs in Sublinear Time.

Cross-References

- ▸ Learning Heavy Fourier Coefficients of Boolean Functions
- ▸ Locally Testable Codes
- ▸ Monotonicity Testing

- ▸ Quantum Error Correction
- ▸ Testing Bipartiteness in the Dense-Graph Model
- ▸ Testing Bipartiteness of Graphs in Sublinear Time
- ▸ Testing if an Array Is Sorted
- ▸ Testing Juntas and Related Properties of Boolean Functions

Acknowledgments The first author was supported in part by NSF award CCF-1422975 and by NSF CAREER award CCF-0845701.

Recommended Reading

1. Alon N, Kaufman T, Krivilevich M, Litsyn S, Ron D (2003) Testing low-degree polynomials over GF(2). In: Proceedings of RANDOM'03, Princeton, pp 188–199
2. Ar S, Blum M, Codenotti B, Gemmell P (1993) Checking approximate computations over the reals. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing, San Diego, pp 786–795
3. Arora S, Lund C, Motwani R, Sudan M, Szegedy M (1998) Proof verification and the hardness of approximation problems. J ACM 45(3): 501–555
4. Aumann Y, Håstad J, Rabin MO, Sudan M (2001) Linear-consistency testing. J Comput Syst Sci 62(4):589–607
5. Bellare M, Coppersmith D, Håstad J, Kiwi M, Sudan M (1996) Linearity testing over characteristic two. IEEE Trans Inf Theory 42(6):1781–1795
6. Bellare M, Goldreich O, Sudan M (1998) Free bits, PCPs, and nonapproximability—towards tight results. SIAM J Comput 27(3):804–915
7. Ben-Or M, Coppersmith D, Luby M, Rubinfeld R (2008) Non-Abelian homomorphism testing, and distributions close to their self-convolutions. Random Struct Algorithms 32(1):49–70
8. Ben-Sasson E, Sudan M, Vadhan S, Wigderson A (2003) Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on the Theory of Computing, San Diego, pp 612–621
9. Blum M, Luby M, Rubinfeld R (1993) Self-testing/correcting with applications to numerical problems. JCSS 47:549–595
10. Ergun F, Kumar R, Rubinfeld R (2001) Checking approximate computations of polynomials and functional equations. SIAM J Comput 31(2):s 550–576
11. Gemmell P, Lipton R, Rubinfeld R, Sudan M, Wigderson A (1991) Self-testing/correcting for polynomials and for approximate functions. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, New Orleans, pp 32–42
12. Goldreich O, Goldwasser S, Ron D (1998) Property testing and its connection to learning and approximation. J ACM 45(4):653–750
13. Håstad J (2001) Some optimal in approximability results. J ACM 48(4):798–859
14. Hastad J, Wigderson A (2003) Simple analysis of graph tests for linearity and PCP. Random Struct Algorithms 22(2):139–160
15. Jutla CS, Patthak AC, Rudra A, Zuckerman D (2009) Testing low-degree polynomials over prime fields. Random Struct Algorithms 35(2): 163–193
16. Kaufman T, Ron D (2006) Testing polynomials over general fields. SIAM J Comput 36(3):779–802
17. Kaufman T, Litsyn S, Xie N (2010) Breaking the epsilon-soundness bound of the linearity test over GF(2). SIAM J Comput 39(5):1988–2003
18. Kiwi M, Magniez F, Santha M (2001) Exact and approximate testing/correcting of algebraic functions: a survey. Electron. Colloq. Comput. Complex. 8(14). http://dblp.uni-trier.de/db/journals/eccc/eccc8.html#ECCC-TR01-014
19. Kiwi M, Magniez F, Santha M (2003) Approximate testing with error relative to input size. JCSS 66(2):371–392
20. Magniez F (2005) Multi-linearity self-testing with relative error. Theory Comput Syst 38(5):573–591
21. O'Donnell R (2014) Analysis of Boolean Functions. Cambridge University Press, New York
22. Parnas M, Ron D, Samorodnitsky A (2002) Testing basic Boolean formulae. SIAM J Discret Math 16(1):20–46
23. Rubinfeld R, Sudan M (1996) Robust characterizations of polynomials with applications to program testing. SIAM J Comput 25(2):252–271
24. Samorodnitsky A, Trevisan L (2000) A PCP characterization of NP with optimal amortized query complexity. In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, Portland, pp 191–199
25. Shpilka A, Wigderson A (2006) Derandomizing homomorphism testing in general groups. SIAM J Comput 36(4):1215–1230
26. Trevisan L (1998) Recycling queries in PCPs and in linearity tests. In: Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, pp 299–308

Linearizability

Maurice Herlihy
Department of Computer Science, Brown University, Providence, RI, USA

Keywords

Atomicity

Years and Authors of Summarized Original Work

1990; Herlihy, Wing

Problem Definition

An *object* in languages such as Java and C++ is a container for data. Each object provides a set of *methods* that are the only way to to manipulate that object's internal state. Each object has a *class* which defines the methods it provides and what they do.

In the absence of concurrency, methods can be described by a pair consisting of a *precondition* (describing the object's state before invoking the method) and a *postcondition*, describing, once the method returns, the object's state and the method's return value. If, however, an object is shared by concurrent threads in a multiprocessor system, then method calls may overlap in time, and it no longer makes sense to characterize methods in terms of pre- and post-conditions.

Linearizability is a correctness condition for concurrent objects that characterizes an object's concurrent behavior in terms of an "equivalent" sequential behavior. Informally, the object behaves "as if" each method call takes effect instantaneously at some point between its invocation and its response. This notion of correctness has some useful formal properties. First, it is *nonblocking*, which means that linearizability as such never requires one thread to wait for another to complete an ongoing method call. Second, it is *local*, which means that an object composed of linearizable objects is itself linearizable. Other proposed correctness conditions in the literature lack at least one of these properties.

Notation

An execution of a concurrent system is modeled by a *history*, a finite sequence of method *invocation* and *response events*. A *subhistory* of a history H is a subsequence of the events of H. A method invocation is written as $\langle x.m(a^*)A\rangle$, where x is an object, m a method name, a^* a sequence of arguments, and A a thread. A method response is written as $\langle x{:}\,t(r^*)A\rangle$ where t is a termination condition and r^* is a sequence of result values.

A response *matches* an invocation if their objects and thread names agree. A *method call* is a pair consisting of an invocation and the next matching response. An invocation is *pending* in a history if no matching response follows the invocation. If H is a history, *complete*(H) is the subsequence of H consisting of all matching invocations and responses. A history H is *sequential* if the first event of H is an invocation, and each invocation, except possibly the last, is immediately followed by a matching response.

Let H be a a history. The *thread subhistory* $H|P$ is the subsequence of events in H with thread name P. The *object subhistory* $H|x$ is similarly defined for an object x. Two histories H and H' are *equivalent* if for every thread $A, H|A = H'|A$. A history H is *well-formed* if each thread subhistory $H|A$ of H is sequential. Notice that thread subhistories of a well-formed history are always sequential, but object subhistories need not be.

A *sequential specification* for an object is a prefix-closed set of sequential object histories that defines that object's *legal* histories. A sequential history H is *legal* if each object subhistory is legal. A method is *total* if it is defined for every object state, otherwise it is *partial*. (For example, a *deq()* method that blocks on an empty queue is partial, while one that throws an exception is total.)

A history H defines an (irreflexive) partial order $\rightarrow_H$ on its method calls: $m_0 \rightarrow_H m_1$ if the result event of m_0 occurs before the invocation event of m_1. If H is a sequential history, then $\rightarrow_H$ is a total order.

Let H be a history and x an object such that $H|x$ contains method calls m_0 and m_1. A call $m_0 \rightarrow_x m_1$ if m_0 precedes m_1 in $H|x$. Note that $\rightarrow_x$ is a total order.

Informally, linearizability requires that each method call appear to "take effect" instantaneously at some moment between its invocation and response. An important implication of this definition is that method calls that do not overlap cannot be reordered:

linearizability preserves the "real-time" order of method calls. Formally,

Definition 1 A history H is *linearizable* if it can be extended (by appending zero or more response events) to a history H' such that:

- L1 $complete(H')$ is equivalent to a legal sequential history S, and
- L2 If method call m_0 precedes method call m_1 in H, then the same is true in S.

S is called a *linearization* of H. (H may have multiple linearizations.) Informally, extending H to H' captures the idea that some pending invocations may have taken effect even though their responses have not yet been returned to the caller.

Key Results

The Locality Property

A property is *local* if all objects collectively satisfy that property provided that each individual object satisfies it.

Linearizability is local:

Theorem 1 *H is linearizable if and only if $H|x$ is linearizable for ever object x.*

Proof The "only if" part is obvious.

For each object x, pick a linearization of $H|x$. Let R_x be the set of responses appended to $H|x$ to construct that linearization, and let $\rightarrow_x$ be the corresponding linearization order. Let H' be the history constructed by appending to H each response in R_x.

The $\rightarrow_H$ and $\rightarrow_x$ orders can be "rolled up" into a single partial order. Define the relation $\rightarrow$ on method calls of $complete(H')$: For method calls m and $\bar{m}$, $m \rightarrow \bar{m}$ if there exist method calls $m_0, \ldots, m_n$, such that $m = m_0, \bar{m} = m_n$, and for each i between 0 and $n-1$, either $m_i \rightarrow_x m_{i+1}$ for some object x, or $m_i \rightarrow_H m_{i+1}$.

It turns out that $\rightarrow$ is a partial order. Clearly, $\rightarrow$ is transitive. It remains to be shown that $\rightarrow$ is anti-reflexive: for all x, it is false that $x \rightarrow x$.

The proof proceeds by contradiction. If not, then there exist method calls $m_0, \ldots, m_n$, such that $m_0 \rightarrow m_1 \rightarrow \cdots \rightarrow m_n, m_n \rightarrow m_0$, and each pair is directly related by some $\rightarrow_x$ or by $\rightarrow_H$.

Choose a cycle whose length is minimal. Suppose all method calls are associated with the same object x. Since $\rightarrow_x$ is a total order, there must exist two method calls m_{i-1} and m_i such that $m_{i-1} \rightarrow_H m_i$ and $m_i \rightarrow_x m_{i-1}$, contradicting the linearizability of x.

The cycle must therefore include method calls of at least two objects. By reindexing if necessary, let m_1 and m_2 be method calls of distinct objects. Let x be the object associated with m_1. None of $m_2, \ldots, m_n$ can be a method call of x. The claim holds for m_2 by construction. Let m_i be the first method call in $m_3, \ldots, m_n$ associated with x. Since m_{i-1} and m_i are unrelated by $\rightarrow_x$, they must be related by $\rightarrow_H$, so the response of m_{i-1} precedes the invocation of m_i. The invocation of m_2 precedes the response of m_{i-1}, since otherwise $m_{i-1} \rightarrow_H m_2$, yielding the shorter cycle $m_2, \ldots, m_{i-1}$. Finally, the response of m_1 precedes the invocation of m_2, since $m_1 \rightarrow_H m_2$ by construction. It follows that the response to m_1 precedes the invocation of m_i, hence $m_1 \rightarrow_H m_i$, yielding the shorter cycle $m_1, m_i, \ldots, m_n$.

Since m_n is not a method call of x, but $m_n \rightarrow m_1$, it follows that $m_n \rightarrow_H m_1$. But $m_1 \rightarrow_H m_2$ by construction, and because $\rightarrow_H$ is transitive, $m_n \rightarrow_H m_2$, yielding the shorter cycle $m_2, \ldots, m_n$, the final contradiction. □

Locality is important because it allows concurrent systems to be designed and constructed in a modular fashion; linearizable objects can be implemented, verified, and executed independently. A concurrent system based on a non-local correctness property must either rely on a centralized scheduler for all objects, or else satisfy additional constraints placed on objects to ensure that they follow compatible scheduling protocols. Locality should not be taken for granted; as discussed below, the literature includes proposals for alternative correctness properties that are not local.

The Non-blocking Property

Linearizability is a *non-blocking* property: a pending invocation of a total method is never required to wait for another pending invocation to complete.

Theorem 2 *Let inv(m) be an invocation of a total method. If* $\langle x\ invP\rangle$ *is a pending invocation in a linearizable history H, then there exists a response* $\langle x\ resP\rangle$ *such that* $H \cdot \langle x\ resP\rangle$ *is linearizable.*

Proof Let S be any linearization of H. If S includes a response $\langle x\ resP\rangle$ to $\langle x\ invP\rangle$, the proof is complete, since S is also a linearization of $H \cdot \langle x\ resP\rangle$. Otherwise, $\langle x\ invP\rangle$ does not appear in S either, since linearizations, by definition, include no pending invocations. Because the method is total, there exists a response $\langle x\ resP\rangle$ such that

$$S' = S \cdot \langle x\ invP\rangle \cdot \langle x\ res\ P\rangle$$

is legal. S', however, is a linearization of $H \cdot \langle x\ resP\rangle$, and hence is also a linearization of H. □

This theorem implies that linearizability by itself never forces a thread with a pending invocation of a total method to block. Of course, blocking (or even deadlock) may occur as artifacts of particular implementations of linearizability, but it is not inherent to the correctness property itself. This theorem suggests that linearizability is an appropriate correctness condition for systems where concurrency and real-time response are important. Alternative correctness conditions, such as serializability [1] do not share this non-blocking property.

The non-blocking property does not rule out blocking in situations where it is explicitly intended. For example, it may be sensible for a thread attempting to dequeue from an empty queue to block, waiting until another thread enqueues an item. The queue specification captures this intention by making the `deq()` method's specification partial, leaving it's effect undefined when applied to an empty queue. The most natural concurrent interpretation of a partial sequential specification is simply to wait until the object reaches a state in which the method is defined.

Other Correctness Properties

Sequential Consistency [4] is a weaker correctness condition that requires Property *L1* but not *L2*: method calls must appear to happen in some one-at-a-time, sequential order, but calls that do not overlap can be reordered. Every linearizable history is sequentially consistent, but not vice versa. Sequential consistency permits more concurrency, but it is not a local property: a system composed of multiple sequentially-consistent objects is not itself necessarily sequentially consistent.

Much work on databases and distributed systems uses *serializability* as the basic correctness condition for concurrent computations. In this model, a *transaction* is a "thread of control" that applies a finite sequence of methods to a set of objects shared with other transactions. A history is *serializable* if it is equivalent to one in which transactions appear to execute sequentially, that is, without interleaving. A history is *strictly serializable* if the transactions' order in the sequential history is compatible with their precedence order: if every method call of one transaction precedes every method call of another, the former is serialized first. (Linearizability can be viewed as a special case of strict serializability where transactions are restricted to consist of a single method applied to a single object.)

Neither serializability nor strict serializability is a local property. If different objects serialize transactions in different orders, then there may be no serialization order common to all objects. Serializability and strict serializability are *blocking* properties: Under certain circumstances, a transaction may be unable to complete a pending method without violating serializability. A *deadlock* results if multiple transactions block one another. Such transactions must be rolled back and restarted, implying that additional mechanisms must be provided for that purpose.

Applications

Linearizability is widely used as the basic correctness condition for many concurrent data structure algorithms [5], particularly for lock-free and wait-free data structures [2]. Sequential consistency is widely used for describing low-level systems such as hardware memory interfaces. Serializability and strict serializability are widely used for database systems in which it must be easy for application programmers to preserve complex application-specific invariants spanning multiple objects.

Open Problems

Modern multiprocessors often support very weak models of memory consistency. There are many open problems concerning how to model such behavior, and how to ensure linearizable object implementations on top of such architectures.

Cross-References

- ▸ Concurrent Programming, Mutual Exclusion
- ▸ Registers

Recommended Reading

The notion of Linearizability is due to Herlihy and Wing [3], while Sequential Consistency is due to Lamport [4], and serializability to Eswaran et al. [1].

1. Eswaran KP, Gray JN, Lorie RA, Traiger IL (1976) The notions of consistency and predicate locks in a database system. Commun ACM 19(11):624–633. doi:10.1145/360363.360369
2. Herlihy M (1991) Wait-free synchronization. ACM Trans Program Lang Syst (TOPLAS) 13(1): 124–149
3. Herlihy MP, Wing JM (1990) Linearizability: a correctness condition for concurrent objects. ACM Trans Program Lang Syst (TOPLAS) 12(3): 463–492
4. Lamport L (1979) How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans Comput C-28(9):690
5. Vafeiadis V, Herlihy M, Hoare T, Shapiro M (2006) Proving correctness of highly-concurrent linearisable objects. In: PPoPP'06: proceedings of the eleventh ACM SIGPLAN symposium on principles and practice of parallel programming, pp 129–136. doi:10.1145/1122971.1122992

List Decoding near Capacity: Folded RS Codes

Atri Rudra
Department of Computer Science and Engineering, State University of New York, Buffalo, NY, USA

Keywords

Decoding; Error correction

Years and Authors of Summarized Original Work

2006; Guruswami, Rudra

Problem Definition

One of the central trade-offs in the theory of error-correcting codes is the one between the amount of redundancy needed and the fraction of errors that can be corrected. (This entry deals with the adversarial or worst-case model of errors—no assumption is made on how the errors and error locations are distributed beyond an upper bound on the total number of errors that may be caused.) The redundancy is measured by the *rate* of the code, which is the ratio of the the number of information symbols in the message to that in the codeword – thus, for a code with encoding function $E : \Sigma^k \rightarrow \Sigma^n$, the rate equals k / n. The *block length* of the code equals n, and Σ is its *alphabet*.

The goal in *decoding* is to find, given a noisy received word, the actual codeword that it could

have possibly resulted from. If the target is to correct a fraction ρ of errors (ρ will be called the error-correction radius), then this amounts to finding codewords within (normalized Hamming) distance ρ from the received word. We are guaranteed that there will be a unique such codeword provided the distance between *every* two distinct codewords is at least 2ρ, or in other words the relative distance of the code is at least 2ρ. However, since the relative distance δ of a code must satisfy $\delta \leq 1 - R$ where R is the rate of the code (by the Singleton bound), if one insists on an unique answer, the best trade-off between ρ and R is $\rho = \rho_U(R) = (1 - R)/2$. But this is an overly pessimistic estimate of the error-correction radius, since the way Hamming spheres pack in space, for *most* choices of the received word there will be at most one codeword within distance ρ from it even for ρ much greater than $\delta/2$. Therefore, *always* insisting on a unique answer will preclude decoding most such received words owing to a few pathological received words that have more than one codeword within distance roughly $\delta/2$ from them.

A notion called list decoding, that dates back to the late 1950s [1, 9], provides a clean way to get around this predicament, and yet deal with worst-case error patterns. Under list decoding, the decoder is required to output a list of all codewords within distance ρ from the received word. Let us call a code C (ρ, L) *-list decodable* if the number of codewords within distance ρ of any received word is at most L. To obtain better trade-offs via list decoding, (ρ, L)-list decodable codes are needed where L is bounded by a polynomial function of the block length, since this an *a priori* requirement for polynomial time list decoding. How large can ρ be as a function of R for which such (ρ, L)-list decodable codes exist? A standard random coding argument shows that $\rho \geq 1 - R - o(1)$ can be achieved over large enough alphabets, cf. [2, 10], and a simple counting argument shows that ρ must be at most $1 - R$. Therefore the *list decoding capacity*, i.e., the information-theoretic limit of list decodability, is given by the trade-off $\rho_{\text{cap}}(R) = 1 - R = 2\rho_U(R)$. Thus list decoding holds the promise of correcting *twice* as many errors as unique decoding, for *every* rate. The above-mentioned list decodable codes are non-constructive. In order to realize the potential of list decoding, one needs explicit constructions of such codes, and on top of that, polynomial time algorithms to perform list decoding.

Building on works of Sudan [8], Guruswami and Sudan [6] and Parvaresh and Vardy [7], Guruswami and Rudra [5] present codes that get arbitrarily close to the list decoding capacity $\rho_{\text{cap}}(R)$ for every rate. In particular, for every $1 > R > 0$ and every $\epsilon > 0$, they give *explicit* codes of rate R together with polynomial time list decoding algorithm that can correct up to a fraction $1 - R - \epsilon$ of errors. These are the first explicit codes (with efficient list decoding algorithms) that get arbitrarily close to the list decoding capacity for *any* rate.

Description of the Code

Consider a Reed–Solomon (RS) code $C = \mathsf{RS}_{\mathbb{F},\mathbb{F}^*}[n, k]$ consisting of evaluations of degree k polynomials over some finite field $\mathbb{F}$ at the set $\mathbb{F}^*$ of nonzero elements of $\mathbb{F}$. Let $q = |\mathbb{F}| = n + 1$. Let γ be a generator of the multiplicative group $\mathbb{F}^*$, and let the evaluation points be ordered as $1, \gamma, \gamma^2, \ldots, \gamma^{n-1}$. Using all nonzero field elements as evaluation points is one of the most commonly used instantiations of Reed–Solomon codes.

Let $m \geq 1$ be an integer parameter called the *folding parameter*. For ease of presentation, it will assumed that m divides $n = q - 1$.

Definition 1 (Folded Reed–Solomon Code) The m-folded version of the RS code C, denoted $\mathsf{FRS}_{\mathbb{F},\gamma,m,k}$, is a code of block length $N = n/m$ over $\mathbb{F}^m$. The encoding of a message $f(X)$, a polynomial over $\mathbb{F}$ of degree at most k, has as its j'th symbol, for $0 \leq j < n/m$, the m-tuple $(f(\gamma^{jm}), f(\gamma^{jm+1}), \cdots, f(\gamma^{jm+m-1}))$. In other words, the codewords of $C' = \mathsf{FRS}_{\mathbb{F},\gamma,m,k}$ are in one-one correspondence with those of the RS code C and are obtained by bundling together consecutive m-tuple of symbols in codewords of C.

Key Results

The following is the main result of Guruswami and Rudra.

Theorem 1 ([5]) *For every $\epsilon > 0$ and $0 < R < 1$, there is a family of folded Reed–Solomon codes that have rate at least R and which can be list decoded up to a fraction $1 - R - \epsilon$ of errors in time (and outputs a list of size at most) $(N/\epsilon^2)^{O(\epsilon^{-1}\log(1/R))}$ where N is the block length of the code. The alphabet size of the code as a function of the block length N is $(N/\epsilon^2)^{O(1/\epsilon^2)}$.*

The result of Guruswami and Rudra also works in a more general setting called *list recovering*, which is defined next.

Definition 2 (List Recovering) A code $C \subseteq \Sigma^n$ is said to be (ζ, l, L)-list recoverable if for every sequence of sets $S_1, \cdots, S_n$ where each $S_i \subseteq \Sigma$ has at most l elements, the number of codewords $c \in C$ for which $c_i \in S_i$ for at least ζn positions $i \in \{1, 2, \ldots, n\}$ is at most L.

A code $C \subseteq \Sigma^n$ is said to (ζ, l)-list recoverable in polynomial time if it is $(\zeta, l, L(n))$-list recoverable for some polynomially bounded function $L(\cdot)$, and moreover there is a polynomial time algorithm to find the at most $L(n)$ codewords that are solutions to any $(\zeta, l, L(n))$-list recovering instance.

Note that when $l = 1$, $(\zeta, 1, \cdot)$-list recovering is the same as list decoding up to a $(1 - \zeta)$ fraction of errors. Guruswami and Rudra have the following result for list recovering.

Theorem 2 ([5]) *For every integer $l \geq 1$, for all R, $0 < R < 1$ and $\epsilon > 0$, and for every prime p, there is an explicit family of folded Reed–Solomon codes over fields of characteristic p that have rate at least R and which can be $(R + \epsilon, l)$-list recovered in polynomial time. The alphabet size of a code of block length N in the family is $(N/\epsilon^2)^{O(\epsilon^{-2}\log l/(1-R))}$.*

Applications

To get within ϵ of capacity, the codes in Theorem 1 have alphabet size $N^{\Omega(1/\epsilon^2)}$ where N is the block length. By concatenating folded RS codes of rate close to 1 (that are list recoverable) with suitable inner codes followed by redistribution of symbols using an expander graph (similar to a construction for linear-time unique decodable codes in [3]), one can get within ϵ of capacity with codes over an alphabet of size $2^{O(\epsilon^{-4}\log(1/\epsilon))}$. A counting argument shows that codes that can be list decoded efficiently to within ϵ of the capacity need to have an alphabet size of $2^{\Omega(1/\epsilon)}$.

For binary codes, the list decoding capacity is known to be $\rho_{\text{bin}}(R) = H^{-1}(1 - R)$ where $H(\cdot)$ denotes the binary entropy function. No explicit constructions of binary codes that approach this capacity are known. However, using the Folded RS codes of Guruswami Rudra in a natural concatenation scheme, one can obtain polynomial time constructable binary codes of rate R that can be list decoded up to a fraction $\rho_{\text{Zyab}}(R)$ of errors, where $\rho_{\text{Zyab}}(R)$ is the "Zyablov bound".

See [5] for more details.

Open Problems

The work of Guruswami and Rudra could be improved with respect to some parameters. The size of the list needed to perform list decoding to a radius that is within ϵ of capacity grows as $N^{O(\epsilon^{-1}\log(1/R))}$ where N and R are the block length and the rate of the code respectively. It remains an open question to bring this list size down to a constant independent of n (the existential random coding arguments work with a list size of $O(1/\epsilon)$). The alphabet size needed to approach capacity was shown to be a constant independent of N. However, this involved a brute-force search for a rather large (inner) code, which translates to a construction time of about $N^{O(\epsilon^{-2}\log(1/\epsilon))}$ (instead of the ideal construction time where the exponent of N does not depend on ϵ). Obtaining a "direct" algebraic construction

over a constant-sized alphabet, such as the generalization of the Parvaresh-Vardy framework to algebraic-geometric codes in [4], might help in addressing these two issues.

Finally, constructing binary codes that approach list decoding capacity remains open.

Cross-References

- ▸ Decoding Reed–Solomon Codes
- ▸ Learning Heavy Fourier Coefficients of Boolean Functions
- ▸ LP Decoding

Recommended Reading

1. Elias P (1957) List decoding for noisy channels. Technical report 335, Research Laboratory of Electronics MIT
2. Elias P (1991) Error-correcting codes for list decoding. IEEE Trans Inf Theory 37:5–12
3. Guruswami V, Indyk P (2005) Linear-time encodable/decodable codes with near-optimal rate. IEEE Trans Inf Theory 51(10):3393–3400
4. Guruswami V, Patthak A (2006) Correlated Algebraic-Geometric codes: improved list decoding over bounded alphabets. In: Proceedings of the 47th annual IEEE symposium on foundations of computer science (FOCS), Oct 2006, Berkley, pp 227–236
5. Guruswami V, Rudra A (2006) Explicit capacity-achieving list-decodable codes. In: Proceedings of the 38th annual ACM symposium on theory of computing, May 2006, Seattle, pp 1–10
6. Guruswami V, Sudan M (1999) Improved decoding of Reed–Solomon and algebraic-geometric codes. IEEE Trans Inf Theory 45:1757–1767
7. Parvaresh F, Vardy A (2005) Correcting errors beyond the Guruswami–Sudan radius in polynomial time. In: Proceedings of the 46th annual IEEE symposium on foundations of computer science, Pittsburgh, pp 285–294
8. Sudan M (1997) Decoding of Reed-Solomon codes beyond the error-correction bound. J Complex 13(1):180–193
9. Wozencraft JM (1958) List decoding. Quarterly progress report, research laboratory of electronics. MIT 48:90–95
10. Zyablov VV, Pinsker MS (1981) List cascade decoding. Probl Inf Transm 17(4):29–34 (in Russian); pp 236–240 (in English) (1982)

List Ranking

Riko Jacob[1,2], Ulrich Meyer[3], and Laura Toma[4]
[1]Institute of Computer Science, Technical University of Munich, Munich, Germany
[2]IT University of Copenhagen, Copenhagen, Denmark
[3]Department of Computer Science, Goethe University Fankfurt am Main, Frankfurt, Germany
[4]Department of Computer Science, Bowdoin College, Brunswick, ME, USA

Keywords

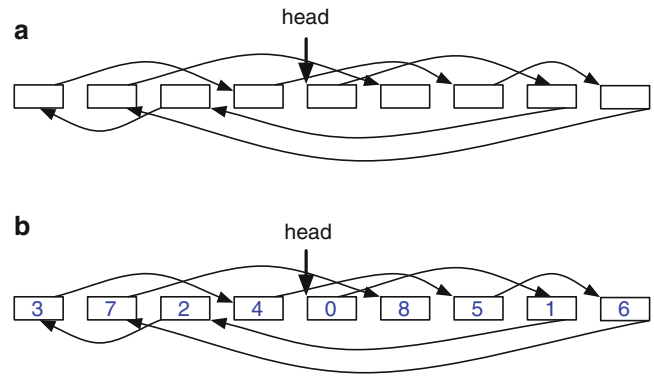
3-coloring; Euler Tour; External memory algorithms; Graph algorithms; Independent set; PRAM algorithms; Time-forward processing

Years and Authors of Summarized Original Work

1995; Chiang, Goodrich, Grove, Tamassia, Vengroff, Vitter

Problem Definition

Let L be a linked list of n vertices $x_1, x_2, \ldots, x_n$ such that every vertex x_i stores a pointer $succ(x_i)$ to its successor in L. As with any linked list, we assume that no two vertices have the same successor, any vertex can reach the tail of the list by following successor pointers, and we denote the *head* of the list the vertex that no other vertex in L points to and the *tail* the vertex whose successor is $null$. Given the head x_h of L, the *list-ranking* problem is to find the *rank*, or distance, of each vertex x_i in L from the head of L: that is, $rank(x_h) = 0$ and $rank(succ(x_i)) = rank(x_i) + 1$; refer to Fig. 1. A generalization of this problem is to consider that each vertex x_i stores, in addition to $succ(x_i)$, a weight $w(x_i)$; in this case the list is given as a set of tuples $\{(x_i, w(x_i), succ(x_i))\}$ and we want to compute



List Ranking, Fig. 1 (**a**) An instance of the LR problem, with the ranks shown in (**b**)

$rank(x_h) = w(x_h)$, and $rank(succ(x_i)) = rank(x_i) + w(succ(x_i))$.

Key Results

List ranking is one of the fundamental problems in the external memory model (EM or I/O-model) which requires nontrivial techniques and illustrates the differences (and connection) between the models of computation, namely, the random access machine (RAM), its parallel version PRAM, and the parallel external memory model (PEM). It also illustrates how ideas from parallel algorithms are used in serial external memory algorithms and the idea of using geometrically decreasing sizes to get an algorithm that in total is as fast as sorting. Furthermore list ranking is the main ingredient in the Euler Tour technique, which is one of the main techniques for obtaining I/O-efficient solutions for fundamental problems on graphs and trees.

In internal memory, list ranking can be solved in $O(n)$ time with a straightforward algorithm that starts from the head of the list and follows successor pointers. In external memory, the same algorithm may use $\Omega(n)$ I/Os – the intuition is that in the worst case, the vertices are arranged in such an order that following a successor pointer will always require loading a new block from disk (one I/O). If the vertices were arranged in order of their ranks, then traversing the list would be trivial, but arranging them in this order would require knowing their ranks, which is exactly the problem we are trying to solve.

The external memory model (see also chapter ▸ I/O-Model) has been extended to a parallel version, the PEM model, which models a private cache shared memory architecture, where the shared memory (external memory) is organized in blocks of size B, each cache (internal memory) has size M, and there are P processors. The cost of executing an algorithm is the number of parallel I/Os. When we use asymptotic notation, we think of M, B, and P as arbitrary nondecreasing functions depending on n, the number of elements constituting the input. Similar to the PRAM, there are different versions of the model, depending on the possibility of concurrent read and write. In the following we assume a Concurrent Read Exclusive Write (CREW) policy [2]. For $M = 2, B = 1$ the PEM model is a PRAM model, and for $P = 1$ the EM model.

Similar to the EM model (see chapter ▸ External Sorting and Permuting), sorting is an important building block for the PEM having complexity $\text{sort}(n) = O(\frac{n}{PB} \log_{\text{d}} \frac{n}{B})$ for $\text{d} = \max\{2, \min\{\frac{n}{PB}, \frac{M}{B}\}\}$ if $P \leq \frac{n}{B}$ [7].

The complexity of permuting in the PEM model is that of sorting unless the direct algorithm with $O(\frac{n}{P})$ I/Os is faster, i.e., for B smaller than the logarithmic term.

In the RAM model, permuting and list ranking have scanning complexity, while (comparison-based) sorting is more expensive. In the PRAM model, this changes slightly: permuting still has scanning complexity, while list ranking has sorting complexity (like any function where a single output depends on all inputs it needs $n/P + \log n$ time). In the external memory model, list ranking

has permuting complexity which is higher than scanning unless $B = O(1)$, and usually it is sorting complexity. In contrast, for many processors and large B (more precisely $P = M = 2B = \sqrt{n}$) and a restricted model where the input is only revealed to the algorithm if certain progress has been made, list ranking has complexity $\Omega(\log^2 n)$ [8].

List-Ranking Algorithms in Parallel External Memory

A solution for list ranking that runs in $O(\text{sort}(n))$ I/Os was described by Chiang et al. [4]. The general idea is based on a PRAM algorithm by Cole and Vishkin and consists of the following steps:

1. Find an independent set I of L (a set of vertices such that no two vertices in I are adjacent in L) consisting of $\Theta(n)$ vertices.
2. Compute a new list $L - I$ by removing the vertices in I from L; that is, all vertices x in I are bridged out: let y be the vertex with $succ(y) = x$, and then we set $succ(y) := succ(x)$; additionally the weight of x is added to the weight of $succ(x)$. This ensures that the rank of any vertex in $L - I$ is the same as its rank in L.
3. Compute the ranks recursively on $L - I$.
4. Compute the ranks of the vertices in I from the ranks of the neighbors in $L - I$.

The key idea of the algorithm is finding an independent set of size $\Omega(c \cdot n)$ for some constant $c \in (0, 1)$ and thus recursing on a list of size $O((1 - c) \cdot n)$. The first step, finding a large independent set, can be performed in $O(\text{sort}(n))$ I/Os and is described in more detail below. The second and fourth step can be performed in a couple of scanning and sorting passes in overall $O(\text{sort}(n))$ I/Os. For example, to update the weights of vertices in $L - I$, it suffices to sort the vertices in I by their successor, sort the vertices in $L - I$ by their vertex ID, and then scan the two sorted lists, and for each pair (x', x) in I with $x = succ(x')$, and $(x, succ(x))$ in $L - I$, we update the weight of x to include the deleted vertex x': $w(x) = w(x) + w(x')$. Similarly, to update the successors of vertices in $L - I$, it suffices to sort I by vertex ID, sort the vertices in $L - I$ by their successor, and then scan the two sorted lists. Once the ranks of the vertices in $L - I$ are computed, it suffices to sort I by vertex ID, sort $L - I$ by successor, and then scan the two lists to update the rank of each vertex $x \in I$. Overall, the I/O-complexity of this list-ranking algorithm is given by the recurrence $T(n) \leq O(\text{sort}(n)) + T(c \cdot n)$, for some constant $c \in (0, 1)$, with solution $O(\text{sort}(n))$ I/Os. This is due to the convexity of the I/O behavior of sorting, i.e., $\text{sort}(c \cdot n) \leq c \cdot \text{sort}(n)$. The algorithm can be used in the parallel setting as well because the scanning here works on pairs of elements that are stored in neighboring cells. For moderately large number of processors, sorting the original instance still dominates the overall running times, and more processors lead to an additional log factor in the number of parallel I/Os [8].

Computing a Large Independent Set of L

There are several algorithms for finding an independent set of a list L that run in $O(\text{sort}(n))$ I/Os. The simplest one is a randomized algorithm by Chiang et al. [4], based on a PRAM algorithm by Anderson and Miller. The idea is to flip a coin for each vertex and then select the vertices whose coin came up heads and their successor's coin came up tail. This produces an independent set of expected size $(n - 1)/4 = \Theta(n)$.

In the serial setting, a different way to compute an independent set of L is to 3-color the vertices in L (assign one of the three colors to each vertex such that no two adjacent vertices have same color) and then pick the most popular of the three colors, an independent set of at least $n/3$ vertices. This can be implemented in $O(\text{sort}(n))$ I/Os using time-forward processing [4]. A more direct algorithm, also based on time-forward processing, is described in [11].

In the parallel setting, time-forward processing is not available. Instead deterministic coin tossing can be used. With permuting complexity, any k-coloring of a list can be transformed to a $\log k$ coloring by what is known as deterministic coin tossing [5]. This immediately leads to a parallel deterministic algorithm with an additional

$\log^* n$ factor in the number of parallel I/Os. Alternatively, as long as sorting is still convex in n, a technique called delayed pointer processing [3] can be used. This can be understood as a parallel version of time-forward processing for a DAG of depth $\log \log n$.

Alternatively, all the CREW-PRAM list-ranking algorithm can be executed on the PEM leading to one parallel I/O per step.

Lower Bounds

There is a permuting complexity lower bound for list ranking, showing that the above-explained algorithm is asymptotically optimal for a large range of the parameters. This was sketched in [4] for the serial case and made precise in [8] by the indivisibility assumption that edges have to be treated as atoms and extended to the parallel case. Observe that in the PEM model, superlinear speedups in the number of processors are possible (the overall available fast memory increases), and hence a lower bound for the serial case does not imply a good lower bound the parallel case.

If the number of processors is high (or for parallel computational models where permuting is easy like BSP or map-reduce), list ranking seems to become more difficult than permuting. All known algorithms are a factor $O(\log n)$ more expensive than permuting. One attempt at an explanation is a lower bound in the mentioned setting where the instance is only gradually revealed (depending on the algorithm already having solved certain other parts of the instance) [8]. For this particular parameter setting, the lower bound shows that the described sorting-based algorithm is optimal.

Applications

List ranking in external memory is particularly useful in connection with *Euler tours* [4]. An Euler tour of an undirected tree $T = (V, E)$ is a traversal of T that visits every edge twice, once in each direction. Such a traversal is represented as a linear list L of edges and can be obtained I/O-efficiently as follows: after fixing an order of the edges $\{v, w_1\}, \ldots, \{v, w_k\}$ incident to each node v of T, we set the successor of $\{w_i, v\}$ in L to be $\{v, w_{i+1}\}$ and the successor of $\{w_k, v\}$ in L to be $\{v, w_1\}$. We break the resulting circular list at some root node r by choosing an edge $\{v, r\}$ with successor $\{r, w\}$, setting the successor of $\{v, r\}$ to be null and marking $\{r, w\}$ to be the first edge of the traversal list L. List ranking on L is then applied in order to lay out the Euler tour on disk in a way that about B consecutive list elements are kept in each block. As an Euler tour reflects the structure of its underlying tree T, many properties of T can be derived from a few scanning and sorting steps on the edge sequence of the Euler tour once it has been stored in a way suitable for I/O-efficient traversal. In fact, this technique is not restricted to external memory but has already been used earlier [9] for parallel (PRAM) algorithms, where the scanning steps are replaced by parallel prefix computations.

Classic tree problems solved with the Euler tour technique include *tree rooting* (finding the parent-child direction of tree edges after a vertex of an unrooted and undirected tree has been chosen to become the root), assigning *pre-/post-/inorder numbers*, and computing *node levels* or *number of descendants*. Euler tours and hence list ranking are also useful for non-tree graphs. For example, they are a basic ingredient of a *clustering* preprocessing step [1] for I/O-efficient breadth first search (BFS) on sparse undirected graphs G: after obtaining a spanning tree T for G, an Euler tour around T is used in order to deterministically obtain low diameter clusters of G.

Experimental Results

Despite their theoretical sorting complexity I/O bound, external-memory list-ranking implementations based on independent set removal suffer from non-negligible constant factors. For small to medium input sizes featuring $n < M^2/(4 \cdot B)$, Sibeyn modifies his connected components algorithm [10] in order to solve practical list-ranking problems in scanning complexity with a small constant factor ($22 \cdot n/B$ I/Os). The algorithm splits the input list into at most $M/(2 \cdot B)$ subproblems of $M/2$ consecutive node indices

each and processes these subproblems in two passes (the first one running from high to low index ranges and the second one vice versa).

For all nodes of the current range, Sibeyn's algorithm follows the links leading to the nodes of the same sublists and updates the information on their final node and the number of links to it. For all nodes with links running outside the current sublist, the required information is requested in a batched fashion from the subproblems containing the nodes to which they are linked. Phase-one-requests from and phase-two-answers to the sublists are processed only when the wave through the data hits the corresponding subproblem. Due to the overall size restriction, a buffer of size $\Theta(B)$ can be kept in main memory for each subproblem in order to facilitate I/O-efficient information transfer between the subproblems.

The implementation of Sibeyn's algorithm in the STXXL [6] framework has been used as a building block in the engineering of many graph traversal algorithms [1]. For example, the improved clustering preprocessing based on list ranking and Euler tours helped in reducing the I/O wait time for BFS by up to two orders of magnitude compared to a previous clustering method.

Cross-References

- ▸ Cache-Oblivious Model
- ▸ External Sorting and Permuting
- ▸ I/O-Model

Recommended Reading

1. Ajwani D, Meyer U (2009) Design and engineering of external memory traversal algorithms for general graphs. In: Lerner J, Wagner D, Zweig KA (eds) Algorithmics of large and complex networks. Springer, Berlin/Heidelberg, pp 1–33
2. Arge L, Goodrich M, Nelson M, Sitchinava N (2008) Fundamental parallel algorithms for private-cache chip multiprocessors. In: SPAA 2008, pp 197–206
3. Arge L, Goodrich M, Sitchinava N (2010) Parallel external memory graph algorithms. In: IPDPS. IEEE, pp 1–11. http://dx.doi.org/10.1109/IPDPS.2010.5470440
4. Chiang Y, Goodrich M, Grove E, Tamassia R, Vengroff D, Vitter J (1995) External memory graph algorithms. In: Proceedings of the 6th annual symposium on discrete algorithms (SODA), San Francisco, pp 139–149
5. Cole R, Vishkin U (1986) Deterministic coin tossing with applications to optimal parallel list ranking. Inf Control 70(1):32–53
6. Dementiev R, Kettner L, Sanders P (2008) STXXL: standard template library for XXL data sets. Software: Pract Exp 38(6): 589–637
7. Greiner G (2012) Sparse matrix computations and their I/O complexity. Dissertation, Technische Universität München, München. http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20121123-1113167-0-6
8. Jacob R, Lieber T, Sitchinava N (2014) On the complexity of list ranking in the parallel external memory model. In: Proceedings 39th international symposium on mathematical foundations of computer science (MFCS'14), Budapest. LNCS, vol 8635. Springer, pp 384–395
9. JáJá J (1992) An introduction to parallel algorithms. Addison-Wesley, Reading
10. Sibeyn J (2004) External connected components. In: Proceedings of the 9th Scandinavian workshop on algorithm theory (SWAT), Lecture Notes in Computer Science, vol 3111. Humlebaek, pp 468–479. http://link.springer.com/chapter/10.1007/978-3-540-27810-8_40
11. Zeh N (2002) I/O-efficient algorithms for shortest path related problems. Phd thesis, School of Computer Science, Carleton University

List Scheduling

Leah Epstein
Department of Mathematics, University of Haifa, Haifa, Israel

Keywords

Online scheduling on identical machines

Years and Authors of Summarized Original Work

1966; Graham

Problem Definition

The paper of Graham [8] was published in the 1960s. Over the years, it served as a common example of online algorithms (though the original algorithm was designed as a simple approximation heuristic). The following basic setting is considered.

A sequence of n jobs is to be assigned to m identical machines. Each job should be assigned to one of the machines. Each job has a size associated with it, which can be seen as its processing time or its load. The load of a machine is the sum of sizes of jobs assigned to it. The goal is to minimize the maximum load of any machine, also called the makespan. We refer to this problem as JOB SCHEDULING.

If jobs are presented one by one and each job needs to be assigned to a machine in tur, without any knowledge of future jobs, the problem is called online. Online algorithms are typically evaluated using the (absolute) *competitive ratio*, which is similar to the *approximation ratio* of approximation algorithms. For an algorithm $\mathcal{A}$, we denote its cost by $\mathcal{A}$ as well. The cost of an optimal offline algorithm that knows the complete sequence of jobs is denoted by OPT. The competitive ratio of an algorithm $\mathcal{A}$ is the infimum $\mathcal{R} \geq 1$ such that for any input, $\mathcal{A} \leq \mathcal{R} \cdot \mathrm{OPT}$.

Key Results

In paper [8], Graham defines an algorithm called LIST SCHEDULING (LS). The algorithm receives jobs one by one. Each job is assigned in turn to a machine which has a minimal current load. Ties are broken arbitrarily.

The main result is the following:

Theorem 1 LS *has a competitive ratio of* $2 - \frac{1}{m}$.

Proof Consider a schedule created for a given sequence. Let ℓ denote a job that determines the makespan (that is, the last job assigned to a machine i that has a maximum load), let L denote its size, and let X denote the total size of all other jobs assigned to i. At the time when L was assigned to i, this was a machine of minimum load. Therefore, the load of each machine is at least X. The makespan of an optimal schedule (i.e., a schedule that minimizes the makespan) is the cost of an optimal offline algorithm and thus is denoted by OPT. Let P be the sum of all job sizes in the sequence.

The two following simple lower bounds on OPT can be obtained:

$$\mathrm{OPT} \geq L. \tag{1}$$

$$\mathrm{OPT} \geq \frac{p}{m} \geq \frac{m \cdot X + L}{m} = X + \frac{L}{m}. \tag{2}$$

Inequality (1) follows from the fact that {OPT} needs to run job ℓ and thus at least one machine has a load of at least L. The first inequality in (2) is due to the fact that at least one machine receives at least a fraction $\frac{1}{m}$ of the total size of jobs. The second inequality in (2) follows from the comments above on the load of each machine.

This proves that the makespan of the algorithm, $X + L$ can be bounded as follows:

$$\begin{aligned} X + L &\leq \mathrm{OPT} + \frac{m-1}{m} L \leq \mathrm{OPT} + \frac{m-1}{m}\,\mathrm{OPT} \\ &= (2 - 1/m)\,\mathrm{OPT}. \end{aligned} \tag{3}$$

The first inequality in (3) follows from (2) and the second one from (1).

To show that the analysis is tight, consider $m(m-1)$ jobs of size 1 followed by a single job of size m. After the smaller jobs arrive, LS obtains a balanced schedule in which every machine has a load of $m-1$. The additional job increases the makespan to $2m-1$. However, an optimal offline solution would be to assign the smaller jobs to $m-1$ machines and the remaining job to the remaining machine, getting a load of m.

A natural question was whether this bound is best possible. In a later paper, Graham [9] showed that applying LS with a sorted sequence of jobs (by nonincreasing order of sizes) actually gives

a better upper bound of $\frac{4}{3} - \frac{1}{3m}$ on the approximation ratio. A polynomial time approximation scheme was given by Hochbaum and Shmoys in [10]. This is the best offline result one could hope for as the problem is known to be NP hard in the strong sense.

As for the online problem, it was shown in [5] that no (deterministic) algorithm has a smaller competitive ratio than $2 - \frac{1}{m}$ for the cases $m = 2$ and $m = 3$. On the other hand, it was shown in a sequence of papers that an algorithm with a smaller competitive ratio can be found for any $m \geq 4$, and even algorithms with a competitive ratio that does not approach 2 for large m were designed.

The best such result is by Fleischer and Wahl [6], who designed a 1.9201-competitive algorithm. Lower bounds of 1.852 and 1.85358 on the competitive ratio of any online algorithm were shown in [1,7]. Rudin [13] claimed a better lower bound of 1.88.

Applications

As the study of approximation algorithms and specifically online algorithms continued, the analysis of many scheduling algorithms used similar methods to the proof above. Below, several variants of the problem where almost the same proof as above gives the exact same bound are mentioned.

Load Balancing of Temporary Tasks

In this problem, the sizes of jobs are seen as loads. Time is a separate axis. The input is a sequence of events, where every event is an arrival or a departure of a job. The set of active jobs at time t is the set of jobs that have already arrived at this time and have not departed yet. The cost of an algorithm at a time t is its makespan at this time. The cost of an algorithm is its maximum cost over time. It turns out that the analysis above can be easily adapted for this model as well. It is interesting to note that in this case, the bound $2 - \frac{1}{m}$ is actually best possible, as shown in [2].

Scheduling with Release Times and Precedence Constraints

In this problem, the sizes represent processing times of jobs. Various versions have been studied. Jobs may have designated release times, which are the times when these jobs become available for execution. In the online scenario, each job arrives and becomes known to the algorithm only at its release time. Some precedence constraints may also be specified, defined by a partial order on the set of jobs. Thus, a job can be run only after its predecessors complete their execution. In the online variant, a job becomes known to the algorithm only after its predecessors have been completed. In these cases, LS acts as follows. Once a machine becomes available, a waiting job that arrived earliest is assigned to it. (If there is no waiting job, the machine is idle until a new job arrives).

The upper bound of $2 - \frac{1}{m}$ on the competitive ratio can be proved using a relation between the cost of an optimal schedule and the amount of time when at least one machine is idle (See [14] for details).

This bound is tight for several cases. For the case where there are release times, no precedence constraints, and processing times (sizes) are not known upon arrival, Shmoys, Wein, and Williamson [15] proved a lower bound of $2 - \frac{1}{m}$. For the case where there are only precedence constraints (no release times, and sizes of jobs are known upon arrival), a lower bound of the same value appeared in [4]. Note that the case with clairvoyant scheduling (i.e., sizes of jobs are known upon arrival), release times, and no precedence constraints is not settled. For $m = 2$, it was shown by Noga and Seiden [11] that the tight bound is $(5 - \sqrt{5})/2 \approx 1.38198$, and the upper bound is achieved using an algorithm that applies waiting with idle machines rather than scheduling a job as soon as possible, as done by LS.

Open Problems

The most challenging open problem is to find the best possible competitive ratio for this basic online problem of job scheduling. The gap between

the upper bound and the lower bound is not large, yet it seems very difficult to find the exact bound. A possibly easier question would be to find the best possible competitive ratio for $m = 4$. A lower bound of $\sqrt{3} \approx 1.732$ has been shown by [12], and the currently known upper bound is 1.733 by [3]. Thus, it may be the case that this bound would turn out to be $\sqrt{3}$.

Cross-References

- Approximation Schemes for Makespan Minimization
- Robust Scheduling Algorithms

Recommended Reading

1. Albers S (1999) Better bounds for online scheduling. SIAM J Comput 29(2):459–473
2. Azar Y, Epstein L (2004) On-line load balancing of temporary tasks on identical machines. SIAM J Discret Math 18(2):347–352
3. Chen B, van Vliet A, Woeginger GJ (1994) New lower and upper bounds for on-line scheduling. Oper Res Lett 16:221–230
4. Epstein L (2000) A note on on-line scheduling with precedence constraints on identical machines. Inf Process Lett 76:149–153
5. Faigle U, Kern W, Turán G (1989) On the performane of online algorithms for partition problems. Acta Cybern 9:107–119
6. Fleischer R, Wahl M (2000) On-line scheduling revisited. J Sched 3:343–353
7. Gormley T, Reingold N, Torng E, Westbrook J (2000) Generating adversaries for request-answer games. In: Proceedings of the 11th symposium on discrete algorithms (SODA2000), San Francisco, pp 564–565
8. Graham RL (1966) Bounds for certain multiprocessing anomalies. Bell Syst Technol J 45:1563–1581
9. Graham RL (1969) Bounds on multiprocessing timing anomalies. SIAM J Appl Math **17**, 263–269
10. Hochbaum DS, Shmoys DB (1987) Using dual approximation algorithms for scheduling problems: theoretical and practical results. J ACM 34(1):144–162
11. Noga J, Seiden SS (2001) An optimal online algorithm for scheduling two machines with release times. Theor Comput Sci 268(1):133–143
12. Rudin JF III, Chandrasekaran R (2003) Improved bounds for the online scheduling problem. SIAM J Comput 32:717–735
13. Rudin JF III (2001) Improved bounds for the online scheduling problem. Ph.D. thesis, The University of Texas at Dallas
14. Sgall J (1998) On-line scheduling. In: Fiat A, Woeginger GJ (eds) Online algorithms: the state of the art. Springer, Berlin/New York, pp 196–231
15. Shmoys DB, Wein J, Williamson DP (1995) Scheduling parallel machines on-line. SIAM J Comput 24:1313–1331

Local Alignment (with Affine Gap Weights)

Henry Leung
Department of Computer Science, The University of Hong Kong, Hong Kong, China

Keywords

Affine gap penalty; Local alignment; Mapping; Pairwise alignment

Years and Authors of Summarized Original Work

1986; Altschul, Erickson

Problem Definition

The pairwise local alignment problem is concerned with identification of a pair of similar substrings from two molecular sequences. This problem has been studied in computer science for four decades. However, most problem models were generally not biologically satisfying or interpretable before 1974. In 1974, Sellers developed a metric measure of the similarity between molecular sequences. [9] generalized this metric to include deletions and insertions of arbitrary length which represent the minimum number of mutational events required to convert one sequence into another.

Given two sequences S and T, a pairwise alignment is a way of inserting space characters '_' in S and T to form sequences S' and T' respectively with the same length. There can be different alignments of two sequences. The score of an alignment is measured by a scoring metric $\delta(x, y)$. At each position i where both x and

y are not spaces, the similarity between $S'[i]$ and $T'[i]$ is measured by $\delta(S'[i], T'[j])$. Usually, $\delta(x, y)$ is positive when x and y are the same and negative when x and y are different. For positions with consecutive space characters, the alignment scores of the space characters are not considered independently; this is because inserting or deleting a long region in molecular sequences is more likely to occur than inserting or deleting several short regions. Smith and Waterman use an affine gap penalty to model the similarity at positions with space characters. They define a consecutive substring with spaces in S' or T' as a gap. For each length l gap, they give a linear penalty $W_k = W_s + l \times W_p$ for some predefined positive constants W_s and W_p. The score of an alignment is the sum of the score at each position i minus the penalties of each gap. For example, the alignment score of the following alignment is $\delta(G, G) + \delta(C, C) + \delta(C, C) + \delta(U, C) + \delta(G, G) - (W_s + 2 \times W_p)$.

$$S : GCCAUUG$$
$$T : GCC\_\_CG$$

The optimal global alignment of sequences S and T is the alignment of S and T with the maximum alignment score.

Sometimes we want to know whether sequences S and T contain similar substrings instead of whether S and T are similar. In this case, they solve the pairwise local alignment problem, which wants to find a substring U in S and another substring V in T such that the global alignment score of U and V is maximized.

Pairwise Local Alignment Problem

Input: Two sequences $S[1 \ldots n]$ and $T[1 \ldots m]$.

Output: A substring U in S and a substring V in T such that the optimal global alignment of U and V is maximized.

O(mn) time and O(mn) space algorithm is based on dynamic programming.

The pairwise local alignment problem can be solved in O(mn) time and O(mn) space by dynamic programming. The algorithm needs to fill in the $4\,m \times n$ tables H, H_N, H_S, and H_T, where each entry takes constant time. The individual meanings of these 4 tables are as follows.

$H(i, j)$: maximum score of the global alignment of U and V over all suffixes U in $S[1 \ldots i]$ and all suffixes V in $T[1 \ldots j]$.

$H_N(i, j)$: maximum score of the global alignment of U and V over all suffixes U in $S[1 \ldots i]$ and all suffixes V in $T[1 \ldots j]$, with the restriction that $S[i]$ and $T[j]$ must be aligned.

$H_S(i, j)$: maximum score of the global alignment of U and V over all suffixes U in $S[1 \ldots i]$ and all suffixes V in $T[1 \ldots j]$, with $S[j]$ aligned with a space character.

$H_T(i, j)$: maximum score of the global alignment of U and V over all suffixes U in $S[1 \ldots i]$ and all suffixes V in $T[1 \ldots j]$, with $T[j]$ aligned with a space character.

The optimal local alignment score of S and T will be $\max\{H(i, j)\}$, and the local alignment of S and T can be found by tracking back table H.

In the tables, each entry can be filled in by the following recursion in constant time.

Basic Step

$$H(i, 0) = H(0, j) = 0, \qquad 0 \leqslant i \leqslant n,\ 0 \leqslant i \leqslant m$$
$$H_N(i, 0) = H_N(0, j) = -\infty, \qquad 0 \leqslant i \leqslant n,\ 0 \leqslant i \leqslant m$$
$$H_s(i, 0) = H_T(0, j) = W_s + W_p, \quad 0 \leqslant i \leqslant n,\ 0 \leqslant i \leqslant m$$
$$H_s(0, j) = H_T(i, 0) = -\infty, \qquad 0 \leqslant i \leqslant n,\ 0 \leqslant i \leqslant m$$

Recursion Step

$$H(i,j) = \max\{H_N(i,j), H_s(i,j), H_T(i,j), 0\}, \quad 1 \leqslant i \leqslant n, \ 1 \leqslant i \leqslant m$$

$$H_N(i,j) = H(i-1, j-1) + \delta(S[i], T[j]), \quad 1 \leqslant i \leqslant n, \ 1 \leqslant i \leqslant m$$

$$H_s(i,j) = \max\{H(i-1,j) - (W_s + W_p), H_S(i-1,j) - W_p\}, \quad 1 \leqslant i \leqslant n, \ 1 \leqslant i \leqslant m$$

$$H_T(i,j) = \max\{H(i,j-1) - (W_s + W_p), H_T(i,j-1) - W_p\}, \quad 1 \leqslant i \leqslant n, \ 1 \leqslant i \leqslant m$$

Applications

Local alignment with affine gap penalty can be used for protein classification, phylogenetic footprinting, and identification of functional sequence elements.

URL to Code

http://bioweb.pasteur.fr/seqanal/interfaces/water.html

Recommended Reading

1. Allgower EL, Schmidt PH (1985) An algorithm for piecewise-linear approximation of an implicitly defined manifold. SIAM J Num Anal 22:322–346
2. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990) Basic local alignment search tool. J Mol Biol 215:403–410
3. Chao KM, Miller W (1995) Linear-space algorithms that build local alignments from fragments. Algorithmica 13:106–134
4. Gusfield D (1999) Algorithms on strings, trees and sequences. Cambridge University Press, Cambridge. ISBN:052158519
5. Ma B, Tromp J, Li M (2002) PatternHunter: faster and more sensitive homology search. Bioinformatics 18:440–445
6. Myers EW, Miller W (1988) Optimal alignments in linear space. Bioinformatics 4:11–17
7. Needleman SB, Wunsch CD (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol 48:443–453
8. Pearson WR, Lipman DJ (1988) Improved tools for biological sequence comparison. Proc Natl Acad Sci USA 85:2444–2448
9. Sellers PH (1974) On the theory and computation of evolutionary distances. SIAM J Appl Math 26:787–793
10. Smith TF, Waterman MS (1981) Identification of common molecular subsequences. J Mol Biol 147:195–197

Local Alignment (with Concave Gap Weights)

S.M. Yiu
Department of Computer Science, University of Hong Kong, Hong Kong, China

Keywords

Pairwise local alignment; Sequence alignment

Years and Authors of Summarized Original Work

1988; Miller, Myers

Problem Definition

This work of Miller and Myers [11] deals with the problem of pairwise sequence alignment in which the distance measure is based on the gap penalty model. They proposed an efficient algorithm to solve the problem when the gap penalty is a concave function of the gap length.

Let X and Y be two strings (sequences) of alphabet Σ. The pairwise alignment $\mathcal{A}$ of X and Y maps X, Y into strings X', Y' that may contain spaces (not in Σ) such that (1) $|X'| = |Y'| = \ell$; (2) removing spaces from X' and Y' returns X and Y, respectively; and (3) for any $1 \leq i \leq \ell$, $X'[i]$ and $Y'[i]$ cannot be both spaces where $X'[i]$ denotes the ith character in X'.

To evaluate the quality of an alignment, there are many different measures proposed (e.g., edit distance, scoring matrix [12]). In this work, they consider the *gap penalty* model.

A *gap* in an alignment $\mathcal{A}$ of X and Y is a maximal substring of contiguous spaces in either X' or Y'. There are gaps and aligned characters (both $X'[i]$ and $Y'[i]$ are not spaces) in an alignment. The score for a pair of aligned characters is based on a distance function $\delta(a, b)$ where $a, b \in \Sigma$. Usually δ is a metric, but this assumption is not required in this work. The penalty of a gap of length k is based on a nonnegative function $W(k)$. The score of an alignment is the sum of the scores of all aligned characters and gaps. An alignment is *optimal* if its score is the minimum possible.

The penalty function $W(k)$ is *concave* if $\triangle W(k) \geq \triangle W(k+1)$ for all $k \geq 1$, where $\triangle W(k) = W(k+1) - W(k)$.

The penalty function $W(k)$ is *affine* if $W(k) = a + bk$ where $a and b$ are constants. Affine function is a special case of concave function. The problem for affine gap penalty has been considered in [1, 7].

The penalty function $W(k)$ is a *P-piece affine curve* if the domain of W can be partitioned into P intervals, $(\tau_1 = 1, \chi_1), (\tau_2, \chi_2), \ldots, (\tau_p, \chi_p = \infty)$, where $\tau_i = \chi_{i-1} + 1$ for all $1 < i \leq p$, such that for each interval, the values of W follow an affine function. More precisely, for any $k \in (\tau_i, \chi_i), W(k) = a_i + b_i k$ for some constants a_i, b_i.

Problem

Input: Two strings X and Y, the scoring function δ, and the gap penalty function $W(k)$.
Output: An optimal alignment of X and Y.

Key Results

Theorem 1 *If $W(k)$ is concave, they provide an algorithm for computing an optimal alignment that runs in $O(n^2 \log n)$ time where n is the length of each string and uses $O(n)$ expected space.*

Corollary 1 *If $W(k)$ is an affine function, the same algorithm runs in $O(n^2)$ time.*

Theorem 2 *For some special types of gap penalty functions, the algorithm can be modified to run faster.*

- *If $W(k)$ is a P-piece affine curve, the algorithm can be modified to run in $O(n^2 \log P)$ time.*
- *For logarithmic gap penalty function, $W(k) = a + b \log k$, the algorithm can be modified to run in $O(n^2)$ time.*
- *If $W(k)$ is a concave function when $k > K$, the algorithm can be modified to run in $O(K + n^2 \log n)$ time.*

Applications

Pairwise sequence alignment is a fundamental problem in computational biology. Sequence similarity usually implies functional and structural similarity. So, pairwise alignment can be used to check whether two given sequences have similar functions or structures and to predict functions of newly identified DNA sequence. One can refer to Gusfield's book for some examples on the importance of sequence alignment (pp. 212–214 of [8]).

The alignment problem can be further divided into the *global* alignment problem and the *local* alignment problem. The problem defined here is the global alignment problem in which the whole input strings are required to align with each other. On the other hand, for local alignment, the main interest lies in identifying a substring from each of the input strings such that the alignment score of the two substrings is the minimum among all possible substrings. Local alignment is useful in aligning sequences that are not similar, but contain a region that are highly conserved (similar). Usually this region is a functional part (domain) of the sequences. Local alignment is particularly useful in comparing proteins. Proteins in the same family from different species usually have some functional domains that are highly conserved while the other parts are not similar at all. Examples are the homeobox genes [4] for

which the protein sequences are quite different in each species except the functional domain *homeodomain*.

Conceptually, the alignment score is used to capture the evolutionary distance between the two given sequences. Since a gap of more than one space can be created by a single mutational event, considering a gap of length k as a unit instead of k different point mutation may be more appropriate in some cases. However, which gap penalty function should be used is a difficult question to answer and sometimes depends on the actual applications. Most applications, such as BLAST, uses the affine gap penalty which is still the dominate model in practice. On the other hand, Benner et al. [2] and Gu and Li [9] suggested to use the logarithmic gap penalty in some cases. Whether using a concave gap penalty function in general is meaningful is still an open issue.

Open Problem

Note that the results of this paper have been independently obtained by Galil and Giancarlo [6], and for affine gap penalty, Gotoh [7] also gave an $O(n^2)$ algorithm for solving the alignment problem. In [5], Eppstein gave a faster algorithm that runs in $O(n^2)$ time for solving the same sequence alignment problem with concave gap penalty function. Whether a subquadratic algorithm exists for solving this problem remains open. As a remark, subquadratic algorithms do exist for solving the sequence alignment problem if the measure is not based on the gap penalty model, but is computed as $\sum_{i=1}^{\ell} \delta(X1'[i], Y'[i])$ based only on a scoring function $\delta(a, b)$ where $a, b \in \Sigma \cup \{\_\}$ where '_' represents the space [3, 10].

Experimental Results

They have performed some experiments to compare their algorithm with Waterman's $O(n^3)$ algorithm [13] on a number of different concave gap penalty functions. Artificial sequences are generated for the experiments. Results from their experiments lead to their conjectures that Waterman's method runs in $O(n^3)$ time when the two given strings are very similar or the score for mismatch characters is small and their algorithm runs in $O(n^2)$ time if the range of the function $W(k)$ is not functionally dependent on n.

Cross-References

▸ Local Alignment (with Affine Gap Weights)

Recommended Reading

1. Altschul SF, Erickson BW (1986) Optimal sequence alignment using affine gap costs. Bull Math Biol 48:603–616
2. Benner SA, Cohen MA, Gonnet GH (1993) Empirical and structural models for insertions and deletions in the divergent evolution of proteins. J Mol Biol 229:1065–1082
3. Crochemore M, Landau GM, Ziv-Ukelson M (2003) A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM J Comput 32(6):1654–1673
4. De Roberts E, Oliver G, Wright C (1990) Homeobox genes and the vertibrate body plan. Sci Am 263(1):46-52
5. Eppstein D (1990) Sequence comparison with mixed convex and concave costs. J Algorithms 11(1):85–101
6. Galil Z, Giancarlo R (1989) Speeding up dynamic programming with applications to molecular biology. Theor Comput Sci 64:107–118
7. Gotoh O (1982) An improved algorithm for matching biological sequences. J Mol Biol 162:705–708
8. Gusfield D (1997) Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, Cambridge
9. Li W-H, Gu X (1995) The size distribution of insertions and deletions in human and rodent pseudogenes suggests the logarithmic gap penalty for sequence alignment. J Mol Evol 40:464–473
10. Masek WJ, Paterson MS (1980) A fater algorithm for computing string edit distances. J Comput Syst Sci 20:18–31
11. Miller W, Myers EW (1988) Sequence comparison with concave weighting functions. Bull Math Biol 50(2):97–120
12. Sankoff D, Kruskal JB (1983) Time warps, strings edits, and macromolecules: the theory and practice of sequence comparison. Addison-Wesley, Reading
13. Waterman MS (1984) Efficient sequence alignment algorithms. J Theor Biol 108:333–337

Local Approximation of Covering and Packing Problems

Fabian Kuhn
Department of Computer Science, ETH Zurich, Zurich, Switzerland

Synomyms

Distributed approximation of covering and packing problems

Years and Authors of Summarized Original Work

2003–2006; Kuhn, Moscibroda, Nieberg, Wattenhofer

Problem Definition

A *local algorithm* is a distributed algorithm on a network with a running time which is independent or almost independent of the network's size or diameter. Usually, a distributed algorithm is called local if its time complexity is at most polylogarithmic in the size n of the network. Because the time needed to send information from one node of a network to another is at least proportional to the distance between the two nodes, in such an algorithm, each node's computation is based on information from nodes in a close vicinity only. Although all computations are based on local information, the network as a whole typically still has to achieve a global goal. Having local algorithms is inevitable to obtain time-efficient distributed protocols for large-scale and dynamic networks such as peer-to-peer networks or wireless ad hoc and sensor networks.

In [2, 6, 7], Kuhn, Moscibroda, and Wattenhofer describe upper and lower bounds on the possible trade-off between locality (time complexity) of distributed algorithms and the quality (approximation ratio) of the achievable solution for an important class of problems called covering and packing problems. Interesting covering and packing problems in the context of networks include minimum dominating set, minimum vertex cover, maximum matching, as well as certain flow maximization problems. All the results given in [2, 6, 7] hold for general network topologies. Interestingly, it is shown by Kuhn, Moscibroda, Nieberg, and Wattenhofer in [3, 4, 5] that covering and packing problems can be solved much more efficiently when assuming that the network topology has special properties which seem realistic for wireless networks.

Distributed Computation Model

In [2, 3, 4, 5, 6, 7], the network is modeled as an undirected and except for [5] unweighted graph $G = (V, E)$. Two nodes $u, v \in V$ of the network are connected by an edge $(u, v) \in E$ whenever there is a direct bidirectional communication channel connecting u and v. In the following, the number of nodes and the maximal degree of G are denoted by $n = |V|$ and by Δ.

For simplicity, communication is assumed to be synchronous. That is, all nodes start an algorithm simultaneously and time is divided into rounds. In each round, every node can send an arbitrary message to each of its neighbors and perform some local computation based on the information collected in previous rounds. The *time complexity* of a synchronous distributed algorithm is the number of rounds until all nodes terminate.

Local distributed algorithms in the described synchronous model have first been considered in [8] and [9]. As an introduction to the above and similar distributed computation models, it is also recommended to read [11].

Distributed Covering and Packing Problems

A fractional covering problem (P) and its dual fractional packing problem (D), are linear programs (LPs) of the canonical forms

$$\begin{array}{lll} \min & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} & \\ \text{s.t.} & A \cdot \boldsymbol{x} \geq \boldsymbol{b} & \text{(P)} \\ & \boldsymbol{x} \geq \boldsymbol{0} & \end{array} \qquad \begin{array}{lll} \max & \boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} & \\ \text{s.t.} & A^{\mathrm{T}} \cdot \boldsymbol{y} \leq \boldsymbol{c} & \text{(D)} \\ & \boldsymbol{y} \geq \boldsymbol{0} & \end{array}$$

L

where all a_{ij}, b_i, and c_i are non-negative. In a distributed context, finding a small (weighted) dominating set or a small (weighted) vertex cover of the network graph are the most important covering problems. A dominating set of a graph G is a subset S of its nodes such that all nodes of G either are in S or have a neighbor in S. The dominating set problem can be formulated as covering integer LP by setting A to be the adjacency matrix with 1s in the diagonal, by setting $\boldsymbol{b}$ to be a vector with all 1s and if $\boldsymbol{c}$ is the weight vector. A vertex cover is a subset of the nodes such that all edges are covered. Packing problems occur in a broad range of resource allocation problems. As an example, in [1] and [10], the problem of assigning flows to a given fixed set of paths is described. Another common packing problem is (weighted) maximum matching, the problem of finding a largest possible set of pairwise non-adjacent edges.

While computing a dominating set, vertex cover, or matching of the network graph are inherently distributed tasks, general covering and packing LPs have no immediate distributed meaning. To obtain a distributed version of these LPs, two dual LPs (P) and (D) are mapped to a bipartite network as follows. For each primal variable x_i and for each dual variable y_j, there are nodes $v_i{}^p$ and $v_j{}^d$, respectively. There is an edge between two nodes $v_i{}^p$ and $v_j{}^d$ whenever $a_{ji} \neq 0$, i.e., there is an edge if the ith variable of an LP occurs in its jth inequality.

In most real-world examples of distributed covering and packing problems, the network graph is of course not equal to the described bipartite graph. However, it is usually straightforward to simulate an algorithm which is designed for the above bipartite network on the actual network graph without affecting time and message complexities.

Bounded Independence Graphs

In [3, 4, 5], local approximation algorithms for covering and packing problems for graphs occuring in the context of wireless ad hoc and sensor networks are studied. Because of scale, dynamism and the scarcity of resources, these networks are a particular interesting area to apply local distributed algorithms.

Wireless networks are often modeled as *unit disk graphs* (UDGs): Nodes are assumed to be in a two-dimensional Euclidean plane and two nodes are connected by an edge iff their distance is at most 1. This certainly captures the inherent geometric nature of wireless networks. However, unit disk graphs seem much too restrictive to accurately model real wireless networks. In [3, 4, 5], Kuhn et. al. therefore consider two generalizations of the unit disk graph model, *bounded independent graphs* (BIGs) and *unit ball graphs* (UBGs). A BIG is a graph where all local independent sets are of bounded size. In particular, it is assumed that there is a function $I(r)$ which upper bounds the size of the largest independent set of every r-neighborhood in the graph. Note that the value of $I(r)$ is independent of n, the size of the network. If $I(r)$ is a polynomial in r, a BIG is said to be polynomially bounded. UDGs are BIGs with $I(r) \in O(r^2)$. UBGs are a natural generalization of UDGs. Given some underlying metric space (V, d) two nodes $u, v \in V$ are connected by an edge iff $d(u, v) \leq 1$. If the metric space (V, d) has constant doubling dimension, (The doubling dimension of a metric space is the logarithm of the maximal number of balls needed to cover a ball $B_r(x)$ in the metric space with balls $B_{r/2}(y)$ of half the radius), a UBG is a polynomially bounded BIG.

Key Results

The first algorithms to solve general distributed covering and packing LPs appear in [1, 10]. In [1], it is shown that it is possible to find a solution which is within a factor of $1 + \varepsilon$ of the optimum in $O(\log^3(\rho n)/\varepsilon^3)$ rounds where ρ is the ratio between the largest and the smallest non-zero coefficient of the LPs. The result of [1] is improved and generalized in [6, 7] where the following result is proven:

Theorem 1 *In k rounds, (P) and (D) can be approximated by a factor of* $(\rho\Delta)^{O(1/\sqrt{k})}$

using messages of size at most $O(\log(\rho\Delta))$. *An* $(1+\varepsilon)$*-approximation can be found in time* $O(\log^2(\rho\Delta)/\varepsilon^4)$.

The algorithm underlying Theorem 1 needs only small messages of size $O(\log(\rho\Delta))$ and extremely simple and efficient local computations. If larger messages and more complicated (but still polynomial) local computations are allowed, it is possible to improve the result of Theorem 1:

Theorem 2 *In k rounds, LPs of the form (P) or (D) can be approximated by a factor of* $O(n^{O(1/k)})$. *This implies that a constant approximation can be found in time* $O(\log n)$.

Theorems 1 and 2 only give bounds on the quality of distributed solutions of covering and packing LPs. However, many of the practically relevant problems are integer versions of covering and packing LPs. Combined with simple randomized rounding schemes, the following upper bounds for dominating set, vertex cover, and matching are proven in [6, 7]:

Theorem 3 *Let* Δ *be the maximal degree of the given network graph. In k rounds, minimum dominating set can be approximated by a factor of* $O(\Delta^{O(1/\sqrt{k})} \cdot \log \Delta)$ *in expectation by using messages of size* $O(\Delta)$. *Without bound on the message size, an expected approximation ratio of* $O(n^{O(1/k)} \cdot \log \Delta)$ *can be achieved. Minimum vertex cover and maximum matching can both be approximated by a factor of* $O(\Delta^{1/k})$ *in k rounds.*

In [2, 7], it is shown that the upper bounds on the trade-offs between time complexity and approximation ratio given by Theorems 1–3 are almost optimal:

Theorem 4 *In k rounds, it is not possible to approximate minimum vertex cover better than by factors of* $\Omega(\Delta^{1/k}/k)$ *and* $\Omega(n^{\Omega(1/k^2)}/k)$. *This implies time lower bounds of* $\Omega(\log \Delta / \log\log \Delta)$ *and* $\Omega(\sqrt{\log n / \log\log n})$ *for constant or even poly-logarithmic approximation ratios. The same bounds hold for minimum dominating set, for maximum matching, as well as for the underlying LPs.*

While Theorem 4 shows that the results given by Theorems 1–3 are close to optimal for worst-case network topologies, the problems might be much simpler if restricted to networks which actually occur in reality. In fact, it is shown in [3, 4, 5] that the above results can indeed be improved if the network graph is assumed to be a BIG or a UBG with constant doubling dimension. In [5], the following result for UBGs is proven:

Theorem 5 *Assume that the network graph* $G = (V, E)$ *is a UBG with underlying metric (V, d). If (V, d) has constant doubling dimension and if all nodes know the distances to their neighbors in G up to a constant factor, it is possible to find constant approximations for minimum dominating set, minimum vertex cover, maximum matching, as well as for general LPs of the forms (P) and (D) in* $O(\log^* n)$ *rounds.* (*The log-star function* $\log^* n$ *is an extremely slowly increasing function which gives the number of times the logarithm has to be taken to obtain a number smaller than 1.*)

While the algorithms underlying the results of Theorems 1 and 2 for solving covering and packing LPs are deterministic or straightforward to be derandomized, all known efficient algorithms to solve minimum dominating set and more complicated integer covering and packing problems are randomized. Whether there are good deterministic local algorithms for dominating set and related problems is a long-standing open question. In [3], it is shown that if the network is a BIG, efficient deterministic distributed algorithms exist:

Theorem 6 *On a BIG it is possible to find constant approximations for minimum dominating set, minimum vertex cover, maximum matching, as well as for LPs of the forms (P) and (D) deterministically in* $O(\log \Delta \cdot \log^* n)$ *rounds.*

In [4], it is shown that on polynomially bounded BIGs, one can even go one step further and efficiently find an arbitrarily good approximation by a distributed algorithm:

Theorem 7 *On a polynomially bounded BIG, there is a local approximation scheme which*

computes a $(1+\varepsilon)$*-approximation for minimum dominating set in time* $O(\log \Delta \log^*(n)/\varepsilon + 1/\varepsilon^{O(1)})$*. If the network graph is a UBG with constant doubling dimension and nodes know the distances to their neighbors, a* $(1+\varepsilon)$*-approximation can be computed in* $O(\log^*(n)/\varepsilon + 1/\varepsilon^{O(1)})$ *rounds.*

Applications

The most important application environments for local algorithms are large-scale decentralized systems such as wireless ad hoc and sensor networks or peer-to-peer networks. On such networks, only local algorithms lead to scalable systems. Local algorithms are particularly well-suited if the network is dynamic and computations have to be repeated frequently.

A particular application of the minimum dominating set problem is the task of clustering the nodes of wireless ad hoc or sensor networks. Assigning each node to an adjacent node in a dominating set induces a simple clustering of the nodes. If the nodes of the dominating set (i.e., the cluster centers) are connected with each other by using additional nodes, the resulting structure can be used as a backbone for routing.

Open Problems

There are a number of open problems related to the distributed approximation of covering and packing problems in particular and to distributed approximation algorithms in general. The most obvious open problem certainly is to close the gaps between the upper bounds of Theorems 1, 2, and 3 and the lower bounds of Theorem 4. It would also be interesting to see how well other optimization problems can be approximated in a distributed manner. In particular, the distributed complexity of more general classes of linear programs remains completely open. A very intriguing unsolved problem is to determine to what extent randomization is needed to obtain time-efficient distributed algorithms. Currently, the best determinic algorithms for finding a dominating set of reasonable size and for many other problems take time $2^{O(\sqrt{\log n})}$ whereas the time complexity of the best randomized algorithms usually is at most polylogarithmic in the number of nodes.

Cross-References

- Fractional Packing and Covering Problems
- Maximum Matching
- Randomized Rounding

Recommended Reading

1. Bartal Y, Byers JW, Raz D (1997) Global optimization using local information with applications to flow control. In: Proceedings of the 38th IEEE symposium on the foundations of computer science (FOCS), pp 303–312
2. Kuhn F, Moscibroda T, Wattenhofer R (2004) What cannot be computed locally! In: Proceedings of the 23rd ACM symposium on principles of distributed computing (PODC), pp 300–309
3. Kuhn F, Moscibroda T, Nieberg T, Wattenhofer R (2005) Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In: Proceedings of the 19th international conference on distributed computing (DISC), pp 273–287
4. Kuhn F, Moscibroda T, Nieberg T, Wattenhofer R (2005) Local approximation schemes for ad hoc and sensor networks. In: Proceedings of the 3rd joint workshop on foundations of mobile computing (DIALM-POMC), pp 97–103
5. Kuhn F, Moscibroda T, Wattenhofer R (2005) On the locality of bounded growth. In: Proceedings of the 24th ACM symposium on principles of distributed computing (PODC), pp 60–68
6. Kuhn F, Wattenhofer R (2005) Constant-time distributed dominating set approximation. Distrib Comput 17(4):303–310
7. Kuhn F, Moscibroda T, Wattenhofer R (2006) The price of being near-sighted. In: Proceedings of the 17th ACM-SIAM symposium on discrete algorithms (SODA), pp 980–989
8. Linial N (1992) Locality in distributed graph algorithms. SIAM J Comput 21(1):193–201
9. Naor M, Stockmeyer L (1993) What can be computed locally? In: Proceedings of the 25th annual ACM symposium on theory of computing (STOC), pp 184–193
10. Papadimitriou C, Yannakakis M (1993) Linear programming without the matrix. In: Proceedings of the 25th ACM symposium on theory of computing (STOC), pp 121–129
11. Peleg D (2000) Distributed computing: a locality-sensitive approach. SIAM

Local Computation in Unstructured Radio Networks

Thomas Moscibroda
Systems and Networking Research Group, Microsoft Research, Redmond, WA, USA

Keywords

Coloring unstructured radio networks; Maximal independent sets in radio networks

Years and Authors of Summarized Original Work

2005; Moscibroda, Wattenhofer

Problem Definition

In many ways, familiar distributed computing communication models such as the *message passing model* do not describe the harsh conditions faced in wireless ad hoc and sensor networks closely enough. Ad hoc and sensor networks are multi-hop radio networks and hence, messages being transmitted may interfere with concurrent transmissions leading to collisions and packet losses. Furthermore, the fact that all nodes share the same wireless communication medium leads to an inherent broadcast nature of communication. A message sent by a node can be received by all nodes in its transmission range. These aspects of communication are modeled by the *radio network model*, e.g., [2].

Definition 1 (Radio Network Model) In the radio network model, the wireless network is modeled as a graph $G = (V, E)$. In every time-slot, a node $u \in V$ can either send or not send a message. A node v, $(u, v) \in E$, receives the message if and only *exactly one* of its neighbors has sent a message in this time-slot.

While communication primitives such as broadcast, wake-up, or gossiping, have been widely studied in the literature on radio networks (e.g., [1, 2, 8]), less is known about the computation of *local network coordination structures* such as clusterings or colorings. The most basic notion of a clustering in wireless networks boils down to the graph-theoretic notion of a dominating set.

Definition 2 (Minimum Dominating Set (MDS)) Given a graph $G = (V, E)$. A dominating set is a subset $S \subseteq V$ such that every node is either in S or has at least one neighbor in S. The minimum dominating set problem asks for a dominating set S of minimum cardinality.

A dominating set $S \subseteq V$ in which no two neighboring nodes are in S is a *maximal independent set (MIS)*. The distributed complexity of computing a MIS in the message passing model has been of fundamental interest to the distributed computing community for over two decades (e.g., [11–13]), but much less is known about the problem's complexity in radio network models.

Definition 3 (Maximal Independent Set (MIS)) Given a graph $G = (V, E)$. An independent set is a subset of pair-wise non-adjacent nodes in G. A maximal independent set in G is an independent set $S \subseteq V$ such that for every node $u \notin S$, there is a node $v \in \Gamma(u)$ in S.

Another important primitive in wireless networks is the *vertex coloring* problem, because associating different colors with different time slots in a time-division multiple access (TDMA) scheme; a correct coloring corresponds to a medium access control (MAC) layer without *direct interference*, that is, no two neighboring nodes send at the same time.

Definition 4 (Minimum Vertex Coloring) Given a graph $G = (V, E)$. A correct vertex coloring for G is an assignment of a color $c(v)$ to each node $v \in V$, such that $c(u) \neq c(v)$ any two adjacent nodes $(u, v) \in E$. A minimum vertex coloring is a correct coloring that minimizes the number of used colors.

In order to capture the especially harsh characteristics of wireless multi-hop networks immediately after their deployment, the unstructured radio network model makes additional assumptions. In particular, a new notion of asynchronous wake-up is considered, because, in a wireless, multi-hop environment, it is realistic to assume that some nodes join the network (e.g., become deployed, or switched on) later than others. Notice that this is different from the notion of asynchronous wake-up defined and studied in [8] and subsequent work, in which nodes are assumed to be "woken up" by incoming messages.

Definition 5 (Unstructured Radio Network Model) In the *unstructured radio network model*, the wireless network is modeled as a unit disk graph (UDG) $G = (V, E)$. In every time-slot, a node $u \in V$ can either send or not send a message. A node v, $(u, v) \in E$, receives the message if and only *exactly one* of its neighbors has sent a message in this time-slot. Additionally, the following assumptions are made:

- *Asynchronous wake-up:* New nodes can wake up/join in *asynchronously* at any time. Before waking-up, nodes do neither receive nor send any messages.
- *No global clock:* Nodes only have access to a local clock that starts increasing after wake-up.
- *No collision detection:* Nodes cannot distinguish between the event of a collision and no message being sent. Moreover, a sending node does not know how many (if any at all!) neighbors have received its transmission correctly.
- *Minimal global knowledge:* At the time of their wake-up, nodes have no information about their neighbors in the network and they do not whether some neighbors are already awake, executing the algorithm. However, nodes know an upper bound for the maximum number of nodes $n = |V|$.

The measure that captures the efficiency of an algorithm defined in the unstructured radio network model is its *time-complexity*. Since every node can wake up at a different time, the time-complexity of an algorithm is defined as the maximum number of time-slots between a node's wake-up and its final, irrevocable decision.

Definition 6 (Time Complexity) The *running time* T_v of a node $v \in V$ is defined as the number of time slots between v's *waking up* and the time v makes an irrevocable *final decision* on the outcome of its protocol (e.g., whether or not it joins the dominating set in a clustering algorithm, or which color to take in a coloring algorithm, etc.). The *time complexity* $T(\mathcal{Q})$ of algorithm $\mathcal{Q}$ is defined as the maximum running time over all nodes in the network, i.e., $T(\mathcal{Q}) := \max_{v \in V} T_v$.

Key Results

Naturally, algorithms for such uninitialized, chaotic networks have a different flavor compared to "traditional" algorithms that operate on a given network graph that is static and well-known to all nodes. Hence, the algorithmic difficulty of the following algorithms partly stems from the fact that since nodes wake up asynchronously and do not have access to a global clock, the different phases of the algorithm may be arbitrarily intertwined or shifted in time. Hence, while some nodes may already be in an advanced stage of the algorithm, there may be nodes that have either just woken up, or that are still in early stage. It was proven in [9] that even in single-hop networks (G is the complete graph), no efficient algorithms exist if nodes have no knowledge on n.

Theorem 1 *If nodes have no knowledge of n, every (possibly randomized) algorithm requires*

up to $\Omega(n/\log n)$ time slots before at least one node can send a message in single-hop networks.

In single-hop networks, and if n is globally known, [8] presented a randomized algorithm that selects a unique leader in time $O(n \log n)$, with high probability. This result has subsequently been improved to $O(\log^2 n)$ by Jurdziński and Stachowiak [9]. The generalized wake-up problem in multi-hop radio network was first studied in [4].

The complexity of local network structures such as clusterings or colorings in unstructured multi-hop radio networks was first studied in [10]: A good approximation to the minimum dominating set problem can be computed in polylogarithmic time.

Theorem 2 *In the unstructured radio network model, an expected O(1)-approximation to the dominating set problem can be computed in expected time $O(\log^2 n)$. That is, every node decides whether to join the dominating set within $O(\log^2 n)$ time slots after its wake-up.*

In a subsequent paper [18], it has been shown that the running time of $O(\log^2 n)$ is sufficient even for computing the more sophisticated MIS structure. This result is asymptotically optimal because – improving on a previously known bound of $\Omega(\log^2 n/\log\log n)$ [9] –, a corresponding lower bound of $\Omega(\log^2 n)$ has been proven in [6].

Theorem 3 *With high probability, a maximal independent set (MIS) can be computed in expected time $O(\log^2 n)$ in the unstructured radio network model. This is asymptotically optimal.*

It is interesting to compare this achievable upper bound on the harsh unstructured radio network model with the best known time lower bounds in message passing models: $\Omega(\log^* n)$ in unit disk graphs [12] and $\Omega(\sqrt{\log n/\log\log n})$ in general graphs [11]. Also, a time bound of $O(\log^2 n)$ was also proven in [7] in a radio network model without asynchronous wake-up and in which nodes have a-priori knowledge about their neighborhood.

Finally, it is also possible to efficiently color the nodes of a network as shown in [17], and subsequently improved and generalized in Chap. 12 of [15].

Theorem 4 *In the unstructured radio network model, a correct coloring with at most $O(\Delta)$ colors can be computed in time $O(\Delta \log n)$ with high probability.*

Similar bounds for a model with collision detection mechanisms are proven in [3].

Applications

In wireless ad hoc and sensor networks, local network coordination structures find important applications. In particular, clusterings and colorings can help in facilitating the communication between adjacent nodes (MAC layer protocols) and between distant nodes (routing protocols), or to improve the energy efficiency of the network.

The following mentions two specific examples of applications: Based on the MIS algorithms of Theorem 3, a protocol is presented in [5], which efficiently constructs a *spanner*, i.e., a more sophisticated initial infrastructure that helps in structuring wireless multi-hop network. In [16], the same MIS algorithm is used as an ingredient for a protocol that minimizes the energy consumption of wireless sensor nodes during the *deployment phase*, a problem that has been first studied in [14].

Recommended Reading

1. Alon N, Bar-Noy A, Linial N, Peleg D (1991) A Lower Bound for Radio Broadcast. J Comput Syst Sci 43:290–298
2. Bar-Yehuda R, Goldreich O, Itai A (1987) On the time-complexity of broadcast in radio networks: an exponential gap between determinism randomization. In: Proceedings of the 6th symposium on principles of distributed computing (PODC), pp 98–108

3. Busch R, Magdon-Ismail M, Sivrikaya F, Yener B (2004) Contention-free MAC protocols for wireless sensor networks. In: Proceedings of the 18th annual conference on distributed computing (DISC)
4. Chrobak M, Gąsieniec L, Kowalski, D (2004) The wake-up problem in multi-hop radio networks. In: Proceedings of the 15th ACM-SIAM symposium on discrete algorithms (SODA), pp 992–1000
5. Farach-Colton M, Fernandes RJ, Mosteiro MA (2005) Bootstrapping a hop-optimal network in the weak sensor model. In: Proceedings of the 13th European symposium on algorithms (ESA), pp 827–838
6. Farach-Colton M, Fernandes RJ, Mosteiro MA (2006) Lower bounds for clear transmissions in radio networks. In: Proceedings of the 7th Latin American symposium on theoretical informatics (LATIN), pp 447–454
7. Gandhi R, Parthasarathy S (2004) Distributed algorithms for coloring and connected domination in wireless ad hoc networks. In: Foundations of software technology and theoretical computer science (FSTTCS), pp 447–459
8. Gąsieniec L, Pelc A, Peleg D (2000) The wakeup problem in synchronous broadcast systems (extended abstract). In: Proceedings of the 19th ACM symposium on principles of distributed computing (PODC), pp 113–121
9. Jurdziński T, Stachowiak G (2002) Probabilistic algorithms for the wakeup problem in single-hop radio networks. In: Proceedings of the 13th Annual International Symposium on Algorithms and Computation (ISAAC), pp 535–549
10. Kuhn F, Moscibroda T, Wattenhofer R (2004) Initializing newly deployed ad hoc and sensor networks. In: Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MOBICOM), pp 260–274
11. Kuhn F, Moscibroda T, Wattenhofer R (2004) What cannot be computed locally! In: Proceedings of 23rd annual symposium on principles of distributed computing (PODC), pp 300–309
12. Linial N (1992) Locality in distributed graph algorithms. SIAM J Comput 21(1):193–201
13. Luby M (1986) A simple parallel algorithm for the maximal independent set problem. SIAM J Comput 15:1036–1053
14. McGlynn MJ, Borbash SA (2001) Birthday protocols for low energy deployment and flexible neighborhood discovery in ad hoc wireless networks. In: Proceedings of the 2nd ACM international symposium on mobile ad hoc networking & computing (MOBIHOC)
15. Moscibroda T (2006) Locality, scheduling, and selfishness: algorithmic foundations of highly decentralized networks. Doctoral thesis Nr. 16740, ETH Zurich
16. Moscibroda T, von Rickenbach P, Wattenhofer R (2006) Analyzing the energy-latency trade-off during the deployment of sensor networks. In: Proceedings of the 25th joint conference of the IEEE computer and communications societies (INFOCOM)
17. Moscibroda T, Wattenhofer R (2005) Coloring unstructured radio networks. In: Proceedings of the 17th ACM symposium on parallel algorithms and architectures (SPAA), pp 39–48
18. Moscibroda T, Wattenhofer R (2005) Maximal independent sets in radio networks. In: Proceedings of the 23rd ACM symposium on principles of distributed computing (PODC), pp 148–157

Local Reconstruction

Comandur Seshadhri
Sandia National Laboratories, Livermore, CA, USA
Department of Computer Science, University of California, Santa Cruz, CA, USA

Keywords

Data reconstruction; Property testing; Sublinear algorithms

Years and Authors of Summarized Original Work

2010; Saks, Seshadhri

Problem Definition

Consider some massive dataset represented as a function $f : D \mapsto R$, where D is discrete and R is an arbitrary range. This dataset could be as varied as an array of numbers, a graph, a matrix, or a high-dimensional function. Datasets are often useful because they possess some property of interest. An array might be sorted, a graph might be connected, a matrix might be orthogonal, or a function might be convex. These properties are critical to the use of the dataset. Yet, due to unavoidable errors (say, in storing the dataset), these properties might not hold any longer. For example, a sorted array could become unsorted because of roundoff errors.

Can we find a function $g : D \mapsto R$ that satisfies the property and is "sufficiently close" to f? Let us formalize this question. Let $\mathcal{P}$ denote a property, which we define as a subset of functions. We define a *distance* between functions, $\mathsf{dist}(f, g) = |\{x | f(x) \neq g(x)\}|/|D|$. In words, this is the fraction of domain points where f and g differ (the relative Hamming distance). This definition naturally extends to properties: $\mathsf{dist}(f, \mathcal{P}) = \min_{h \in \mathcal{P}} \mathsf{dist}(f, h)$. This is the minimum amount of change f must undergo to have property $\mathcal{P}$. Our aim is to construct $g \in \mathcal{P}$ such that $\mathsf{dist}(f, g)$ is "small." The latter can be quantified by comparing with the baseline, $\mathsf{dist}(f, \mathcal{P})$.

The *offline reconstruction* problem involves explicitly constructing g from f. But this is prohibitively expensive if f is a large dataset. Instead, we wish to *locally* construct g, meaning we want to quickly compute $g(x)$ (for $x \in D$) without constructing g completely.

Local filters [13]: A *local filter* for property $\mathcal{P}$ is an algorithm A satisfying the following conditions. The filter has *oracle access* to function $f : D \mapsto R$, meaning that it can access $f(x)$ for any $x \in D$. Each such access is called a *lookup*. The filter takes as input an auxiliary random seed ρ and $x \in D$. For fixed f, ρ, A runs deterministically on input x to produce an output $A_{f,\rho}(x)$. Note that $A_{f,\rho}$ specifies a function on domain D, which will be the desired function g.

1. For each f and ρ, $A_{f,\rho}$ always satisfies $\mathcal{P}$.
2. For each f, with high probability over ρ, the function $A_{f,\rho}$ is suitably close to f.
3. For each x, $A_{f,\rho}(x)$ can be computed with very few lookups.
4. The size of the random seed ρ should be much smaller than D.

Let g be $A_{f,\rho}$. Condition 2 has been formalized in at least two different ways. The original definition demanded that $\mathsf{dist}(f, g) \leq c \cdot \mathsf{dist}(f, \mathcal{P})$, where c is a fixed constant [13]. Other results only enforce Condition 2 when $f \in \mathcal{P}$ [3, 9]. One could imagine desiring $|\mathsf{dist}(f, g) - \mathsf{dist}(f, \mathcal{P})| < \delta$, for input parameter δ. Conditions 3 and 4 typically demand that the lookup complexity and random seed lengths are $o(|D|)$ (sometimes we desire them to be $\mathrm{poly}(\log |D|)$ or even constant).

Connections with Other Models

The notion of reconstruction through filters was first proposed by Ailon et al. in [1], though the requirements were less stringent. There is a sequence $x_1, x_2, \ldots$ of domain points generated online. Given x_i, the filter outputs value $g(x_i)$. The filter is allowed to store previous outputs to ensure consistency. Saks and Seshadhri [13] defined local filters to address two concerns with this model. First, the storage of all previous queries and answers is a massive space overhead. Second, different runs of the filter construct different g's (because the filter is randomized). So we cannot instantiate multiple copies of the filter to handle queries independently and consistently.

Independent of this line of work, Brakerski defined *local restorers* [6], which are basically equivalent to filters with an appropriate setting of Conditions 1 and 2. A major generalization of local filters, called *local computation algorithms*, was subsequently proposed by Rubinfeld et al. [12]. This model considers computation on a large input where the output itself is large (e.g., one may want a maximal independent set of a massive graph). The aim is to locally compute the output, by an algorithm akin to a filter.

Depending on how Property 2 is instantiated, filters can easily be used for tolerant testing and distance approximation [11]. If the filter ensures that $\mathsf{dist}(f, g)$ is comparable to $\mathsf{dist}(f, \mathcal{P})$, then it suffices to estimate $\mathsf{dist}(f, g)$ for distance approximation.

A special case of local reconstruction that has received extensive attention is decoding of error correcting codes. Here, f is some string, and $\mathcal{P}$ is the set of all valid code words. In local decoding, there is either *one* correct output or a sparse list of possible correct outputs. For general properties, there may be many (possibly infinitely many) ways to construct a valid reconstruction g. This creates challenges for designing filters. Once the random seed is fixed, all query answers

provided by the filter must be consistent with a *single* function having the property.

Key Results

Over the past decade, there have been numerous results on local reconstruction, over a variety of domains.

Monotonicity

The most studied property for local reconstruction is monotonicity [1, 3, 5, 13]. Consider $f : [n]^d \mapsto \mathbb{R}$, where d, n are positive integers and $[n] = \{1, 2, \ldots, n\}$. The domain is equipped with the natural coordinate-wise partial order. Namely, $x \prec y$ if $x \neq y$ and all coordinates of x are less than or equal those of y. A function f is monotone if: $\forall x \prec y$, $f(x) \leq f(y)$. When $d = 1$, monotone functions are exactly sorted arrays.

Most initial work on local filters focused exclusively on monotonicity. There exists a filter of running time $(\log n)^{O(d)}$ with $\mathsf{dist}(f, g) \leq 2^{O(d)}\mathsf{dist}(f, \mathcal{P})$ [13]. There are nearly matching lower bounds that are extremely strong; even for relaxed versions of Condition 2 [3].

Lipschitz Continuity

Let n, d be positive integers and c be a positive real number. A function $f : [n]^d \mapsto \mathbb{R}$ is c-Lipschitz if $\forall x, y, |f(x) - f(y)| \leq c\|x - y\|_1$. This is a fundamental property of functions and appears in functional analysis, statistics, and optimization. Recently, Lipschitz continuity was studied from a property testing perspective by Jha and Raskhodnikova [9]. Most relevant to this essay, they gave an application of local Lipschitz filters for differential privacy. The guarantees on their filter are analogous to monotonicity (with a weaker form of Property 2). Awasthi, Jha, Molinaro, and Raskhodnikova [3] gave matching lower bounds for these reconstruction problems.

Dense Graph Properties

Dense graphs are commonly studied in property testing, where the input is given as a binary adjacency matrix. Brakerski's work on local restorers provides filters for bipartiteness and existence of large cliques. Large classes of dense graphs are known to be tolerant testable. These results have been extended to local filters for hypergraph properties by Austin and Tao [2]. This work was developed independently of all the previous work on filters, and their algorithms are called "repair" algorithms.

Connectivity Properties of Sparse Graphs

In the sparse graph setting, the input G is an adjacency list of a bounded-degree graph. Filters have been given for several properties regarding connectivity. Campagna, Guo, and Rubinfeld [7] provide reconstructors for k-connectivity and the property of having low diameter. Local reconstructors for the property of expansion were given by Kale and Seshadhri [10].

Convexity in 2, 3-Dimensions

Chazelle and Seshadhri [8] studied reconstruction in the geometric setting. They focus on convex polygon and 3D convex polytope reconstruction. These results were in the online filter setting of [1], though their 3D result is a local filter.

Open Problems

For any property tester, one can ask if the associated property has a local filter. Given the breadth of this area, we cannot hope to give a good summary of open problems. Nonetheless, we make a few suggestions.

The Curse of Dimensionality for Monotonicity and Lipschitz

Much work has gone into understanding local filters for monotonicity, but it is not clear how to remove the exponential dependence on d. Can we find a reasonable setting for filters where a $\mathrm{poly}(d, \log n)$ lookup complexity is possible? One possibility is requiring only "additive error" in Condition 2. For some parameter $\delta > 0$, we only want $|\mathsf{dist}(f, g) - \mathsf{dist}(f, \mathcal{P})| \leq \delta$. Is there a filter with lookup complexity $\mathrm{poly}(d, \log n, 1/\delta)$? This definition would avoid previous lower bounds [3].

Filters for Convexity

Filters for convexity could have a great impact on optimization. A local filter would implicitly represent a close enough convex function to an input non-convex function, which would be extremely useful for (say) minimization. For this application, it would be essential to handle high-dimensional functions. Unfortunately, there are no known property testers for convexity in this setting, so designing local filters is a distant goal.

Filters for Properties of Bounded-Degree Graphs

The large class of minor-closed properties (such as planarity) is known to be testable for bounded-degree graphs [4]. Can we get local filters for these properties? This appears to be a challenging question, since even approximating the distance to planarity is a difficult problem. Nonetheless, the right relaxations of filter conditions could lead to positive results for filters.

Cross-References

- ▸ Monotonicity Testing
- ▸ Testing if an Array Is Sorted

Recommended Reading

1. Ailon N, Chazelle B, Comandur S, Liu D (2008) Property-preserving data reconstruction. Algorithmica 51(2):160–182
2. Austin T, Tao T (2010) Testability and repair of hereditary hypergraph properties. Random Struct Algorithms 36(4):373–463
3. Awasthi P, Jha M, Molinaro M, Raskhodnikova S (2012) Limitations of local filters of lipschitz and monotone functions. In: Workshop on approximation, randomization, and combinatorial optimization. Algorithms and techniques (RANDOM-APPROX), pp 374–386
4. Benjamini I, Schramm O, Shapira A (2010) Every minor-closed property of sparse graphs is testable. Adv Math 223(6):2200–2218
5. Bhattacharyya A, Grigorescu E, Jha M, Jung K, Raskhodnikova S, Woodruff D (2012) Lower bounds for local monotonicity reconstruction from transitive-closure spanners. SIAM J Discret Math 26(2):618–646
6. Brakerski Z (2008) Local property restoring. Manuscript
7. Campagna A, Guo A, Rubinfeld R (2013) Local reconstructors and tolerant testers for connectivity and diameter. In: Workshop on approximation, randomization, and combinatorial optimization. Algorithms and techniques (RANDOM-APPROX), pp 411–424
8. Chazelle B, Seshadhri C (2011) Online geometric reconstruction. J ACM 58(4):14
9. Jha M, Raskhodnikova S (2013) Testing and reconstruction of lipschitz functions with applications to data privacy. SIAM J Comput 42(2):700–731
10. Kale S, Seshadhri C (2011) Testing expansion in bounded degree graphs. SIAM J Comput 40(3):709–720
11. Parnas M, Ron D, Rubinfeld R (2006) Tolerant property testing and distance approximation. J Comput Syst Sci 6(72):1012–1042
12. Rubinfeld R, Tamir G, Vardi S, Xie N (2011) Fast local computation algorithms. In: Proceedings of innovations in computer science, pp 223–238
13. Saks M, Seshadhri C (2006) Local monotonicity reconstruction. SIAM J Comput 39(7):2897–2926

Local Search for *K*-medians and Facility Location

Kamesh Munagala
Levine Science Research Center, Duke University, Durham, NC, USA

Keywords

Clustering; Facility location; *k*-Means; *k*-Medians; *k*-Medioids; Point location; Warehouse location

Years and Authors of Summarized Original Work

2001; Arya, Garg, Khandekar, Meyerson, Munagala, Pandit

Problem Definition

Clustering is a form of *unsupervised learning*, where the goal is to "learn" useful patterns in a data set $\mathcal{D}$ of size n. It can also be thought of

as a data compression scheme where a large data set is represented using a smaller collection of "representatives". Such a scheme is characterized by specifying the following:

1. A *distance* metric $\mathbf{d}$ between items in the data set. This metric should satisfy the triangle inequality: $\mathbf{d}(i,j) \leq \mathbf{d}(j,k) + \mathbf{d}(k,i)$ for any three items $i,j,k \in \mathcal{D}$. In addition, $\mathbf{d}(i,j) = \mathbf{d}(j,i)$ for all $i,j \in S$ and $\mathbf{d}(i,i) = 0$. Intuitively, if the distance between two items is smaller, they are more similar. The items are usually points in some high dimensional Euclidean space $\mathcal{R}^d$. The commonly used distance metrics include the Euclidean and Hamming metrics, and the cosine metric measuring the angle between the vectors representing the items.
2. The output of the clustering process is a partitioning of the data. This chapter deals with *center-based* clustering. Here, the output is a smaller set $C \subset \mathcal{R}^d$ of *centers* which best represents the input data set $S \subset \mathcal{R}^d$. It is typically the case that $|C| \ll |\mathcal{D}|$. Each item $j \in \mathcal{D}$ is *mapped to* or *approximated by* the the closest center $i \in C$, implying $\mathbf{d}(i,j) \leq \mathbf{d}(i',j)$ for all $i' \in C$. Let $\sigma : \mathcal{D} \to C$ denote this mapping. This is intuitive since closer-by (similar) items will be mapped to the same center.
3. A measure of the *quality* of the clustering, which depends on the desired output. There are several commonly used measures for the quality of clustering. In each of the clustering measures described below, the goal is to choose C such that $|C| = k$ and the objective function $f(C)$ is minimized.

   ***k*-center:** $f(C) = \max_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j))$.
   ***k*-median:** $f(C) = \sum_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j))$.
   ***k*-means:** $f(C) = \sum_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j))^2$.

All the objectives described above are NP-HARD to optimize in general metric spaces $\mathbf{d}$, leading to the study of heuristic and approximation algorithms. In the rest of this chapter, the focus is on the k-median objective. The approximation algorithms for k-median clustering are designed for $\mathbf{d}$ being a general possibly non-Euclidean metric space. In addition, a collection $\mathcal{F}$ of possible center locations is given as input, and the set of centers C is restricted to $C \subseteq \mathcal{F}$. From the perspective of approximation, the restriction of the centers to a finite set $\mathcal{F}$ is not too restrictive – for instance, the optimal solution which is restricted to $\mathcal{F} = \mathcal{D}$ has objective value at most a factor 2 of the optimal solution which is allowed arbitrary $\mathcal{F}$. Denote $|\mathcal{D}| = n$, and $|\mathcal{F}| = m$. The running times of the heuristics designed will be polynomial in m n, and a parameter $\varepsilon > 0$. The metric space $\mathbf{d}$ is now defined over $\mathcal{D} \cup \mathcal{F}$.

A related problem to k-medians is its Lagrangean relaxation, called FACILITY LOCATION. In this problem, there is a again collection $\mathcal{F}$ of possible center locations. Each location $i \in \mathcal{F}$ has a location cost r_i. The goal is to choose a collection $C \subseteq \mathcal{F}$ of centers and construct the mapping $\sigma : S \to C$ from the items to the centers such that the following function is minimized:

$$f(C) = \sum_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j)) + \sum_{i \in C} r_i \ .$$

The facility location problem effectively gets rid of the hard bound k on the number of centers in k-medians, and replaces it with the center cost term $\sum_{i \in C} r_i$ in the objective function, thereby making it a Lagrangean relaxation of the k-median problem. Note that the costs of centers can now be non-uniform.

The approximation results for both the k-median and facility location problems carry over as is to the weighted case: Each item $j \in \mathcal{D}$ is allowed to have a non-negative weight w_j. In the objective function $f(C)$, the term $\sum_{j \in \mathcal{D}} \mathbf{d}(j, \sigma(j))$ is replaced with $\sum_{j \in \mathcal{D}} w_j \cdot \mathbf{d}(j, \sigma(j))$. The weighted case is especially relevant to the FACILITY LOCATION problem where the item weights signify user demands for a resource, and the centers denote locations of the resource. In the remaining discussion, "items" and "users" are used interchangably to denote members of the set $\mathcal{D}$.

Key Results

The method of choice for solving both the k-median and FACILITY LOCATION problems are the class of local search heuristics, which run in "local improvement" steps. At each step t, the heuristic maintains a set C_t of centers. For the k-median problem, this collection satisfies $|C_t| = k$. A local improvement step first generates a collection of new solutions $\mathcal{E}_{t+1}$ from C_t. This is done such that $|\mathcal{E}_{t+1}|$ is polynomial in the input size. For the k-median problem, in addition, each $C \in \mathcal{E}_{t+1}$ satisfies $|C| = k$. The improvement step sets $C_{t+1} = \text{argmin}_{C \in \mathcal{E}_{t+1}} f(C)$. For a pre-specified parameter $\varepsilon > 0$, the improvement iterations stop at the first step T where $f(C_T) \geq (1-\varepsilon) f(C_{T-1})$.

The key design issue is the specification of the start set C_0, and the construction of $\mathcal{E}_{t+1}$ from C_t. The key analysis issues are bounding the number of steps T till termination, and the quality of the final solution $f(C_T)$ against the optimal solution $f(C^*)$. The ratio $(f(C_T))/(f(C^*))$ is termed the "locality gap" of the heuristic.

Since each improvement step reduces the value of the solution by at least a factor of $(1-\varepsilon)$, the running time in terms of number of improvement steps is given by the following expression (here D is the ratio of the largest to smallest distance in the metric space over $\mathcal{D} \cup \mathcal{F}$).

$$T \leq \log_{1/(1-\varepsilon)} \left(\frac{f(C_0)}{f(C_T)} \right) \leq \frac{\log \left(\frac{f(C_0)}{f(C_T)} \right)}{\varepsilon} \leq \frac{\log(nD)}{\varepsilon}$$

which is polynomial in the input size. Each improvement step needs computation of $f(C)$ for $C \in \mathcal{E}_t$. This is polynomial in the input size since $|\mathcal{E}_t|$ is assumed to be polynomial.

k-Medians

The first local search heuristic with provable performance guarantees is presented in the work of Arya et al. [1]. The is the natural p-swap heuristic: Given the current center set C_t of size k, the set $\mathcal{E}_{t+1}$ is defined by:

$$\mathcal{E}_{t+1} = \{(C_t \setminus \mathcal{A}) \cup \mathcal{B}\,, \text{ where } \mathcal{A} \subseteq C_t, \mathcal{B} \subseteq \mathcal{F} \setminus C_t, |\mathcal{A}| = |\mathcal{B}| \leq p\}.$$

The above simply means swap at most p centers from C_t with the same number of centers from $\mathcal{F} \setminus C_t$. Recall that $|\mathcal{D}| = n$ and $|\mathcal{F}| = m$. Clearly, $|\mathcal{E}_{t+1}| \leq (k(m-k))^p \leq (km)^p$. The start set C_0 is chosen arbitrarily. The value p is a parameter which affects the running time and the approximation ratio. It is chosen to be a constant, so that $|\mathcal{E}_t|$ is polynomial in m.

Theorem 1 ([1]) *The p-swap heuristic achieves locality gap* $(3 + 2/p) + \varepsilon$ *in running time* $O(nk(\log(nD))/\varepsilon(mk)^p)$. *Furthermore, for every p there is a k-median instance where the p-swap heuristic has locality gap exactly* $(3 + 2/p)$.

Setting $p = 1/\varepsilon$, the above heuristic achieves a $3 + \varepsilon$ approximation in running time $\tilde{O}(n(mk)^{O(1/\varepsilon)})$.

Facility Location

For this problem, since there is no longer a constraint on the number of centers, the local improvement step needs to be suitably modified. There are two local search heuristics both of which yield a locality gap of $3 + \varepsilon$ in polynomial time.

The "add/delete/swap" heuristic proposed by Kuehn and Hamburger [10] either adds a center to C_t, drops a center from C_t, or swaps a center in C_t with one in $\mathcal{F} \setminus C_t$. The start set C_0 is again arbitrary.

$$\mathcal{E}_{t+1} = \{(C_t \setminus \mathcal{A}) \cup \mathcal{B}, \text{ where } \mathcal{A} \subseteq C_t, \mathcal{B} \subseteq \mathcal{F} \setminus C_t, |\mathcal{A}| = 0, |\mathcal{B}| = 1 \text{ or } |\mathcal{A}| = 1, |\mathcal{B}| = 0, \text{ or } |\mathcal{A}| = 1, |\mathcal{B}| = 1\}$$

Clearly, $|\mathcal{E}_{t+1}| = O(m^2)$, making the running time polynomial in the input size and $1/\varepsilon$. Korupolu, Plaxton, and Rajaraman [9] show that this heuristic achieves a locality gap of at most $5 + \varepsilon$. Arya et al. [1] strengthen this analysis to show that this heuristic achieves a locality gap of $3 + \varepsilon$, and that bound this is tight in the sense

that there are instances where the locality gap is exactly 3.

The "add one/delete many" heuristic proposed by Charikar and Guha [2] is slightly more involved. This heuristic adds one facility and drops all facilities which become irrelevant in the new solution.

$$\mathcal{I}_{t+1} = \{(C_t \cup \{i\}) \setminus I(i), \text{ where } i \in \mathcal{F} \setminus C_t, I(i) \subseteq C_t\}$$

The set $I(i)$ is computed as follows: Let W denote the set of items closer to i than to their assigned centers in C_t. These items are ignored from the computation of $I(i)$. For every center $s \in C_t$, let U_s denote all items which are assigned to s. If $f_s + \sum_{j \in U_s \setminus W} d_j \mathbf{d}(j,s) > \sum_{j \in U_s \setminus W} d_j \mathbf{d}(j,i)$, then it is cheaper to remove location s and reassign the items in $U_s \setminus W$ to i. In this case, s is placed in $I(i)$. Let N denote $m + n$. Computing $I(i)$ is therefore a $O(N)$ time greedy procedure, making the overall running time polynomial. Charikar and Guha [2] show the following theorem:

Theorem 2 ([2]) *The local search heuristic which attempts to add a random center $i \notin C_t$ and remove set I(i), computes a $3 + \varepsilon$ approximation with high probability within $T = O(N \log N(\log N + 1/\varepsilon))$ improvement steps, each with running time O(N).*

Capacitated Variants

Local search heuristics are also known for capacitated variants of the *k*-median and facility location problems. In this variant, each possible location $i \in \mathcal{F}$ can serve at most u_i number of users. In the soft capacitated variant of facility location, some $r_i \geq 0$ copies can be opened at $i \in \mathcal{F}$ so that the facility cost is $f_i r_i$ and the number of users served is at most $r_i u_i$. The optimization goal is now to decide the value of r_i for each $i \in \mathcal{F}$ so that the assignment of users to the centers satisfies the capacity constraints at each center, and the cost of opening the centers and assigning the users is minimized. For this variant, Arya et al. [1] show a local search heuristic with a locality gap of $4 + \varepsilon$.

In the version of facility location with hard capacities, location $i \in \mathcal{F}$ has a hard bound u_i on the number of users that can be assigned here. If all the capacities u_i are equal (uniform case), Korupolu, Plaxton, and Rajaraman [9] present an elegant local search heuristic based on solving a transshipment problem which achieves a $8 + \varepsilon$ locality gap. The analysis is improved by Chudak and Williamson [4] to show a locality gap $6 + \varepsilon$. The case of non-uniform capacities requires significantly new ideas – Pál, Tardos, and Wexler [14] present a network flow based local search heuristic that achieves a locality gap of $9 + \varepsilon$. This bound is improved to $8 + \varepsilon$ by Mahdian and Pál [12], who generalize several of the local search techniques described above in order to obtain a constant factor approximation for the variant of facility location where the facility costs are arbitrary non-decreasing functions of the demands they serve.

Related Algorithmic Techniques

Both the *k*-median and facility location problems have a rich history of approximation results. Since the study of uncapacitated facility location was initiated by Cornuejols, Nemhauser, and Wolsey [5], who presented a natural linear programming (LP) relaxation for this problem, several constant-factor approximations have been designed via several techniques, ranging from rounding of the LP solution [11, 15], local search [2, 9], the primal-dual schema [7], and dual fitting [6]. For the *k*-median problem, the first constant factor approximation [3] of $6\frac{2}{3}$ was obtained by rounding the natural LP relaxation via a generalization of the filtering technique in [11]. This result was subsequently improved to a 4 approximation by Lagrangean relaxation and the primal-dual schema [2, 7], and finally to a $(3 + \varepsilon)$ approximation via local search [1].

Applications

The facility location problem has been widely studied in operations research [5, 10], and forms

a fundamental primitive for several resource location problems. The *k*-medians and *k*-means metrics are widely used in clustering, or unsupervised learning. For clustering applications, several heuristic improvements to the basic local search framework have been proposed: *k*-Medioids [8] selects a random input point and replaces it with one of the existing centers if there is an improvement; the CLARA [8] implementation of *k*-Medioids chooses the centers from a random sample of the input points to speed up the computation; the CLARANS [13] heuristic draws a fresh random sample of feasible centers before each improvement step to further improve the efficiency.

Cross-References

▸ Facility Location

Recommended Reading

1. Arya V, Garg N, Khandekar R, Meyerson A, Munagala K, Pandit V (2004) Local search heuristics for k-median and facility location problems. SIAM J Comput 33(3):544–562
2. Charikar M, Guha S (2005) Improved combinatorial algorithms for facility location problems. SIAM J Comput 34(4):803–824
3. Charikar M, Guha S, Tardos É, Shmoys DB (1999) A constant factor approximation algorithm for the k-median problem(extended abstract). In: STOC '99: proceedings of the thirty-first annual ACM symposium on theory of computing, Atlanta, 1–4 May 1999, pp 1–10
4. Chudak FA, Williamson DP (2005) Improved approximation algorithms for capacitated facility location problems. Math Program 102(2):207–222
5. Cornuejols G, Nemhauser GL, Wolsey LA (1990) The uncapacitated facility location problem. In: Discrete location theory. Wiley, New York, pp 119–171
6. Jain K, Mahdian M, Markakis E, Saberi A, Vazirani VV (2003) Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. J ACM 50(6):795–824
7. Jain K, Vazirani VV (2001) Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. J ACM 48(2):274–296
8. Kaufman L, Rousseeuw PJ (1990) Finding groups in data: an introduction to cluster analysis. Wiley, New York
9. Korupolu MR, Plaxton CG, Rajaraman R (1998) Analysis of a local search heuristic for facility location problems. In: SODA '98: proceedings of the ninth annual ACM-SIAM symposium on discrete algorithms, San Francisco, 25–26 Jan 1998, pp 1–10
10. Kuehn AA, Hamburger MJ (1963) A heuristic program for locating warehouses. Manag Sci 9(4):643–666
11. Lin JH, Vitter JS (1992) ε-approximations with minimum packing constraint violation (extended abstract). In: STOC '92: proceedings of the twenty-fourth annual ACM symposium on theory of computing, Victoria, pp 771–782
12. Mahdian M, Pál M (2003) Universal facility location. In: European symposium on algorithms, Budapest, 16–19 Sept 2003, pp 409–421
13. Ng RT, Han J (1994) Efficient and effective clustering methods for spatial data mining. In: Proceedings of the symposium on very large data bases (VLDB), Santiago de Chile, 12–15 Sept 1994, pp 144–155
14. Pál M, Tardos É, Wexler T (2001) Facility location with nonuniform hard capacities. In: Proceedings of the 42nd annual symposium on foundations of computer science, Las Vegas, 14–17 Oct 2001, pp 329–338
15. Shmoys DB, Tardos É, Aardal K (1997) Approximation algorithms for facility location problems. In: Proceedings of the twenty-ninth annual ACM symposium on theory of computing, El Paso, 4–6 May 1997, pp 265–274

Locality in Distributed Graph Algorithms

Pierre Fraigniaud
Laboratoire d'Informatique Algorithmique: Fondements et Applications, CNRS and University Paris Diderot, Paris, France

Keywords

Coloring; Distributed computing; Maximal independent set; Network computing

Years and Authors of Summarized Original Work

1992; Linial
1995; Naor, Stockmeyer
2013; Fraigniaud, Korman, Peleg

Problem Definition

In the context of *distributed network computing*, an important concern is the ability to design *local* algorithms, that is, distributed algorithms in which every node (Each node is a computing entity, which has the ability to exchange messages with its neighbors in the network along its communication links.) of the network can deliver its result after having consulted only nodes in its vicinity. The word "vicinity" has a rather vague interpretation in general. Nevertheless, the objective is commonly to design algorithms in which every node outputs after having exchanged information with nodes at constant distance from it (i.e., at distance independent of the number of nodes n in the networks) or at distance at most polylogarithmic in n, but certainly significantly smaller than n or than the diameter of the network.

The *tasks* to be solved by distributed algorithms acting in networks can be formalized as follows. The network itself is modeled by an undirected connected graph G with node set $V(G)$ and edge set $E(G)$, without loops and double edges. In the sequel, by *graph* we are only referring to this specific type of graphs. Nodes are *labeled* by a function $\ell : V \to \{0,1\}^*$ that assigns to every node v its label $\ell(v)$. A pair (G,ℓ), where G is a graph and ℓ is a labeling of G, is called *configuration*, and a collection $\mathcal{L}$ of configurations is called a *distributed language*. A typical example of a distributed language is: $\mathcal{L}_{\text{properly colored}} = \{(G,\ell) : \ell(v) \neq \ell(v') \text{ for all } \{v,v'\} \in E(G)\}$.

Unless specified otherwise, we are always assuming that the considered languages are decidable in the sense of classical (sequential) computability theory. To every distributed language $\mathcal{L}$ can be associated a *construction* task which consists in computing the appropriate labels for a given network (Here, we are restricting ourselves to *input-free* construction tasks, but the content of this chapter can be generalized to tasks with inputs, in which case the labels are input-output pairs, and, given the inputs, the nodes must produce the appropriate outputs to fit in the considered language.):

Problem 1 (Construction Task for $\mathcal{L}$)

INPUT: A graph G (in which nodes haves no labels);
OUTPUT: A label $\ell(v)$ at each node v, such that $(G,\ell) \in \mathcal{L}$.

For instance, the construction task for $\mathcal{L}_{\text{properly colored}}$ consists, for each node of a graph G, to output a color so that any two adjacent nodes do not output the same color. To every distributed language $\mathcal{L}$ can also be associated a *decision* task, which consists in having nodes deciding whether any given configuration (G,ℓ) is in $\mathcal{L}$ (in this case, every node v is given its label $\ell(v)$ as inputs). This type of tasks finds applications whenever it is desired to check the correctness of a solution produced by another algorithm or, say, by some black box that may act incorrectly. The decision *rule*, motivated by various considerations including *termination detection*, is as follows: if $(G,\ell) \in \mathcal{L}$, then all nodes must *accept* the configuration, while if $(G,\ell) \notin \mathcal{L}$, then at least one node must *reject* that configuration. In other words:

Problem 2 (Decision Task for $\mathcal{L}$)

INPUT: A configuration (G,ℓ) (i.e., each node $v \in V(G)$ has a label $\ell(v)$);
OUTPUT: A boolean $b(v)$ at each node v such that:

$$(G,\ell) \in \mathcal{L} \iff \bigwedge_{v \in V(G)} b(v) = \mathit{true}.$$

For instance, a decision algorithm for $\mathcal{L}_{\text{properly colored}}$ consists, for each node v, of a graph G with input some color $\ell(v)$, to accept if all its neighbors have colors distinct from $\ell(v)$, and to reject otherwise. Finally, the third type of tasks can be associated to distributed languages, called *verification* tasks, which can also be seen as a *nondeterministic* variant of the decision tasks. In the context of verification, in addition to its label $\ell(v)$, every node $v \in V(G)$ is also given a *certificate* $c(v)$. This provides G with a global distributed certificate $c : V(G) \to \{0,1\}^*$

that is supposed to attest to the fact that the labels are correct. If this is indeed the case, i.e., if $(G, \ell) \in \mathcal{L}$, then all nodes must accept the instance (provided with the due certificate). Note that a verification algorithm is allowed to reject a configuration $(G, \ell) \in \mathcal{L}$ in case the certificate is not appropriate for that configuration since for every configuration $(G, \ell) \in \mathcal{L}$, one just asks for the existence of *at least one* appropriate certificate. In addition, to prevent the nodes to be fooled by some certificate on an illegal instance, it is also required that if $(G, \ell) \notin \mathcal{L}$, then for *every* certificate, at least one node must reject that configuration. In other words:

Problem 3 (Verification Task for $\mathcal{L}$)

INPUT: A configuration (G, ℓ), and a distributed certificate c;

OUTPUT: A boolean $b(v, c)$ at each node v, which may indeed depend on c, such that:

$$(G, \ell) \in \mathcal{L} \iff \bigvee_{c' \in \{0,1\}^*} \bigwedge_{v \in V(G)} b(v, c') = \text{true}.$$

For instance, cycle-freeness cannot be locally decided, as even cycles and paths cannot be locally distinguished. However, cycle-freeness can be locally verified, using certificates on $O(\log n)$ bits, as follows. The certificate of node v in a cycle-free graph G is its distance in G to some fixed node $v_0 \in V(G)$. The verification algorithm essentially checks that every node v with $c(v) > 0$ has a unique neighbor v' with $c(v') = c(v) - 1$ and all its other neighbors w with $c(w) = c(v) + 1$, while a node v with $c(v) = 0$ checks that all its neighbors w satisfy $c(w) = 1$. If G has a cycle, then no certificates can pass these tests. As in sequential computability theory, the terminology "verification" comes from the fact that a distributed certificate can be viewed as a (distributed) *proof* that the current configuration is in the language, and the role of the algorithm is to verify this proof. The ability to simultaneously construct a labeling ℓ for G as well as a proof c certifying the correctness of ℓ is a central notion in the design of distributed *self-stabilizing* algorithms – in which variables can be transiently corrupted.

Locality in distributed graph algorithms is dealing with the design and analysis of distributed network algorithms solving any of the above three kinds of tasks.

Computational Model

The study of local algorithms is usually tackled in the framework of the so-called LOCAL model, formalized and thoroughly studied in [13]. In this model, every node v is a Turing machine which is given an *identity*, i.e., a nonnegative integer $\text{id}(v)$. All identities given to the nodes of any given network are pairwise distinct. All nodes execute the same algorithm. They wake up simultaneously, and the computation performs in synchronous *rounds*, where each round consists in three phases executed by each node: (1) send a message to all neighboring nodes in the network, (2) receive the messages sent by the neighboring nodes in the network, and (3) perform some individual computation. The complexity of an algorithm in the LOCAL model is measured in term of *number of rounds* until all nodes terminate. This number of rounds is actually simply the maximum, taken over all nodes in the network, of the distance at which information is propagated from a node in the network. In fact, an algorithm performing in t rounds can be rewritten into an algorithm in which every node, first, collects all data from the nodes at distance at most t from it in G and, second, performs some individual computation on these data.

Observe that the LOCAL model is exclusively focusing on the locality issue and ignores several aspects of the computation. In particular, it is synchronous and fault-free. Also, the model is oblivious to the amount of individual computation performed at each node, and it is oblivious to the amount of data that are transmitted between neighbors at each round. An important consequence of these facts is that lower bounds derived in this model are very robust, in the sense that they are not resulting from clock drifts, crashes, nor from any kind of limitation on the individual computation or on the volume

of transmitted data. Instead, lower bounds in the LOCAL model result solely from the fact that every node is unaware of what is lying beyond a certain horizon in the network and must cope with this uncertainty (Most upper bounds are however based on algorithms that perform polynomial-time individual computations at each node and exchange only a polylogarithmic amount of bits between nodes.).

Note also that the identities given to the nodes may impact the result of the computation. In particular the label ℓ produced by a construction algorithm may not only depend on G but also on the identity assignment $\mathrm{id} : V(G) \to \mathbb{N}$. The same holds for decision and verification algorithms, in which the accept/reject decision at a node may be impacted by its identity (thus, for an illegal configuration, the nodes that reject may differ depending on the identity assignment to the nodes). However, in the case of verification, it is desirable that the certificates given to the nodes do not depend on their identities, but solely on the current configuration. Indeed, the certificates should rather be solely depending on the given configuration with respect to the considered language and should better not depend on implementation factors such as, say, the IP address given to a computer (The theory of *proof-labeling scheme* [7] however refers to scenarios in which it is fully legitimate that certificates may also depend on the node identities).

Classical Tasks

Many tasks investigated in the framework of network computing are related to classical graph problems, including computing proper colorings, independent sets, matchings, dominating sets, etc. Optimization problems are however often weakened. For instance, the coloring problem considered in the distributed setting is typically $(\Delta+1)$-coloring, where Δ denotes the maximum node degree of the current network. Similarly, instead of looking for a minimum dominating set, or for a maximum independent set, one typically looks for dominating sets (resp., independent sets) that are minimal (resp., maximal) for inclusion. There are at least two reasons for such relaxations, besides the fact that the relaxed versions are sequentially solvable in polynomial time by simple greedy algorithms while the original versions are NP-hard. First, one can trivially locally decide whether a solution of the aforementioned relaxed problems satisfies the constraints of the relaxed variants, which yield the question of whether one can also construct their solutions locally (Instead, problems like minimum-weight spanning tree construction cannot be checked locally as the presence of an edge in the solution may depend of another edge, arbitrarily far in the network.). Second, these relaxed problems already involve one of the most severe difficulties distributed computing has to cope with, that is, *symmetry breaking*.

Key Results

In this section, we say that a distributed algorithm is *local* if and only if it performs in a constant number of rounds in the LOCAL model, and we are interested in identifying distributed languages that are locally constructible, locally decidable, and/or locally verifiable.

Local Algorithms

Naor and Stockmeyer [11] have thoroughly studied the distributed languages that can be locally constructed. They established that it is TM-undecidable whether a distributed language can be locally constructed, and this holds even if one restricts the problem to distributed languages that can be locally decided (On the other hand, it appears to be not easy to come up with a nontrivial example of a distributed language that can be constructed locally. One such nontrivial example is given in [11]: *weak coloring*, in which every non-isolated node must have at least one neighbor colored differently, is locally constructible for a large class of graphs. This problem is related to some *resource allocation* problem.). The crucial notion of *order-invariant* algorithms, defined as algorithms such that the

output at every node is identical for every two identity assignments that preserve the relative ordering of the identities, was also introduced in [11]. Using Ramsey theory, it is proved that in networks with constant maximum degree, for every locally decidable distributed language $\mathcal{L}$ with constant-size labels, if $\mathcal{L}$ can be constructed by a local algorithm, then $\mathcal{L}$ can also be constructed by a local order-invariant algorithm. This result has many important consequences. One is for instance the impossibility to solve $(\Delta + 1)$-coloring and maximal independent set (MIS) in a constant number of rounds. This follows from the fact that a t-round order-invariant algorithm cannot solve these problems in rings where nodes are consecutively labeled from 1 to n, because adjacent nodes with identities in $[t + 1, n - t - 1]$ must produce the same output. Another important consequence of the restriction to order-invariant algorithms is the derandomization theorem in [11] stating that, in constant degree graphs, for every locally decidable distributed language $\mathcal{L}$ with constant-size label, if $\mathcal{L}$ can be constructed by a randomized Monte Carlo local algorithm, then $\mathcal{L}$ can also be constructed by a deterministic local algorithm.

The distributed languages that can be locally decided, or verified, have been studied by Fraigniaud, Korman, and Peleg in [6]. Several complexity classes are defined and separated, and complete languages are identified for the local reduction. It is also shown in [6] that the class of all distributed languages that can be locally verified by a randomized Monte Carlo algorithm with success probability $\frac{\sqrt{5}-1}{2}$ includes *all* distributed languages. The impact of randomization is however somehow limited, at least for the class of distributed languages closed under node deletion. Indeed, [6] establishes that for any such language $\mathcal{L}$, if $\mathcal{L}$ can be locally decided by a randomized Monte Carlo algorithm with success probability greater than $\frac{\sqrt{5}-1}{2}$, then $\mathcal{L}$ can be locally decided by a deterministic algorithm. Finally, [6] additionally discusses the power of *oracles* providing nodes with information about the current network, like, typically, its number of nodes.

Almost Local Algorithms

Linial [8] proved that constructing a $(\Delta + 1)$-coloring, or, equivalently, a MIS, requires $\Omega(\log^* n)$ rounds, where $\log^* x$ is the number of times one should iterate the log function, starting from x, for reaching a value less than 1. The $\log^*$ function grows quite slowly (e.g., $\log^* 10^{100} = 5$), but is not constant. This lower bound holds even for n-node rings in which identities are in $[1, n]$, nodes know n, and nodes share a consistent notion of clockwise direction. Linial's lower bound is tight, as a 3-coloring algorithm performing in $O(\log^* n)$ rounds can be obtained by adapting the algorithm by Cole and Vishkin [5] originally designed for the PRAM model to the setting of the lower bound. Also, Linial [8] describes a $O(\log^* n)$-round algorithm for Δ^2-coloring. Note that the $\Omega(\log^* n)$-round lower bound for $(\Delta + 1)$-coloring extends to randomized Monte Carlo algorithms [10]. On the other hand, the best known upper bounds on the number of rounds to solve $(\Delta + 1)$-coloring in arbitrary graphs are $2^{O(\sqrt{\log n})}$ for deterministic algorithms [12] and expected $O(\log n)$ for randomized Las Vegas algorithms [1, 9].

By expressing the complexity of local algorithms in terms of both the size n of the network and its max-degree Δ, one can distinguish the impact of these two parameters. For instance, Linial's $O(\log^* n)$-round algorithm for Δ^2-coloring [8] can be adapted to produce an $O(\Delta^2 + \log^* n)$-round algorithm for $(\Delta + 1)$-coloring. This bound has been improved by a series of contributions, culminating to the currently best known algorithm for $(\Delta + 1)$-coloring performing in $O(\Delta + \log^* n)$ rounds [3]. Also, there is a randomized $(\Delta+1)$-coloring algorithms performing in expected $O\left(\log \Delta + \sqrt{\log n}\right)$ rounds [14]. This algorithm was recently improved to another algorithm performing in $O\left(\log \Delta + e^{O(\sqrt{\log \log n})}\right)$ rounds [4].

Additional Results

The reader is invited to consult the monograph [2] for more inputs on local distributed graph coloring algorithms, the survey [15] for a detailed

survey on local algorithms, as well as the textbook [13] for the design of distributed graph algorithms in various contexts, including the LOCAL model.

Open Problems

As far as local construction tasks are concerned, in a way similar to what happens in sequential computing, the theory of local distributed computing lacks lower bounds. (Celebrated Linial's lower bound [8] is actually one of the very few examples of a nontrivial lower bound for local computation). As a consequence, one observes large gaps between the numbers of rounds of the best known lower bounds and of the best known algorithms. This is typically the case for $(\Delta + 1)$-coloring. One of the most important open problems in this field is in fact to close these gaps for coloring as well as for many other graph problems. Similarly, although studied in depth by, e.g., Naor and Stockmeyer [11], the power of randomization is still not fully understood in the context of local computation. In general, the best known randomized algorithms are significantly faster than the best known deterministic algorithms, as witnessed by the case of $(\Delta + 1)$-coloring. Nevertheless, it is not known whether this is just an artifact of a lack of knowledge or an intrinsic separation between the two classes of algorithms.

In the context of local decision and verification tasks, the interplay between the ability to decide or verify locally and the ability to search (i.e., construct) locally is not fully understood. The completeness notions for local decision in [6] do not seem to play the same role as the completeness notions in classical complexity theory. In particular, in the context of local computing, one has not yet observed phenomena similar to self-reduction for NP-complete problems. Yet, the theory of local decision and verification is in its infancy, and it may be too early for drawing conclusions about its impact on distributed local search. An intriguing question is related to generalizing decision and verification tasks in a way similar to the polynomial hierarchy in sequential computing, by adding more alternating quantifiers in the specification of Problem 3. For instance, it would then be interesting to figure out whether each level of the hierarchy has a "natural" language as representative.

Recommended Reading

1. Alon N, Babai L, Itai A (1986) A fast and simple randomized parallel algorithm for the maximal independent set problem. J. Algorithms 7(4): 567–583
2. Barenboim L, Elkin M (2013) Distributed graph coloring: fundamentals and recent developments. Synthesis lectures on distributed computing theory. Morgan & Claypool Publishers
3. Barenboim L, Elkin M, Kuhn F (2014) Distributed (Delta+1)-coloring in linear (in delta) time. SIAM J Comput 43(1):72–95
4. Barenboim L, Elkin M, Pettie S, Schneider J (2012) The locality of distributed symmetry breaking. In: Proceedings of the 53rd IEEE symposium on foundations of computer science (FOCS), New Brunswick, pp 321–330
5. Cole R, Vishkin U (1986) Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. In: Proceedings of the 18th ACM symposium on theory of computing (STOC), Berkeley, pp 206–219
6. Fraigniaud P, Korman A, Peleg D (2013) Towards a complexity theory for local distributed computing. J ACM 60(5):35
7. Korman A, Kutten S, Peleg D (2010) Proof labeling schemes. Distrib Comput 22(4):215–233
8. Linial N (1992) Locality in distributed graph algorithms. SIAM J Comput 21(1):193–201
9. Luby M (1986) A simple parallel algorithm for the maximal independent set problem. SIAM J Comput 15:1036–1053
10. Naor M (1991) A lower bound on probabilistic algorithms for distributive ring coloring. SIAM J Discret Math 4(3):409–412
11. Naor M, Stockmeyer L (1995) What can be computed locally? SIAM J Comput 24(6): 1259–1277
12. Panconesi A, Srinivasan A (1996) On the complexity of distributed network decomposition. J Algorithms 20(2):356–374
13. Peleg D (2000) Distributed computing: a locality-sensitive approach. SIAM, Philadelphia
14. Schneider J, Wattenhofer R (2010) A new technique for distributed symmetry breaking. In: Proceedings of the 29th ACM symposium on principles of distributed computing (PODC), Zurich, pp 257–266
15. Suomela J (2013) Survey of local algorithms. ACM Comput Surv 45(2):24

Locally Decodable Codes

Shubhangi Saraf
Department of Mathematics and Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Keywords

Error correcting codes; Locally decodable codes; Sublinear time algorithms

Years and Authors of Summarized Original Work

2000; Katz, Trevisan
2002; Goldreich, Karloff, Schulman, Trevisan
2004; Kerenedis, de Wolf
2007; Woodruff
2007; Raghavendra
2008; Yekhanin
2009; Efremenko
2010; Dvir, Gopalan, Yekhanin
2010; Woodruff
2011; Kopparty, Saraf, Yekhanin
2013; Hemenway, Ostrovsky, Wootters
2013; Guo, Kopparty, Sudan
2015; Kopparty, Meir, Ron-Zewi, Saraf

Problem Definition

Classical error-correcting codes allow one to encode a k-bit message $\mathbf{x}$ into an n-bit codeword $C(\mathbf{x})$, in such a way that $\mathbf{x}$ can still be accurately recovered even if $C(\mathbf{x})$ gets corrupted in a small number of coordinates. The traditional way to recover even a small amount of information contained in $\mathbf{x}$ from a corrupted version of $C(\mathbf{x})$ is to run a traditional decoder for C, which would read and process the entire corrupted codeword, to recover the entire original message $\mathbf{x}$. The required information or required piece of $\mathbf{x}$ can then be read off. In the current digital age where huge amounts of data need to be encoded and decoded, even running in linear time to read the entire encoded data might be too wasteful, and the need for *sublinear* algorithms for error correction is greater than ever. Specially if one is only interested in recovering a single bit or a few bits of $\mathbf{x}$, it is possible to have codes with much more efficient decoding algorithms, which allow for the decoding to take place by only reading a sublinear number of code positions. Such codes are known as locally decodable codes (LDCs). Locally decodable codes allow reconstruction of an arbitrary bit $\mathbf{x}_i$ with high probability by only reading $t \ll k$ randomly chosen coordinates of (a possibly corrupted) $C(\mathbf{x})$.

The two main interesting parameters of a locally decodable code are (1) the codeword length n (as a function of the message length k) which measures the amount of redundancy that is introduced into the message by the encoder and (2) the query complexity of local decoding which counts the number of bits that need to be read from a (corrupted) codeword in order to recover a single bit of the message. Ideally, one would like to have both of these parameters as small as possible. One cannot, however, simultaneously minimize both of them; there is a trade-off. On one end of the spectrum, we have LDCs with the codeword length close to the message length, decodable with somewhat large query complexity. Such codes are useful for data storage and transmission. On the other end we have LDCs where the query complexity is a small constant, but the codeword length is large compared to the message length. Such codes find applications in complexity theory, derandomization, and cryptography (and this was the reason they were originally studied [15]). The true shape of the trade-off between the codeword length and the query complexity of LDCs is not known. Determining it is a major open problem (see [23] for an excellent recent survey of the LDC literature).

Natural variants of locally decodable codes are *locally correctable codes* (LCCs) where every symbol of the true codeword can be recovered with high probability by reading only a small number of locations of the corrupted codeword. When the underlying code is linear, it is known that every LCC is also an LDC.

Notation and Formal Definition

For a set Σ and two vectors $c, c' \in \Sigma^n$, we define the relative Hamming distance between c and c', which we denote by $\Delta(c, c')$, to be the fraction of coordinates where they differ: $\Delta(c, c' = \Pr_{i \in [n]}[c_i \neq c'_i]$.

An error-correcting code of length n over the alphabet Σ is a subset $\mathcal{C} \subseteq \Sigma^n$. The *rate* of $\mathcal{C}$ is defined to be $\frac{\log |\mathcal{C}|}{n \log |\Sigma|}$. The (minimum) distance of $\mathcal{C}$ is defined to be the smallest $\delta > 0$ such that for every distinct $c_1, c_2 \in \mathcal{C}$, we have $\Delta(c_1, c_2) \geq \delta$.

For an algorithm A and a string r, we will use A^r to represent that A is given query access to r.

We now define locally decodable codes.

Definition 1 (Locally Decodable Code) Let $\mathcal{C} \subseteq \Sigma^n$ be a code with $|\mathcal{C}| = |\Sigma|^k$. Let $E : \Sigma^k \to \mathcal{C}$ be a bijection (we refer to E as the encoding map for $\mathcal{C}$; note that k/n equals the rate of the code $\mathcal{C}$). We say that $(\mathcal{C}, E)$ is locally decodable from a δ'-fraction of errors with t queries if there is a randomized algorithm A, such that:

- **Decoding:** Whenever a message $x \in \Sigma^k$ and a received word $r \in \Sigma^n$ are such that $\Delta(r, E(x)) < \delta'$, then, for each $i \in [k]$,

$$\Pr[\mathsf{A}^r(i) = x_i] \geq 2/3.$$

- **Query complexity** t**:** Algorithm $\mathsf{A}^r(i)$ always makes at most t queries to r.

Key Results

Locally decodable codes have been implicitly studied in coding theory for a long time, starting with Reed's "majority-logic decoder" for binary Reed-Muller codes [18]. They were first formally defined by Katz and Trevisan [15] (see also [19]). Since then, the quest for understanding locally decodable codes has generated many developments.

As mentioned before, there are two main regimes in which LDCs have been studied. The first regime, on which most prior work has focused, is the regime where the query complexity is small – even a constant. In this setting, the most interesting question is to construct codes with rate as large as possible (though it is known that some significant loss in rate is inevitable). The second regime, which has been the focus of more recent work, is the high-rate regime. In this regime, one insists on the rate of the code being very close to 1 and then tries to obtain as low-query complexity as possible. We now discuss the known results in both regimes.

Low-Query Regime

A significant body of work on LDCs has focused on local decoding with a constant number of queries. Local decoding with 2 queries is almost fully understood. The Hadamard code is a 2-query LDC with codeword length $n = 2^k$. Moreover, it was shown in [9,14] that any 2-query locally decodable code (that is decodable from some small fixed constant fraction of errors) must have $n = 2^{\Omega(k)}$.

For a long time, it was generally believed that for decoding with any constant number of queries, this exponential blowup is needed: a k-bit message must be encoded into at least $\exp(k^\epsilon)$ bits, for some constant $\epsilon > 0$. This is precisely the trade-off exhibited by the Reed-Muller codes, which were believed to be optimal. Recently, in a surprising and beautiful sequence of works [6, 8, 17, 22], a new family of codes called matching vector codes was constructed, and they were shown to have local decoding parameters surprisingly much better than that of Reed-Muller codes! This family of codes gives constant query locally decodable codes which encode k bits into as few as $n = \exp(\exp(\log^\epsilon(k)))$ bits for some small (Parameter ϵ can be chosen arbitrarily close to 0 by increasing the number of queries as a function of ϵ.) constant $\epsilon > 0$ and are locally decodable from some fixed small constant fraction of errors.

There has also been considerable work [9, 14, 15, 20, 21] on lower bounds on the length of low-query locally decodable codes. However, there is still a huge gap in our understanding of the best rate achievable for any query complexity that is at least 3. For instance, for 3-query

LDCs that decode from a fixed small constant fraction of errors, the best lower bounds we know are of the form $n = \Omega(k^2)$ [21]. The best upper bounds on the other hand only give constructions with $n = \exp(\exp(\log^{O(1)}(k)))$. For codes locally decodable for general query complexity t, it is known [15] that $n = k^{1+\Omega(1/t)}$ (again, a really long way off from the upper bounds). Thus, in particular, for codes of constant rate, local decoding requires at least $\Omega(\log k)$ queries.

High-Rate Regime

In the high-rate regime, one fixes the rate to be constant, i.e., $n = O(k)$ or even $(1 + \alpha)k$, for some small constant α and then tries to construct locally decodable codes with the smallest query complexity possible. Reed-Muller codes in this regime demonstrate the following setting of parameters: for any constant $\epsilon > 0$, there is a family of Reed-Muller codes of rate $= \exp(1/\epsilon)$ that is decodable with n^{ϵ} queries. Till recently this was the best trade-off for parameters known in this regime, and in fact we did not know any family of codes of rate $> 1/2$ with any sublinear query complexity.

In the last few years, three very different families of codes have been constructed [10–12] that go well beyond the parameters of Reed-Muller codes. These codes show that for arbitrary $\alpha, \epsilon > 0$, and for every finite field $\mathbb{F}$, for infinitely many n, there is a linear code over $\mathbb{F}$ of length n with rate $1 - \alpha$, which is locally decodable (and even locally correctable) from a constant fraction (This constant is positive and is a function of only α and ϵ.) of errors with $O(n^{\epsilon})$ queries. Codes with such parameters were in fact conjectured not to exist, and this conjecture, if it were true, would have yielded progress on some well-known open questions in arithmetic circuit complexity [7].

Even more recently, it was shown that one can achieve even subpolynomial query complexity while keeping rate close to 1. In [13], it was shown that, for any $\alpha > 0$ and for every finite field $\mathbb{F}$, for infinitely many n, there is a code over $\mathbb{F}$ of length n with rate $1 - \alpha$, which is locally decodable (and even locally correctable) from a constant fraction (This constant is positive and is a function of only α.) of errors with $2^{O(\sqrt{\log n \log\log n})}$ queries.

On the lower bound front, all we know are the $n = k^{1+\Omega(1/t)}$ lower bounds by [15]. Thus, in particular, it is conceivable that for any small constant α, one could have a family of codes of rate $1 - \alpha$ that are decodable with $O(\log n)$ queries.

Applications

In theoretical computer science, locally decodable codes have played an important part in the proof-checking revolution of the early 1990s [1,2,4,16], as well as in other fundamental results in hardness amplification and pseudorandomness [3, 5, 19]. Variations of locally decodable codes are also beginning to find practical applications in data storage and data retrieval.

Open Problems

In the constant query regime, the most important question is to get the rate to be as large as possible, and the major open question in this direction is the following:

Question 1 Do there exist LDCs with polynomial rate, i.e., $n = k^{O(1)}$, that are decodable with $O(1)$ queries?

In the high-rate regime, the best query lower bounds we know are just logarithmic, and the main challenge is to construct codes with improved query complexity, hopefully coming close to the best lower bounds we can prove.

Question 2 Do there exist LDCs of rate $1 - \alpha$ or even $\Omega(1)$ that are decodable with $poly(\log n)$ queries?

Given the recent constructions of new families of codes in both the high-rate and low-query regimes with the strengthened parameters, it doesn't seem all that farfetched to imagine that we might soon be able to give much better answers to the above questions.

Recommended Reading

1. Arora S, Lund C, Motwani R, Sudan M, Szegedy M (1998) Proof verification and the hardness of approximation problems. J ACM 45(3):501–555
2. Arora S, Safra S (1998) Probabilistic checking of proofs: a new characterization of NP. J ACM 45(1):70–122
3. Arora S, Sudan M (2003) Improved low-degree testing and its applications. Combinatorica 23:365–426
4. Babai L, Fortnow L, Levin L, Szegedy M (1991) Checking computations in polylogarithmic time. In: 23rd ACM symposium on theory of computing (STOC), New Orleans, pp 21–31
5. Babai L, Fortnow L, Nisan N, Wigderson A (1993) BPP has subexponential time simulations unless EXPTIME has publishable proofs. Comput Complex 3:307–318
6. Dvir Z, Gopalan P, Yekhanin S (2010) Matching vector codes. In: 51st IEEE symposium on foundations of computer science (FOCS), Las Vegas, pp 705–714
7. Dvir Z (2010) On matrix rigidity and locally self-correctable codes. In: 26th IEEE computational complexity conference (CCC), Cambridge, pp 291–298
8. Efremenko K (2009) 3-query locally decodable codes of subexponential length. In: 41st ACM symposium on theory of computing (STOC), Bethesda, pp 39–44
9. Goldreich O, Karloff H, Schulman L, Trevisan L (2002) Lower bounds for locally decodable codes and private information retrieval. In: 17th IEEE computational complexity conference (CCC), Montréal, pp 175–183
10. Guo A, Kopparty S, Sudan M (2013) New affine-invariant codes from lifting. In: ACM innovations in theoretical computer science (ITCS), Berkeley, pp 529–540
11. Hemenway B, Ostrovsky R, Wootters M (2013) Local correctability of expander codes. In: 40th international colloquium on automata, languages, and programming (ICALP), Riga, pp 540–551
12. Kopparty S, Saraf S, Yekhanin S (2011) High-rate codes with sublinear-time decoding. In: 43rd ACM symposium on theory of computing (STOC), San Jose, pp 167–176
13. Kopparty S, Meir O, Ron-Zewi N, Saraf S (2015) High rate locally-correctable and locally-testable codes with sub-polynomial query complexity. In: Electronic colloquium on computational complexity (ECCC). TR15-068
14. Kerenidis I, de Wolf R (2004) Exponential lower bound for 2-query locally decodable codes via a quantum argument. J Comput Syst Sci 69:395–420
15. Katz J, Trevisan L (2000) On the efficiency of local decoding procedures for error-correcting codes. In: 32nd ACM symposium on theory of computing (STOC), Portland, pp 80–86
16. Lipton R (1990) Efficient checking of computations. In: 7th international symposium on theoretical aspects of computer science (STACS), Rouen. Lecture notes in computer science, vol 415, pp 207–215. Springer, Berlin/Heidelberg
17. Raghavendra P (2007) A note on Yekhanin's locally decodable codes. In: Electronic colloquium on computational complexity (ECCC). TR07-016
18. Reed IS (1954) A class of multiple-error-correcting codes and the decoding scheme. IEEE Trans Inf Theory 4:38–49
19. Sudan M, Trevisan L, Vadhan S (1999) Pseudorandom generators without the XOR lemma. In: 31st ACM symposium on theory of computing (STOC), Atlanta, pp 537–546
20. Woodruff D (2007) New lower bounds for general locally decodable codes. In: Electronic colloquium on computational complexity (ECCC). TR07-006
21. Woodruff D (2010) A quadratic lower bound for three-query linear locally decodable codes over Any Field. In: Approximation, randomization, and combinatorial optimization. Algorithms and techniques, 13th international workshop, APPROX 2010, and 14th international workshop, RANDOM, Barcelona, pp 766–779
22. Yekhanin S (2008) Towards 3-query locally decodable codes of subexponential length. J ACM 55:1–16
23. Yekhanin S (2012) Locally decodable codes. Found Trends Theor Comput Sci 6(3):139–255

Locally Testable Codes

Prahladh Harsha
Tata Institute of Fundamental Research,
Mumbai, Maharashtra, India

Keywords

Error-correcting codes; Locally checkable; PCPs; Property testing

Years and Authors of Summarized Original Work

1990; Blum, Luby, Rubinfeld
2002; Goldreich, Sudan

Problem Definition

Locally testable codes (LTC) are error-correcting codes that support algorithms which can distinguish valid codewords from words that are "far"

from all codewords by probing a given word only at a sublinear (typically constant) number of locations. LTCs are useful in the following scenario. Suppose data is transmitted by encoding it using a LTC. Then, one could check if the received data is nearly uncorrupted or has been considerably corrupted by making very few probes into the received data.

An error-correcting code $\mathcal{C} : \{0,1\}^k \to \{0,1\}^n$ is a function mapping k-bit messages to n-bit codewords. The ratio k/n is referred to as the rate of the code $\mathcal{C}$. The Hamming distance between two n-bit strings x and y, denoted by $\Delta(x,y)$, is the number of locations where x and y disagree, i.e., $\Delta(x,y) = \{i \in [n] \mid x_i \neq y_i\}$. The relative distance between x and y, denoted by $\delta(x,y)$, is the normalized distance, i.e., $\delta(x,y) = \Delta(x,y)/n$. The distance of the code $\mathcal{C}$, denote by $d(\mathcal{C})$, is the minimum Hamming distance between two distinct codewords, i.e., $d(\mathcal{C}) = \min_{x \neq y} \Delta(\mathcal{C}(x), \mathcal{C}(y))$. The distance of a string w from the code $\mathcal{C}$, denoted by $\Delta(w,\mathcal{C})$, is the distance of the nearest codeword to w, i.e., $\Delta(w,\mathcal{C}) = \min_x \Delta(w,\mathcal{C}(x))$. The relative distance of a code and the relative distance of a string w to the code are the normalized versions of the corresponding distances.

Definition 1 (locally testable code (LTC)) A code $\mathcal{C} : \{0,1\}^k \to \{0,1\}^n$ is said to be (q,δ,ε)-locally testable if there exists a probabilistic oracle algorithm T, also called a tester that, on oracle access to an input string $w \in \{0,1\}^n$, makes at most q queries (A *query* models a probe into the input string w in which one symbol (here a bit) of w is read.) to the string w and has the following properties:

Completeness: For every message $x \in \{0,1\}^k$, with probability 1 (over the tester's internal random coins), the tester T accepts the word $\mathcal{C}(x)$. Equivalently,

$$\forall x \in \{0,1\}^k, \qquad \Pr[T^{\mathcal{C}(x)} \text{ accepts }] = 1.$$

Soundness: For every string $w \in \{0,1\}^n$ such that $\Delta(w,\mathcal{C}) \geq \delta n$, the tester T rejects the word w with probability at least ε (despite reading only q bits of the word w). Equivalently,

$$\forall w \in \{0,1\}^n,$$
$$\Delta(w,\mathcal{C}) \geq \delta n \implies \Pr[T^w \text{ accepts }] \leq 1 - \varepsilon.$$

Local testability was first studied in the context of program checking by Blum, Luby, and Rubinfeld [8] who showed that the Hadamard code is locally testable (Strictly, speaking Blum et al. only showed that "Reed-Muller codes of order 1," a strict subclass of Hadamard codes are locally testable, while later Kaufman and Litsyn [15] demonstrated the local testability of the entire class of Hadamard codes.) and Gemmell et al. [11] who showed that the Reed-Muller codes are locally testable. The notion of LTCs is implicit in the work on locally checkable proofs by Babai et al. [2] and subsequent works on PCP. The explicit definition appeared independently in the works of Rubinfeld and Sudan [17], Friedl and Sudan [10], Arora's PhD thesis [1] (under the name of "probabilistically checkable proofs"), and Spielman's PhD thesis [18] (under the name of "checkable codes"). A formal study of LTCs was initiated by Goldreich and Sudan [14].

The following variants of the above definition of locally testability have also been studied.

- 2-sided vs. 1-sided error: The above definition of LTCs has perfect completeness, in the sense that every valid codeword is accepted with probability exactly 1. The tester, in this case, is said to have 1-sided error. A 2-sider error tester, on the other hand, accepts valid codewords with probability at least c for some $c \in (1-\varepsilon, 1]$. However, most constructions of LTCs have perfect completeness.
- Strong/robust LTCs: The soundness requirement in Definition 1 can be strengthened in the following sense. We can require that there exists a constant $\rho \in (0,1)$ such that for every string $w \in \{0,1\}^n$ which satisfies $\Delta(w,\mathcal{C}) \geq d$, we have

$$\Pr[T^w \text{ accepts }] \leq 1 - \frac{\rho d}{n}.$$

In other words, non-codewords which are at least a certain minimum distance from the code are not only rejected with probability at least ε but are in fact rejected with probability proportional to the distance of the non-codeword from the code. Codes that have such testers are called (q, ρ)-strong locally testable codes. They are sometimes also referred to as (q, ρ)-robust locally testable codes. Most constructions of LTCs satisfy the stronger soundness requirement.

- Adaptive vs. nonadaptive: The q queries of the tester T could either be adaptive or nonadaptive. Almost all constructions of LTCs are nonadaptive.
- Tolerant testers: Tolerant LTCs are codes with testers that accept not only valid codewords but also words which are close to the code, within a particular tolerance parameter δ' for $\delta' \leq \delta$.

LTCs are closely related to probabilistically checkable proofs (PCPs). Most known constructions of PCPs yield LTCs with similar parameters. In fact, there is a generic transformation to convert a PCP of proximity (which is a PCP with more requirements) into an LTC with comparable parameters [7,19]. See a survey by Goldreich [13] for the interplay between PCP and LTC constructions.

Locally decodable codes (LDCs), in contrast to LTCs, are codes with sublinear time decoders. Informally, such decoders can recover each message entry with high probability by probing the word at a sublinear (even constant) number of locations provided that the codeword has not been corrupted at too many locations. Observe that LTCs distinguish codewords from words that are far from the code while LDCs allow decoding from words that are close to the code.

Key Results

Local Testability of Hadamard Codes

As a first example, we present the seminal result of Blum, Luby, and Rubinfeld [8] that showed that Hadamard codes are locally testable. In this setting, we are given a string $f \in \{0,1\}^{2^k}$ and we would like to test if f is a Hadamard codeword. It will be convenient to view the 2^k-bit long string f as a function $f : \{0,1\}^k \to \{0,1\}$. In this alternate view, Hadamard codewords (strictly speaking, "Reed-Muller codewords of order 1") correspond to linear functions, i.e., they satisfy $\forall x, y \in \{0,1\}^k, f(x) + f(y) = f(x+y)$ (Here addition "+" refers to bitwise xor. In other words, $b_1 + b_2 := b_1 \oplus b_2$ for $b_1, b_2 \in \{0,1\}$ and $x + y = (x_1, x_2 \ldots, x_k) + (y_1, y_2 \ldots, y_k) := (x_1 \oplus y_1, x_2 \oplus y_2 \ldots, x_k \oplus y_k)$ for $x, y \in \{0,1\}^k$). The following test is due to Blum, Luby, and Rubinfeld. The accompanying theorem shows that this is in fact a robust characterization of linear functions.

BLR-Test

Input: Parameter k and oracle access to $f : \{0,1\}^k \to \{0,1\}$:

1. Choose $x, y \in_R \{0,1\}^k$ uniformly at random.
2. Query f at locations x, y and $x + y$.
3. Accept iff $f(x) + f(y) = f(x + y)$.

Clearly, the BLR-test always accepts all linear functions (i.e., Hadamard codewords). Blum, Luby, and Rubinfeld (with subsequent improvements due to Coppersmith) showed that if f has relative distance at least δ from all linear functions, then BLR-test rejects with probability at least $\min\{\delta/2, 2/9\}$. Their result was more general in the sense that it applied to all additive groups and not just $\{0,1\}$. For the special case of $\{0,1\}$, Bellare et al. [3] obtained the following stronger result:

Theorem 1 ([3,8]) *If f is at relative distance at least δ from all linear functions, then the BLR-test rejects f with probability at least δ.*

Local Testability of Reed-Muller Codes

Rubinfeld and Sudan [17] considered the problem of local testability of the Reed-Muller codes. Here, we consider codes over non-Boolean alphabets and the natural extension of LTCs to this non-Boolean setting. Given a field $\mathbb{F}$ and

parameters d and m (where $d+1 < |\mathbb{F}|$), the Reed-Muller code consists of codewords which are evaluations of m-variate polynomials of total degree at most d. Let $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{d+1}$ be $(d+2)$ distinct elements in the field $\mathbb{F}$ (recall that $d+1 < |\mathbb{F}|$). The following test checks if a given $f : \mathbb{F}^m \to \mathbb{F}$ is close to the Reed-Muller code.

RS-test

Input: Field $\mathbb{F}$, parameter d, and oracle access to $f : \mathbb{F}^m \to \mathbb{F}$:

1. Choose $x, y \in_R \mathbb{F}^m$ uniformly at random.
2. Query f at locations $(x + \alpha_i \cdot y), i = 1, \ldots, d+1$.
3. Interpolate to construct a univariate polynomial $q : \mathbb{F} \to \mathbb{F}$ of degree at most d such that $q(\alpha_i) = f(x+\alpha_i \cdot y), i = 1, \ldots, d+1$.
4. If $q(\alpha_0) = f(x+\alpha_0 \cdot y)$ accept, else reject.

The above test checks that the restriction of the function f to the line $l(t) = x + ty$ is a univariate polynomial of degree at most d. Clearly, multivariate polynomials of degree at most d are always accepted by the RS-test.

Theorem 2 ([17]) *There exists a constant c such that if $|\mathbb{F}| \geq 2d+2$ and $\delta < c/d^2$, then the following holds for every (positive) integer m. If f has relative distance at least δ from all m-variate polynomials of total degree at most d, then the RS-test rejects f with probability at least $\delta/2$.*

Open Problems

The Hadamard code is testable with three queries but has inverse exponential rate, whereas the Reed-Muller code (for certain setting of parameters d, m, and $\mathbb{F}$) has polylogarithmic query complexity and inverse polynomial rate. We can ask if there exist codes good with respect to both parameters. In other words, do there exist codes with inverse polynomial rate and linear distance which are testable with a constant number of queries? Such a construction, with nearly linear rate, was obtained by Ben-Sasson and Sudan [5] and Dinur [9].

Theorem 3 ([5,9]) *There exists a constant q and an* explicit *family of codes $\{\mathcal{C}_k\}_k$ where $\mathcal{C}_k : \{0,1\}^k \to \{0,1\}^{k \cdot \text{poly}\log n}$ that have linear distance and are q-locally testable.*

This construction is obtained by combining the algebraic PCP of proximity-based constructions due to Ben-Sasson and Sudan [5] with the gap amplification technique of Dinur [9]. Meir [16] obtained a LTC with similar parameters using a purely combinatorial construction, albeit a non-explicit one.

It is open if this construction can be further improved. In particular, it is open if there exist codes with constant rate and linear relative distance (such codes are usually referred to as *good codes*) that are constant query locally testable. We do not know of even a non-explicit code with such properties. To the contrary, it is known that random low-density parity check matrix (LDPC) codes are in fact not locally testable [6]. For a more detailed survey on LTCs, their constructions, and limitations, the interested reader is directed to excellent surveys by Goldreich [13], Trevisan [19], and Ben-Sasson [4].

Cross-References

▸ Linearity Testing/Testing Hadamard Codes
▸ Locally Decodable Codes

Recommended Reading

1. Arora S (1994) Probabilistic checking of proofs and the hardness of approximation problems. PhD thesis, University of California, Berkeley
2. Babai L, Fortnow L, Levin LA, Szegedy M (1991) Checking computations in polylogarithmic time. In: Proceedings of the 23rd ACM symposium on theory of computing (STOC), pp 21–31. doi:10.1145/103418.103428
3. Bellare M, Coppersmith D, Håstad J, Kiwi MA, Sudan M (1996) Linearity testing in characteristic two. IEEE Trans Inf Theory 42(6):1781–1795. doi:10.1109/18.556674, (Preliminary version in *36th FOCS*, 1995)

4. Ben-Sasson E (2010) Limitation on the rate of families of locally testable codes. In: [12], pp 13–31. doi:10.1007/978-3-642-16367-8_3
5. Ben-Sasson E, Sudan M (2008) Short PCPs with polylog query complexity. SIAM J Comput 38(2):551–607. doi:10.1137/050646445, (Preliminary version in *37th STOC*, 2005)
6. Ben-Sasson E, Harsha P, Raskhodnikova S (2005) Some 3CNF properties are hard to test. SIAM J Comput 35(1):1–21. doi:10.1137/S0097539704445445, (Preliminary version in *35th STOC*, 2003)
7. Ben-Sasson E, Goldreich O, Harsha P, Sudan M, Vadhan S (2006) Robust PCPs of proximity, shorter PCPs and applications to coding. SIAM J Comput 36(4):889–974. doi:10.1137/S0097539705446810, (Preliminary version in *36th STOC*, 2004)
8. Blum M, Luby M, Rubinfeld R (1993) Self-testing/correcting with applications to numerical problems. J Comput Syst Sci 47(3):549–595. doi:10.1016/0022-0000(93)90044-W, (Preliminary version in *22nd STOC*, 1990)
9. Dinur I (2007) The PCP theorem by gap amplification. J ACM 54(3):12. doi:10.1145/1236457.1236459, (Preliminary version in *38th STOC*, 2006)
10. Friedl K, Sudan M (1995) Some improvements to total degree tests. In: Proceedings of the 3rd Israel symposium on theoretical and computing systems, pp 190–198. Corrected version available online at http://theory.lcs.mit.edu/~madhu/papers/friedl.ps
11. Gemmell P, Lipton RJ, Rubinfeld R, Sudan M, Wigderson A (1991) Self-testing/correcting for polynomials and for approximate functions. In: Proceedings of the 23rd ACM symposium on theory of computing (STOC), pp 32–42. doi:10.1145/103418.103429
12. Goldreich O (ed) (2010) Property testing (Current research and surverys). Lecture notes in computer science, vol 6390. Springer, Berlin
13. Goldreich O (2010) Short locally testable codes and proofs: a survey in two parts. In: [12], pp 65–104. doi:10.1007/978-3-642-16367-8_6
14. Goldreich O, Sudan M (2006) Locally testable codes and PCPs of almost linear length. J ACM 53(4):558–655. doi:10.1145/1162349.1162351, (Preliminary Verison in *43rd FOCS*, 2002)
15. Kaufman T, Litsyn S (2005) Almost orthogonal linear codes are locally testable. In: Proceedings of the 46th IEEE symposium on foundations of computer science (FOCS), pp 317–326. doi:10.1109/SFCS.2005.16
16. Meir O (2009) Combinatorial construction of locally testable codes. SIAM J Comput 39(2):491–544. doi:10.1137/080729967, (Preliminary version in *40th STOC*, 2008)
17. Rubinfeld R, Sudan M (1996) Robust characterizations of polynomials with applications to program testing. SIAM J Comput 25(2):252–271. doi:10.1137/S0097539793255151, (Preliminary version in *23rd STOC*, 1991 and *3rd SODA*, 1992)
18. Spielman DA (1995) Computationally efficient error-correcting codes and holographic proofs. PhD thesis, Massachusetts Institute of Technology. http://www.cs.yale.edu/homes/spielman/Research/thesis.html
19. Trevisan L (2004) Some applications of coding theory in computational complexity. Quaderni di Matematica 13:347–424. cs/0409044

Low Stretch Spanning Trees

Michael Elkin
Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Keywords

Spanning trees with low average stretch

Years and Authors of Summarized Original Work

2005; Elkin, Emek, Spielman, Teng

Problem Definition

Consider a weighted connected multigraph $G = (V, E, \omega)$, where ω is a function from the edge set E of G into the set of positive reals. For a path P in G, the *weight* of P is the sum of weights of edges that belong to the path P. For a pair of vertices $u, v \in V$, the *distance* between them in G is the minimum weight of a path connecting u and v in G. For a spanning tree T of G, the stretch of an edge $(u,v) \in E$ is defined by

$$\text{stretch}_T(u, v) = \frac{\text{dist}_T(u, v)}{\text{dist}_G(u, v)},$$

and the average stretch over all edges of E is

$$\text{avestr}(G,T) = \frac{1}{|E|}\sum_{(u,v)\in E} \text{stretch}_T(u,v).$$

The average stretch of a multigraph $G = (V, E, \omega)$ is defined as the smallest average stretch of a spanning tree T of G, $\text{avestr}(G,T)$. The average stretch of a positive integer n, $\text{avestr}(n)$, is the maximum average stretch of an n-vertex multigraph G. The problem is to analyze the asymptotic behavior of the function $\text{avestr}(n)$.

A closely related (dual) problem is to construct a probability distribution $\mathcal{D}$ of spanning trees for G, so that

$$\text{expstr}(G,\mathcal{D}) = \max_{e=(u,v)\in E} \mathbb{E}_{T\in\mathcal{D}}(\text{stretch}_T(u,v))$$

is small as possible. Analogously, $\text{expstr}(G) = \min_{\mathcal{D}}\{\text{expstr}(G,\mathcal{D})\}$, where the minimum is over all distributions $\mathcal{D}$ of spanning trees of G, and $\text{expstr}(n) = \max_G\{\text{expstr}(G)\}$, where the maximum is over all n-vertex multigraphs.

By viewing the problem as a 2-player zero-sum game between a tree player that aims to minimize the payoff and an edge player that aims to maximize it, it is easy to see that for every positive integer n, $\text{avestr}(n) = \text{expstr}(n)$ [3]. The probabilistic version of the problem is, however, particularly convenient for many applications.

Key Results

The problem was studied since 1960s [9, 14, 16, 17]. A major progress in its study was achieved by Alon et al. [3], who showed that

$$\Omega(\log n) = \text{avestr}(n) = \text{expstr}(n)$$
$$= \exp\left(O\left(\sqrt{\log n \cdot \log\log n}\right)\right).$$

Elkin et al. [10] improved the upper bound and showed that

$$\text{avestr}(n) = \text{expstr}(n) = O\left(\log^2 n \cdot \log\log n\right).$$

Applications

One application of low-stretch spanning trees is for solving symmetric diagonally dominant linear systems of equations. Boman and Hendrickson [6] were the first to discover the surprising relationship between these two seemingly unrelated problems. They applied the spanning trees of [3] to design solvers that run in time $m^{3/2}2^{O(\sqrt{\log n \log\log n})\log(1/\epsilon)}$. Spielman and Teng [15] improved their results by showing how to use the spanning trees of [3] to solve diagonally dominant linear systems in time

$$m2^{O(\sqrt{\log n \log\log n})}\log(1/\epsilon).$$

By applying the low-stretch spanning trees developed in [10], the time for solving these linear systems reduces to

$$m\log^{O(1)} n \log(1/\epsilon),$$

and to $O(n(\log n \log\log n)^2 \log(1/\epsilon))$ when the systems are planar. Applying a recent reduction of Boman, Hendrickson, and Vavasis [7], one obtains a $O(n(\log n \log\log n)^2 \log(1/\epsilon))$ time algorithm for solving the linear systems that arise when applying the finite element method to solve two-dimensional elliptic partial differential equations.

Chekuri et al. [8] used low-stretch spanning trees to devise an approximation algorithm for nonuniform buy-at-bulk network design problem. Their algorithm provides a first polylogarithmic approximation guarantee for this problem.

Abraham et al. [2] use a technique of Star decomposition introduced by Elkin et al. [10] to construct embeddings with a constant average stretch, where the average is over all *pairs of vertices*, rather than over all edges. The result of Abraham et al. [2] was, in turn, already used in

a yet more recent work of Elkin et al. [11] on fundamental circuits.

Open Problems

Abraham and Neiman [1] subsequently devised an algorithm for constructing a spanning tree with average stretch $O(\log n \log\log n)$. The most evident open problem is to close the gap between this algorithm and the $\Omega(\log n)$ lower bound. Another intriguing subject is the study of low-stretch spanning trees for various restricted families of graphs. Progress in this direction was recently achieved by Emek and Peleg [12] that constructed low-stretch spanning trees with average stretch $O(\log n)$ for unweighted series-parallel graphs. Discovering other applications of low-stretch spanning trees is another promising venue of study.

Finally, there is a closely related relaxed notion of low-stretch *Steiner* or *Bartal* trees. Unlike a spanning tree, a Steiner tree does not have to be a subgraph of the original graph, but rather is allowed to use edges and vertices that were not present in the original graph. It is, however, required that the distances in the Steiner tree will be no smaller than the distances in the original graph. Low-stretch Steiner trees were extensively studied [4, 5, 13]. Fakcharoenphol et al. [13] devised a construction of low-stretch Steiner trees with an average stretch of $O(\log n)$. It is currently unknown whether the techniques used in the study of low-stretch Steiner trees can help in improving the bounds for the low-stretch spanning trees.

Cross-References

▸ Approximating Metric Spaces by Tree Metrics

Recommended Reading

1. Abraham I, Neiman O (2012) Using petal-decompositions to build low-stretch spanning trees. In: Proceedings of the 44th annual ACM symposium on theory of computing, New York, June 2012, pp 395–406
2. Abraham I, Bartal Y, Neiman O (2007) Embedding metrics into ultrametrics and graphs into spanning trees with constant average distortion. In: Proceedings of the 18th ACM-SIAM symposium on discrete algorithms, New Orleans, Jan 2007
3. Alon N, Karp RM, Peleg D, West D (1995) A graph-theoretic game and its application to the k-server problem. SIAM J Comput 24(1):78–100. Also available technical report TR-91-066, ICSI, Berkeley (1991)
4. Bartal Y (1996) Probabilistic approximation of metric spaces and its algorithmic applications. In: Proceedings of the 37th annual symposium on foundations of computer science, Berlington, Oct 1996, pp 184–193
5. Bartal Y (1998) On approximating arbitrary metrices by tree metrics. In: Proceedings of the 30th annual ACM symposium on theory of computing, Dallas, 23–26 May 1998, pp 161–168
6. Boman E, Hendrickson B (2001) On spanning tree preconditioners. Manuscript, Sandia National Lab
7. Boman E, Hendrickson B, Vavasis S (2004) Solving elliptic finite element systems in near-linear time with suppost preconditioners. Manuscript, Sandia National Lab and Cornell. http://arXiv.org/abs/cs/0407022. Accessed 9 July 2004
8. Chekuri C, Hagiahayi MT, Kortsarz G, Salavatipour M (2006) Approximation algorithms for non-uniform buy-at-bulk network design. In: Proceedings of the 47th annual symposium on foundations of computer science, Berkeley, Oct 2006, pp 677–686
9. Deo N, Prabhu GM, Krishnamoorthy MS (1982) Algorithms for generating fundamental cycles in a graph. ACM Trans Math Softw 8:26–42
10. Elkin M, Emek Y, Spielman D, Teng S-H (2005) Lower-stretch spanning trees. In: Proceedings of the 37th annual ACM symposium on theory of computing (STOC'05), Baltimore, May 2005, pp 494–503
11. Elkin M, Liebchen C, Rizzi R (2007) New length bounds for cycle bases. Inf Process Lett 104(5):186–193
12. Emek Y, Peleg D (2006) A tight upper bound on the probabilistic embedding of series-parallel graphs. In: Proceedings of symposium on discrete algorithms (SODA'06), Miami, Jan 2006, pp 1045–1053
13. Fakcharoenphol J, Rao S, Talwar K (2003) A tight bound on approximating arbitrary metrics by tree metrics. In: Proceedings of the 35th annual ACM symposium on theory of computing, San Diego, June 2003, pp 448–455
14. Horton JD (1987) A polynomial-time algorithm to find the shortest cycle basis of a graph. SIAM J Comput 16(2):358–366
15. Spielman D, Teng S-H (2004) Nearly-linear time algorithm for graph partitioning, graph sparsification, and solving linear systems. In: Proceedings of the 36th annual ACM symposium on theory of computing (STOC'04), Chicago, June 2004, pp 81–90

16. Stepanec GF (1964) Basis systems of vector cycles with extremal properties in graphs. Uspekhi Mat Nauk 19:171–175. (In Russian)
17. Zykov AA (1969) Theory of finite graphs. Nauka, Novosibirsk. (In Russian)

Lower Bounds Based on the Exponential Time Hypothesis: Edge Clique Cover

Michał Pilipczuk
Institute of Informatics, University of Warsaw, Warsaw, Poland
Institute of Informatics, University of Bergen, Bergen, Norway

Keywords

Cocktail party graph; Edge clique cover; Exponential Time Hypothesis; Parameterized complexity

Years and Authors of Summarized Original Work

2008; Gramm, Guo, Hüffner, Niedermeier
2013; Cygan, Pilipczuk, Pilipczuk

The Exponential Time Hypothesis and Its Consequences

In 2001, Impagliazzo, Paturi, and Zane [5, 6] introduced the *Exponential Time Hypothesis* (ETH): a complexity assumption saying that there exists a constant $c > 0$ such that no algorithm for 3-SAT can achieve the running time of $\mathcal{O}(2^{cn})$, where n is the number of variables of the input formula. In particular, this implies that there is no *subexponential-time* algorithm for 3-SAT, that is, one with running time $2^{o(n)}$. The key result of Impagliazzo, Paturi, and Zane is the *Sparsification Lemma*, proved in [6]. Without going into technical details, the Sparsification Lemma provides a reduction that allows us to assume that the input instance of 3-SAT is sparse in the following sense: the number of clauses is linear in the number of variables. Thus, a direct consequence is that, assuming ETH, there is a constant $c > 0$ such that there is no algorithm for 3-SAT with running time $\mathcal{O}(2^{c(n+m)})$. Hence, an algorithm with running time $2^{o(n+m)}$ is excluded in particular.

After the introduction of the Exponential Time Hypothesis and the Sparsification Lemma, it turned out that ETH can be used as a robust assumption for proving sharp lower bounds on the time complexity of various computational problems. For many classic NP-hard graph problems, like Vertex Cover, 3-Coloring, or Hamiltonian Cycle, the known NP-hardness reductions from 3-SAT are *linear*, i.e., they transform an instance of 3-SAT with n variables and m clauses into an instance of the target problem whose total size is $\mathcal{O}(n + m)$. Consequently, if any of these problems admitted an algorithm with running time $2^{o(N+M)}$, where N and M are the numbers of vertices and edges of the graph, respectively, then the composition of the reduction and such an algorithm would yield an algorithm for 3-SAT with running time $2^{o(n+m)}$, thus contradicting ETH. As all these problems indeed can be solved in time $2^{\mathcal{O}(N)}$, this shows that the single-exponential running time is essentially optimal. The same problems restricted to planar graphs have NP-hardness reductions from 3-SAT with a quadratic size blowup, which excludes the existence of $2^{o(\sqrt{N+M})}$ algorithms under ETH. Again, this is matched by $2^{\mathcal{O}(\sqrt{N})}$ algorithms obtained using the Lipton-Tarjan planar separator theorem.

Of particular interest to us are applications in parameterized complexity. Recall that a parameterized problem is *fixed-parameter tractable* (FPT) if there is an algorithm solving it in time $f(k) \cdot n^c$, where n is the total input size, k is the parameter, f is some computable function, and c is a universal constant. Observe that if we provide a reduction from 3-SAT to a parameterized problem where the output parameter depends linearly on $n + m$, then assuming ETH we exclude the existence of a *subexponential parameterized algorithm*, i.e., one with running

time $2^{o(k)} \cdot n^{\mathcal{O}(1)}$. If the dependence of the output parameter on $n + m$ is different, then we obtain a lower bound for a different function f. This idea has been successfully applied for many various parameterizations and different running times. For example, Lokshtanov et al. [8] introduced a framework for proving lower bounds excluding the running time of the form $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$. The framework can be used to show the optimality of known FPT algorithm for several important problems, with a notable example of CLOSEST STRING.

More information on lower bounds based on ETH can be found in the survey of Lokshtanov et al. [7] or in the PhD thesis of the current author [9].

Problem Definition

In the EDGE CLIQUE COVER problem, we are given a graph G and an integer k, and the question is whether one can find k complete subgraphs $C_1, C_2, \ldots, C_k$ of G such that $E(G) = \bigcup_{i=1}^{k} E(C_i)$. In other words, we have to cover the whole edge set of G using k complete subgraphs of G. Such a selection of k complete subgraphs is called an *edge clique cover*.

The study of the parameterized complexity of EDGE CLIQUE COVER was initiated by Gramm et al. [3]. The main observation of Gramm et al. is the applicability of the following data reduction rule: as long as in G there exists a pair of *perfect twins* (i.e., adjacent vertices having exactly the same neighborhood), then it is safe to remove one of them (and decrement the parameter k in the case when these twins form an isolated edge in G). Once there are no perfect twins and no isolated vertices in the graph (the latter ones can be also safely removed), then one can easily show the following: there is no edge clique cover of size less than $\log |V(G)|$. Consequently, instances with $k < \log |V(G)|$ can be discarded as no-instances, and we are left with instances satisfying $|V(G)| \leq 2^k$. In the language of parameterized complexity, this is called a *kernel* with 2^k vertices. By applying a standard covering dynamic programming algorithm on this kernel, we obtain an FPT algorithm for EDGE CLIQUE COVER with running time $2^{2^{\mathcal{O}(k)}} + |V(G)|^{\mathcal{O}(1)}$; the second summand is the time needed to apply the data reduction rule exhaustively.

Given the striking simplicity of the approach of Gramm et al. [3], the natural open question was whether the obtained double-exponential running time of the algorithm for EDGE CLIQUE COVER could be improved.

Key Results

This question has been resolved by Cygan et al. [1], who showed that, under ETH, the running time obtained by Gramm et al. [3] is essentially optimal. More precisely, they proved the following result:

Lemma 1 *There exists a polynomial-time algorithm that, given an instance φ of* 3-SAT *with n variables and m clauses, constructs an equivalent* EDGE CLIQUE COVER *instance (G, k) with $k = \mathcal{O}(\log n)$ and $|V(G)| = \mathcal{O}(n + m)$.*

Thus, by considering a composition of the reduction of Lemma 1 with a hypothetical algorithm for EDGE CLIQUE COVER with running time $2^{2^{o(k)}} \cdot |V(G)|^{\mathcal{O}(1)}$, we obtain the following lower bound:

Theorem 1 *Unless ETH fails, there is no algorithm for* EDGE CLIQUE COVER *with running time $2^{2^{o(k)}} \cdot |V(G)|^{\mathcal{O}(1)}$.*

Curiously, Lemma 1 can be also used to show that the kernelization algorithm of Gramm et al. [3] is also essentially optimal. More precisely, we have the following theorem.

Theorem 2 *There exists a universal constant $\varepsilon > 0$ such that, unless* P = NP, *there is no constant λ and a polynomial-time algorithm $\mathcal{A}$ that takes an instance (G, k) of* EDGE CLIQUE COVER *and outputs an equivalent instance (G', k') of* EDGE CLIQUE COVER *with binary encoding of length at most $\lambda \cdot 2^{\varepsilon k}$.*

The idea of the proof of Theorem 2 is to consider the composition of three algorithms: (i) the reduction of Lemma 1, (ii) a hypothetical algorithm as in the statement of Theorem 2 for a very small $\varepsilon > 0$, and (iii) a polynomial-time reduction from EDGE CLIQUE COVER to 3-SAT, whose existence follows from the fact that the former problem is in NP and the latter one is NP-hard. Since the constants hidden in the bounds for algorithms (i) and (iii) are universal, for some very small $\varepsilon > 0$ this composition would result in an algorithm that takes any instance of 3-SAT on n variables and shrinks it to an instance that has total bitsize $o(n)$, i.e., *sublinear* in n. Hence, by applying this algorithm multiple times, we would eventually shrink the instance at hand to constant size, and then we could solve it by brute force. As all the algorithms involved run in polynomial time, this would imply that P = NP.

We remark that Theorem 2 was not observed in the original work of Cygan et al. [1], but its proof can be found in the PhD thesis of the current author [9], and it will appear in the upcoming journal version of [1]. Also, in an earlier work, Cygan et al. [2] proved a weaker statement that EDGE CLIQUE COVER does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. Theorem 2 shows that even a subexponential kernel is unlikely under the weaker assumption of P $\neq$ NP.

Let us now shed some light on the proof of Lemma 1, which is the crucial technical ingredient of the results. The main idea is to base the reduction on the analysis of the *cocktail party graph*: for an integer $n > 1$, the cocktail party graph H_{2n} is defined as a complete graph on $2n$ vertices with a perfect matching removed. Observe that H_{2n} does not contain any perfect twins, so it is immune to the data reduction rule of Gramm et al. [3] and the minimum size of an edge clique cover in H_{2n} is at least $1 + \log n$. On the other hand, it is relatively easy to construct a large family of edge clique covers of H_{2n} that have size $2\lceil \log n \rceil$. Actually, the question of determining the minimum size of an edge clique cover in H_{2n} was studied from a purely combinatorial point of view: Gregory and Pullman [4] proved that it is equal to $\inf\{k : n \leq \binom{k-1}{\lceil k/2 \rceil}\}$, answering an earlier question of Orlin. Note that this value is larger than $1 + \log n$ only by an additive factor of $\mathcal{O}(\log \log n)$.

Thus, the cocktail party graph provides a natural example of a hard instance where the parameter is logarithmic. The crux of the construction is to start with the cocktail party graph H_{2n} and, by additional gadgeteering, force the solution inside it to belong to the aforementioned family of edge clique covers of size $2\lceil \log n \rceil$. The behavior of these edge clique covers (called *twin covers*) can be very well understood, and we can encode the evaluation of the variables of the input 3-SAT formula as a selection of a twin cover to cover H_{2n}. In order to verify that the clauses of the input formula are satisfied, we construct additional clause gadgets. This involves only a logarithmic number of additional cliques and is based on careful constructions using binary encodings.

Discussion

After announcing the results of Cygan et al. [1], there was some discussion about their actual meaning. For instance, some authors suggested that the surprisingly high lower bound for EDGE CLIQUE COVER may be a argument against the plausibility of the Exponential Time Hypothesis. Our view on this is quite different: the double-exponential lower bound suggests that EDGE CLIQUE COVER is an inherently hard problem, even though it may not seem as such at first glance. The relevant parameter in this problem is not really the number of cliques k, but rather 2^k, the number of possible different neighborhoods that can arise in a graph that is a union of k complete graphs. The lower bound of Cygan et al. [1] intuitively shows that one cannot expect to significantly reduce the number of neighborhoods that needs to be considered.

Recommended Reading

1. Cygan M, Pilipczuk M, Pilipczuk M (2013) Known algorithms for edge clique cover are probably optimal. In: Proceedings of the twenty-fourth annual ACM-SIAM symposium on discrete algorithms, SODA 2013, New Orleans, 6–8 Jan 2013, pp 1044–1053
2. Cygan M, Kratsch S, Pilipczuk M, Pilipczuk M, Wahlström M (2014) Clique cover and graph separation: new incompressibility results. TOCT 6(2):6
3. Gramm J, Guo J, Hüffner F, Niedermeier R (2008) Data reduction and exact algorithms for clique cover. ACM J Exp Algorithmics 13:article 2.2
4. Gregory DA, Pullman NJ (1982) On a clique covering problem of Orlin. Discret Math 41(1): 97–99
5. Impagliazzo R, Paturi R (2001) On the complexity of k-SAT. J Comput Syst Sci 62(2):367–375
6. Impagliazzo R, Paturi R, Zane F (2001) Which problems have strongly exponential complexity? J Comput Syst Sci 63(4):512–530
7. Lokshtanov D, Marx D, Saurabh S (2011) Lower bounds based on the exponential time hypothesis. Bull EATCS 105:41–72
8. Lokshtanov D, Marx D, Saurabh S (2011) Slightly superexponential parameterized problems. In: Proceedings of the twenty-second annual ACM-SIAM symposium on discrete algorithms, SODA 2011, San Francisco, 23–25 Jan 2011. SIAM, pp 760–776
9. Pilipczuk M (2013) Tournaments and optimality: new results in parameterized complexity. PhD thesis, University of Bergen, Norway. Available at the webpage of the author

Lower Bounds for Dynamic Connectivity

Mihai Pătraşcu
Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology (MIT), Cambridge, MA, USA

Keywords

Dynamic trees

Years and Authors of Summarized Original Work

2004; Pătraşcu, Demaine

Problem Definition

The dynamic connectivity problem requests maintenance of a graph G subject to the following operations:

insert(u, v): insert an undirected edge (u, v) into the graph.
delete(u, v): delete the edge (u, v) from the graph.
connected(u, v): test whether u and v lie in the same connected component.

Let m be an upper bound on the number of edges in the graph. This entry discusses cell-probe lower bounds for this problem. Let t_u be the complexity of `insert` and `delete` and t_q the complexity of `query`.

The Partial-Sums Problem

Lower bounds for dynamic connectivity are intimately related to lower bounds for another classic problem: maintaining partial sums. Formally, the problem asks one to maintain an array $A[1..n]$ subject to the following operations:

update(k, Δ): let $A[k] \leftarrow \Delta$.
sum(k): returns the partial sum $\sum_{i=1}^{k} A[i]$.
testsum(k, σ): returns a boolean value indicating whether $\texttt{sum}(k) = \sigma$.

To specify the problem completely, let elements $A[i]$ come from an arbitrary group G containing at least 2^{δ} elements. In the cell-probe model with b-bit cells, let t_u^{Σ} be the complexity of `update` and t_q^{Σ} the complexity of `testsum` (which is also a lower bound on `sum`).

The tradeoffs between t_u^{Σ} and t_q^{Σ} are well understood for all values of b and δ. However, this entry only considers lower bounds under the standard assumptions that $b = \Omega(\lg n)$ and $t_u \geq t_q$. It is standard to assume $b = \Omega(\lg n)$ for upper bounds in the RAM model; this assumption also means that the lower bound applies to the pointer machine. Then, Pătraşcu and Demaine [6] prove:

Theorem 1 *The complexity of the partial-sums problems satisfies:* $t_q^{\Sigma} \cdot \lg(t_u^{\Sigma}/t_q^{\Sigma}) = \Omega(\delta/b \cdot \lg n)$.

Observe that this matches the textbook upper bound using augmented trees. One can build a balanced binary tree over $A[1], \ldots, A[n]$ and store in every internal node the sum of its subtree. Then, updates and queries touch $O(\lg n)$ nodes (and spend $O(\lceil \delta/b \rceil)$ time in each one due to the size of the group). To decrease the query time, one can use a B-tree.

Relation to Dynamic Connectivity

We now clarify how lower bounds for maintaining partial sums imply lower bounds for dynamic connectivity. Consider the partial-sums problem over the group $G = S_n$, i.e., the permutation group on n elements. Note that $\delta = \lg(n!) = \Omega(n \lg n)$. It is standard to set $b = \Theta(\lg n)$, as this is the natural word size used by dynamic connectivity upper bounds. This implies $t_q^{\Sigma} \lg(t_u^{\Sigma}/t_q^{\Sigma}) = \Omega(n \lg n)$.

The lower bound follows from implementing the partial-sums operations using dynamic connectivity operations. Refer to Fig. 1. The vertices of the graph form an integer grid of size $n \times n$. Each vertex is incident to at most two edges, one edge connecting to a vertex in the previous column and one edge connecting to a vertex in the next column. Point (x, y_1) in the grid is connected to point $(x + 1, A[x](y_1))$, i.e.,the edges between two adjacent columns describe the corresponding permutation from the partial-sums vector.

To implement `update` (x, π), all the edges between column x and $x + 1$ are first deleted and then new edges are inserted according to π. This gives $t_u^{\Sigma} = O(2n \cdot t_u)$. To implement `testsum` (x, π), one can use n `connected` queries between the pairs of points $(1, y) \rightsquigarrow (x + 1, \pi(y))$. Then, $t_q^{\Sigma} = O(n \cdot t_q)$. Observe that the `sum` query cannot be implemented as easily. Dynamic connectivity is the main motivation to study the `testsum` query.

The lower bound of Theorem 1 translates into $nt_q \cdot \lg(2nt_u/nt_q) = \Omega(n \lg n)$; hence $t_q \lg(t_u/t_q) = \Omega(\lg n)$. Note that this lower bound implies $\max\{t_u, t_q\} = \Omega(\lg n)$. The best known upper bound (using amortization and randomization) is $O(\lg n(\lg \lg n)^3)$ [9]. For any $t_u = \Omega(\lg n(\lg \lg n)^3)$, the lower bound tradeoff is known to be tight. Note that the graph in the lower bound is always a disjoint union of paths. This implies optimal lower bounds for two important special cases: dynamic trees [8] and dynamic connectivity in plane graphs [2].

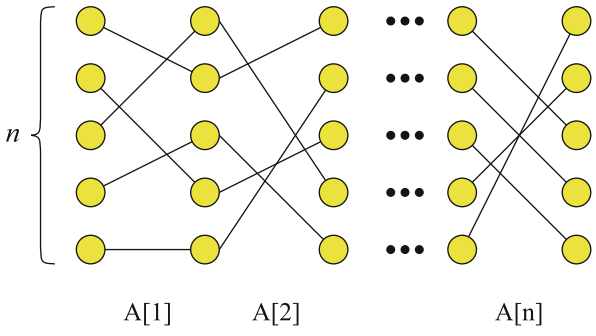
Key Results

Understanding Hierarchies

Epochs

To describe the techniques involved in the lower bounds, first consider the `sum` query and assume $\delta = b$. In 1989, Fredman and Saks [3] initiated the study of dynamic cell-probe lower bounds, essentially showing a lower bound of $t_q^{\Sigma} \lg t_u^{\Sigma} = \Omega(\lg n)$. Note that this implies $\max\{t_q^{\Sigma}, t_u^{\Sigma}\} = \Omega(\lg n / \lg \lg n)$.

At an intuitive level, their argument proceeded as follows. The hard instance will have n random updates, followed by one random query. Leave $r \geq 2$ to be determined. Looking *back* in time



Lower Bounds for Dynamic Connectivity, Fig. 1 Constructing an instance of dynamic connectivity that mimics the partial-sums problem

from the query, one groups the updates into exponentially growing epochs: the latest r updates are epoch 1, the earlier r^2 updates are epoch 2, etc. Note that epoch numbers increase going back in time, and there are $O(\log_r n)$ epochs in total.

For some epoch i, consider revealing to the query all updates performed in all epochs different from i. Then, the query reduces to a partial-sums query among the updates in epoch i. Unless the query is to an index below the minimum index updated in epoch i, the answer to the query is still uniformly random, i.e., has δ bits of entropy. Furthermore, even if one is given, say, $r^i\delta/100$ bits of information about epoch i, the answer still has $\Omega(\delta)$ bits of entropy on average. This is because the query and updates in epoch i are uniformly random, so the query can ask for any partial sum of these updates, uniformly at random. Each of the r^i partial sums is an independent random variable of entropy δ.

Now one can ask how much information is available to the query. At the time of the query, let each cell be associated with the epoch during which it was last written. Choosing an epoch i uniformly at random, one can make the following intuitive argment:

1. No cells written by epochs $i+1, i+2, \ldots$ can contain information about epoch i, as they were written in the past.
2. In epochs $1, \ldots, i-1$, a number of $bt_u^\Sigma \cdot \sum_{j=1}^{i-1} r^j \leq bt_u^\Sigma \cdot 2r^{i-1}$ bits were written. This is less than $r^i\delta/100$ bits of information for $r > 200t_u^\Sigma$ (recall the assumption $\delta = b$). By the above, this implies the query answer still has $\Omega(\delta)$ bits of entropy.
3. Since i is uniformly random among $\Theta(\log_r n)$ epochs, the query makes an expected $O(t_q^\Sigma / \log_r n)$ probes to cells from epoch i. All queries that make no cell probes to epoch i have a fixed answer (entropy 0), and all other queries have answers of entropy $\leq \delta$. Since an average query has entropy $\Omega(\delta)$, a query must probe a cell from epoch i with constant probability. That means $t_q^\Sigma / \log_r n = \Omega(1)$, and $\sum = \Omega(\log_r n) = \Omega(\lg n / \lg t_u^\Sigma)$.

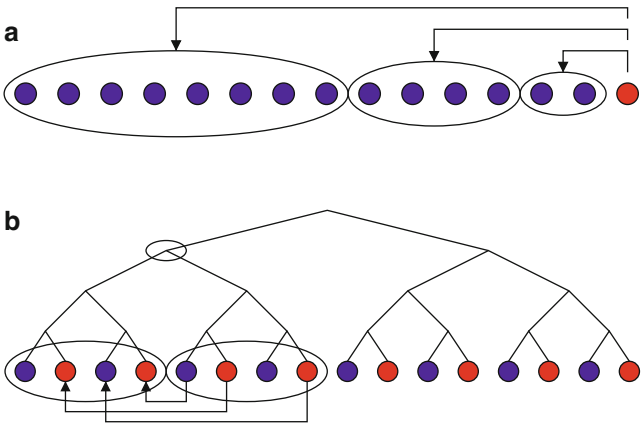
One should appreciate the duality between the proof technique and the natural upper bounds based on a hierarchy. Consider an upper bound based on a tree of degree r. The last r random updates (epoch 1) are likely to be uniformly spread in the array. This means the updates touch different children of the root. Similarly, the r^2 updates in epoch 2 are likely to touch every node on level 2 of the tree, and so on. Now, the lower bound argues that the query needs to traverse a root-to-leaf path, probing a node on every level of the tree (this is equivalent to one cell from every epoch).

Time Hierarchies

Despite considerable refinement to the lower bound techniques, the lower bound of $\Omega(\lg n / \lg\lg n)$ was not improved until 2004. Then, Pătraşcu and Demaine [6] showed an optimal bound of $t_q^\Sigma \lg(t_u^\Sigma / t_q^\Sigma) = \Omega(\lg n)$, implying $\max\{t_u^\Sigma, t_q^\Sigma\} = \Omega(\lg n)$. For simplicity, the discussion below disregards the tradeoff and just sketches the $\Omega(\lg n)$ lower bound.

Pătraşcu and Demaine's [6] counting technique is rather different from the epoch technique; refer to Fig. 2. The hard instance is a sequence of n operations alternating between updates and queries. They consider a balanced binary tree over the time axis, with every leaf being an operation. Now for every node of the tree, they propose to count the number of cell probes made in the right subtree to a cell written in the left subtree. Every probe is counted exactly once, for the lowest common ancestor of the read and write times.

Now focus on two sibling subtrees, each containing k operations. The $k/2$ updates in the left subtree, and the $k/2$ queries in the right subtree, are expected to interleave in index space. Thus, the queries in the right subtree ask for $\Omega(k)$ different partial sums of the updates in the left subtree. Thus, the right subtree "needs" $\Omega(k\delta)$ bits of information about the left subtree, and this information can only come from cells written in the left subtree and read in the right one. This implies



Lower Bounds for Dynamic Connectivity, Fig. 2 Analysis of cell probes in the **a** epoch-based and **b** time-hierarchy techniques

a lower bound of $\Omega(k)$ probes, associated with the parent of the sibling subtrees. This bound is linear in the number of leaves, so summing up over the tree, one obtains a total $\Omega(n \lg n)$ lower bound, or $\Omega(\lg n)$ cost per operation.

An Optimal Epoch Construction

Rather surprisingly, Pătraşcu and Tarniţă [7] managed to reprove the optimal tradeoff of Theorem 1 with minimal modifications to the epoch argument. In the old epoch argument, the information revealed by epochs $1, \ldots, i-1$ about epoch i was bounded by the number of cells written in these epochs. The key idea is that an equally good bound is the number of cells read during epochs $1, \ldots, i-1$ and written during epoch i.

In principle, all cell reads from epoch $i-1$ could read data from epoch i, making these two bounds identical. However, one can randomize the epoch construction by inserting the query after an unpredictable number of updates. This randomization "smooths" out the distribution of epochs from which cells are read, i.e., a query reads $O(t_q^{\Sigma} / \log_r n)$ cells from every epoch, in expectation over the randomness in the epoch construction. Then, the $O(r^{i-1})$ updates in epochs $1, \ldots, i-1$ only read $O(r^{i-1} \cdot t_u^{\Sigma} / \log_r n)$ cells from epoch i. This is not enough information if $r \gg t_u^{\Sigma} / \log_r n = \Theta(t_u^{\Sigma} / t_q^{\Sigma})$, which implies $t_q^{\Sigma} = \Omega(\log_r n) = \Omega(\lg n / \lg(t_u^{\Sigma} / t_q^{\Sigma}))$.

Technical Difficulties

Nondeterminism

The lower bounds sketched above are based on the fact that the `sum` query needs to output $\Omega(\delta)$ bits of information about every query. If dealing with the *decision* `testsum` query, an argument based on output entropy can no longer work.

The most successful idea for decision queries has been to convert them to queries with non-boolean output, in an extended cell-probe model that allows nondeterminism. In this model, the query algorithm is allowed to spawn an arbitrary number of computation threads. Each thread can make t_q cell probes, after with it must either terminate with a 'reject' answer, or return an answer to the query. All nonrejecting threads must return the same output. In this model, a query with arbitrary output is equivalent to a decision query, because one can just nondeterministically guess the answer, and then verify it.

By the above, the challenge is to prove good lower bounds for `sum` even in the nondeterminstic model. Nondeterminism shakes our view that when analyzing epoch i, only cell probes to epoch i matter. The trouble is that the query may not know *which* of its probes are actually to epoch i. A probe that reads a cell from a previous epoch provides at least some information about epoch i: no update in the epoch decided to overwrite the cell. Earlier this was not a problem because the goal was only to rule out the

case that there are *zero* probes to epoch i. Now, however, different threads can probe any cell in memory, and one cannot determine which threads actually avoid probing anything in epoch i. In other words, there is a covert communication channel between epoch i and the query in which the epoch can use the choice of which cells to write in order to communicate information to the query.

There are two main strategies for handling nondeterministic query algorithms. Husfeldt and Rauhe [4] give a proof based on some interesting observations about the combinatorics of nondeterministic queries. Pătraşcu and Demaine [6] use the power of nondeterminism itself to output a small certificate that rules out useless cell probes. The latter result implies the optimal lower bound of Theorem 1 for `testsum` and, thus, the logarithmic lower bound for dynamic connectivity.

Alternative Histories

The framework described above relies on fixing all updates in epochs different from i to an average value and arguing that the query answer still has a lot of variability, depending on updates in epoch i. This is true for aggregation problems but not for search problems. If a searched item is found with equal probability in any epoch, then fixing all other epochs renders epoch i irrelevant with probability $1 - 1/(\log_r n)$.

Alstrup et al. [1] propose a very interesting refinement to the technique, proving $\Omega(\lg n/\lg\lg n)$ lower bounds for an impressive collection of search problems. Intuitively, their idea is to consider $O(\log_r n)$ alternative histories of updates, chosen independently at random. Epoch i is relevant in at least one of the histories with constant probability. On the other hand, even if one knows what epochs 1 through $i-1$ learned about epoch i in *all histories*, answering a random query is still hard.

Bit-Probe Complexity

Intuitively, if the word size is $b = 1$, the lower bound for connectivity should be roughly $\Omega(\lg^2 n)$, because a query needs $\Omega(\lg n)$ bits from every epoch. However, ruling out anything except zero probes to an epoch turns out to be difficult, for the same reason that the nondeterministic case is difficult. Without giving a very satisfactory understanding of this issue, Pătraşcu and Tarniţă [7] use a large bag of tricks to show an $\Omega((\lg n/\lg\lg n)^2)$ lower bound for dynamic connectivity. Furthermore, they consider the partial-sums problem in $\mathbb{Z}_2$ and show an $\Omega(\lg n/\lg\lg\lg n)$ lower bound, which is a triply-logarithmic factor away from the upper bound!

Applications

The lower bound discussed here extends by easy reductions to virtually all natural fully dynamic graph problems [6].

Open Problems

By far, the most important challenge for future research is to obtain a lower bound of $\omega(\lg n)$ per operation for some dynamic data structure in the cell-probe model with word size $\Theta(\lg n)$. Miltersen [5] specifies a set of technical conditions for what qualifies as a solution to such a challenge. In particular, the problem should be a *dynamic language membership* problem.

For the partial-sums problem, though `sum` is perfectly understood, `testsum` still lacks tight bounds for certain ranges of parameters [6]. In addition, obtaining tight bounds in the bit-probe model for partial sums in $\mathbb{Z}_2$ appears to be rather challenging.

Recommended Reading

1. Alstrup S, Husfeldt T, Rauhe T (1998) Marked ancestor problems. In: Proceedings of the 39th IEEE symposium on foundations of computer science (FOCS), pp 534–543
2. Eppstein D, Italiano GF, Tamassia R, Tarjan RE, Westbrook JR, Yung M (1992) Maintenance of a minimum spanning forest in a dynamic planar graph. J Algorithm 13:33–54. See also SODA'90

3. Fredman ML, Saks ME (1989) The cell probe complexity of dynamic data structures. In: Proceedings of the 21st ACM symposium on theory of computing (STOC), pp 345–354
4. Husfeldt T, Rauhe T (2003) New lower bound techniques for dynamic partial sums and related problems. SIAM J Comput 32:736–753. See also ICALP'98
5. Miltersen PB (1999) Cell probe complexity – a survey. In: 19th conference on the foundations of software technology and theoretical computer science (FSTTCS) (Advances in Data Structures Workshop)
6. Pătraşcu M, Demaine ED (2006) Logarithmic lower bounds in the cell-probe model. SIAM J Comput 35:932–963. See also SODA'04 and STOC'04
7. Pătraşcu M, Tarniţă C (2007) On dynamic bit-probe complexity. Theor Comput Sci 380:127–142. See also ICALP'05
8. Sleator DD, Tarjan RE (1983) A data structure for dynamic trees. J Comput Syst Sci 26:362–391, See also STOC'81
9. Thorup M (2000) Near-optimal fully-dynamic graph connectivity. In: Proceedings of the 32nd ACM symposium on theory of computing (STOC), pp 343–350

Lower Bounds for Online Bin Packing

Rob van Stee
University of Leicester, Leicester, UK

Keywords

Bin packing; Competitive analysis; Lower bounds; Online algorithms

Years and Authors of Summarized Original Work

1992; van Vliet
2012; Balogh, Békési, Galambos

Problem Definition

In the online bin packing problem, a sequence of *items* with sizes in the interval $(0, 1]$ arrive one by one and need to be packed into *bins*, so that each bin contains items of total size at most 1. Each item must be irrevocably assigned to a bin before the next item becomes available. The algorithm has no knowledge about future items. There is an unlimited supply of bins available, and the goal is to minimize the total number of used bins (bins that receive at least one item).

The most common performance measure for online bin packing algorithms is the asymptotic performance ratio, or asymptotic competitive ratio, which is defined as

$$R_{\mathrm{ASY}}(A) := \limsup_{n\to\infty}\left\{\max_{L}\left\{\frac{A(L)}{n}\,\middle|\,\mathrm{OPT}(L)=n\right\}\right\}. \tag{1}$$

Hence, for any input L, the number of bins used by an online algorithm A is compared to the optimal number of bins needed to pack the same input. Note that calculating the optimal number of bins might take exponential time; moreover, it requires that the entire input is known in advance.

Key Results

Yao showed that no online algorithm has performance ratio less than $\frac{3}{2}$ [7]. The following construction is very important in the context of proving lower bounds for online algorithms. Start with an item of type 1 and size $1/2 + \varepsilon$, for some very small $\varepsilon > 0$. Now, in each step, add an item of the largest possible size of the form $1/s + \varepsilon$ that can fit with all previous items into a single bin. That is, the second item has size $1/3 + \varepsilon$, the third item has size $1/7 + \varepsilon$, etc. To be more precise, it can be shown that the sizes in this input sequence are given by $1/t_i + \varepsilon$ $(i \geq 1)$, where t_i is defined by

$$t_1 = 2, \quad t_{i+1} = t_i(t_i - 1) + 1 \quad i \geq 1.$$

The first few numbers of this sequence are $2, 3, 7, 43, 1{,}807$. This sequence was first examined by Sylvester [5].

Since we allow an additive constant to the competitive ratio, in order to turn the above set of items into an input that can be used to prove a lower bound for any online algorithm, we need to repeat each item in the input N times for some arbitrarily large N. To summarize the preceding discussion, the input has the following form:

$$N \times \left(\frac{1}{2}+\varepsilon\right), N \times \left(\frac{1}{3}+\varepsilon\right), N \times \left(\frac{1}{7}+\varepsilon\right), N \times \left(\frac{1}{43}+\varepsilon\right), N \times \left(\frac{1}{1{,}807}+\varepsilon\right), \ldots, \quad (2)$$

where the items are given to the algorithm in order of *nondecreasing* size.

Brown and Liang independently gave a lower bound to 1.53635 [2, 3], using the sequence (2). Van Vliet showed how to use the input defined by (2) to prove a lower bound of 1.54014. Van Vliet set up a linear programming formulation to define all possible online algorithms for this input to prove the lower bound.

This works by characterizing online algorithms by which *patterns* they use and by how frequently they use them. A pattern is a multiset of items which fit together in one bin (have total size at most 1). As N tends to infinity, an online algorithm can be fully characterized by the fraction of bins that it packs according to each pattern.

As an example, consider the two largest item sizes in (2). The only valid patterns are $(1, 0)$, $(1, 1)$, $(0, 2)$, where (x, y) means that there are x items of size $\frac{1}{2}+\varepsilon$ and y items of size $\frac{1}{3}+\varepsilon$ in the bin. The N smallest items arrive first, and the online algorithm will pack them into bins with patterns $(1, 1)$ and $(0, 2)$ (where the first choice means that one item is now packed into it, and in the future only one item of size $\frac{1}{2}+\varepsilon$ is possibly packed with it). Say it uses x_1 bins with pattern $(1, 1)$ and x_2 with pattern $(0, 2)$; then we must have $x_1 + 2x_2 \geq N$ or $x_1/N + 2x_2/N \geq 1$. In the linear program, we use variables $x_i' = x_i/N$, thus eliminating the appearance of the number N altogether.

The input (2) appeared to be "optimal" to make life hard for online algorithms: the smallest items arrive first, and the input is constructed in such a way that each item is as large as possible given the larger items (that are defined first). Intuitively, larger items are more difficult to handle by online algorithms than smaller items. Surprisingly, however, in 2012 Balogh et al. [1] managed to prove a lower bound of $248/161 \approx 1.54037$ using a slight modification of the input. Instead of using $1/43$ as the fourth item size, they use $1/49$ and then continue the construction in the same manner as before. We get the following input:

$$N \times \left(\frac{1}{2}+\varepsilon\right), N \times \left(\frac{1}{3}+\varepsilon\right), N \times \left(\frac{1}{7}+\varepsilon\right), N \times \left(\frac{1}{49}+\varepsilon\right), N \times \left(\frac{1}{343}+\varepsilon\right), \ldots, \quad (3)$$

For the input that consists only of the first four phases of this input, the resulting lower bound is now slightly lower than before, but this is more than compensated for by the next items.

Open Problems

Other variations of the input sequence (2) do not seem to give better lower bounds. Yet there is still a clear gap to the best known upper bound of 1.58889 by Seiden [4]. Can we give a stronger lower bound using some other construction?

Cross-References

- Bin Packing with Cardinality Constraints
- Current Champion for Online Bin Packing
- Harmonic Algorithm for Online Bin Packing

Recommended Reading

1. Balogh J, Békési J, Galambos G (2012) New lower bounds for certain bin packing algorithms. Theor Comput Sci 440–441:1–13
2. Brown DJ (1979) A lower bound for on-line one-dimensional bin packing algorithms. Technical report R-864, Coordinated Science Laboratory, Urbana
3. Liang FM (1980) A lower bound for online bin packing. Inf Process Lett 10:76–79
4. Seiden SS (2002) On the online bin packing problem. J ACM 49(5):640–671
5. Sylvester JJ (1880) On a point in the theory of vulgar fractions. Am J Math 3:332–335
6. van Vliet A (1992) An improved lower bound for on-line bin packing algorithms. Inf Process Lett 43:277–284
7. Yao AC-C (1980) New algorithms for bin packing. J ACM 27:207–227

Lowest Common Ancestors in Trees

Martín Farach-Colton
Department of Computer Science,
Rutgers University, Piscataway, NJ, USA

Keywords

Least common ancestor; Nearest common ancestor; Range minimum query; Succinct structures

Years and Authors of Summarized Original Work

1984; Gabow, Bentley, Tarjan
1984; Harel, Tarjan
1989; Berkman, Breslauer, Galil, Schieber, Vishkin
2000; Bender, Farach-Colton

Problem Definition

One of the most fundamental algorithmic problems on trees is how to find the *lowest common ancestor (LCA)* of a pair of nodes. The LCA of nodes u and v in a tree is the shared ancestor of u and v that is located farthest from the root. More formally, the *lowest common ancestor (LCA)* problem is:

Preprocess: A rooted tree T having n nodes.
Query: For nodes u and v of tree T, query $\text{LCA}_T(u, v)$ returns the least common ancestor of u and v in T, that is, it returns the node farthest from the root that is an ancestor of both u and v. (When the context is clear, we drop the subscript T on the LCA.)

The goal is to optimize both the preprocessing time and the query time. We will therefore refer to the running time of an algorithm with preprocessing time $T_P(N)$ and query time of $T_Q(N)$ as having run time $\langle T_P(N), T_Q(N)\rangle$.

The LCA problem has been studied intensively both because it is inherently beautiful algorithmically and because fast algorithms for the LCA problem can be used to solve other algorithmic problems.

Key Results

In [7], Harel and Tarjan showed the surprising result that LCA queries can be answered in constant time after only linear preprocessing of the tree T. This result was simplified over the course of several papers, and current solutions are based on combinations of four themes:

1. The LCA problem is equivalent to the range minimum query (RMQ) problem, defined below, in that they can be reduced to each other in linear preprocessing time and constant query time. Thus, an optimal solution for one yields an optimal solution for the other.
2. The LCA of certain trees, notably complete binary trees and trees that are linear paths, can be computed quickly. General trees can be decomposed into special trees. Similarly, the RMQ of certain classes of arrays can be computed quickly.
3. Nodes can be labeled to capture information about their position in the tree. These labels can be used to compute the label of the LCA of two nodes.

Harel and Tarjan [7] showed that LCA computation has a lower bound of $\Omega(\log \log n)$ on a pointer machine. Therefore, the fast algorithms presented here will all require operations on $O(\log n)$-bit words. We will see how to use this assumption not only to make queries $O(1)$ time but to improve preprocessing from $O(n \log n)$ to $O(n)$ via Four Russians encoding of small problem instances.

Below, we explore each of these themes for LCA computation.

RMQ

The *range minimum query (RMQ) problem*, which seems quite different from the LCA problem, is, in fact, intimately linked. It is defined as:

Preprocess: A length n array A of numbers.

Query: For indices i and j between 1 and n, query $\text{RMQ}_A(x, y)$ returns the index of the smallest element in the subarray $A[i \ldots j]$. (When the context is clear, we drop the subscript A on the RMQ.)

The following two lemmas give linear reductions between LCA and RMQ.

Reducing LCA to RMQ

Lemma 1 ([3]) *If there is a $\langle f(n), g(n)\rangle$-time solution for RMQ, then there is a $\langle f(2n-1)+O(n), g(2n-1)+O(1)\rangle$-time solution for LCA.*

Proof Let T be the input tree. The reduction relies on one key observation:

Observation 1 *The LCA of nodes u and v is the shallowest node encountered between the visits to u and to v during a depth-first search traversal of T.*

Therefore, the reduction proceeds as follows.

1. Let array $E[1, \ldots, 2n-1]$ store the nodes visited in an Euler tour of the tree T. (The Euler tour of T is the sequence of nodes we obtain if we write down the label of each node each time it is visited during a DFS. The array of the Euler tour has length $2n-1$ because we start at the root and subsequently output a node each time we traverse an edge. We traverse each of the $n-1$ edges twice, once in each direction.) That is, $E[i]$ is the label of the ith node visited in the Euler tour of T.
2. Let the *level* of a node be its distance from the root. Compute the Level Array $L[1, \ldots, 2n-1]$, where $L[i]$ is the level of node $E[i]$ of the Euler tour.
3. Let the *representative* of a node in an Euler tour be the index of the first occurrence of the node in the tour (In fact, any occurrence of i will suffice to make the algorithm work, but we consider the first occurrence for the sake of concreteness.); formally, the representative of i is $\text{argmin}_j\{E[j] = i\}$. Compute the Representative Array $R[1, \ldots, n]$, where $R[i]$ is the index of the representative of node i.

Each of these three steps takes $O(n)$ time, yielding $O(n)$ total time. To compute $\text{LCA}_T(x, y)$, we note the following:

- The nodes in the Euler tour between the first visits to u and to v are $E[R[u], \ldots, R[v]]$ (or $E[R[v], \ldots, R[u]]$).
- The shallowest node in this subtour is at index $\text{RMQ}_L(R[u], R[v])$, since $L[i]$ stores the level of the node at $E[i]$ and the RMQ will thus report the position of the node with minimum level.
- The node at this position is $E[\text{RMQ}_L(R[u], R[v])]$, which is thus the output of $\text{LCA}_T(u, v)$, by Observation 1.

Thus, we can complete our reduction by preprocessing Level Array L for RMQ. As promised, L is an array of size $2n-1$, and building it takes time $O(n)$. The total preprocessing is $f(2n-1)+O(n)$. To calculate the query time, observe that an LCA query in this reduction uses one RMQ query in L and three array references at $O(1)$ time each, for a total of $g(2n-1)+O(1)$ time, and we have completed the proof of the reduction. ■

Reducing RMQ to LCA

Lemma 2 ([6]) *If there is a $\langle f(n), f(1)\rangle$ solution for LCA, then there is a $\langle f(n)+O(n), g(n)+O(1)\rangle$ solution for RMQ.*

Proof Let $A[1,\ldots,n]$ be the input array.

The Cartesian tree of an array is defined as follows. The root of a Cartesian tree is the minimum element of the array, and the root is labeled with the position of this minimum. Removing the root element splits the array into two pieces. The left and right children of the root are the recursively constructed Cartesian trees of the left and right subarrays, respectively.

A Cartesian tree can be built in linear time as follows. Suppose C_i is the Cartesian tree of $A[1,\ldots,i]$. To build C_{i+1}, we notice that node $i+1$ will belong to the rightmost path of C_{i+1}, so we climb up the rightmost path of C_i until finding the position where $i+1$ belongs. Each comparison either adds an element to the rightmost path or removes one, and each node can only join the rightmost path and leave it once. Thus, the total time to build C_n is $O(n)$.

The reduction is as follows.

– Let C be the Cartesian tree of A. Recall that we associate with each node in C the index i corresponding to $A[i]$.

Claim $\text{RMQ}_A(i,j) = \text{LCA}_C(i,j)$.

Proof Consider the least common ancestor, k, of i and j in the Cartesian tree C. In the recursive description of a Cartesian tree, k is the first node that separates i and j. Thus, in the array A, element $A[k]$ is between elements $A[i]$ and $A[j]$. Furthermore, $A[k]$ must be the smallest such element in the subarray $A[i,\ldots,j]$ since, otherwise, there would be a smaller element k' in $A[i,\ldots,j]$ that would be an ancestor of k in C, and i and j would already have been separated by k'.

More concisely, since k is the first element to split i and j, it is between them because it splits them, and it is minimal because it is the first element to do so. Thus, it is the RMQ. ■

We can complete our reduction by preprocessing the Cartesian tree C for LCA. Tree C takes time $O(n)$ to build, and because C is an n node tree, LCA preprocessing takes $f(n)$ time, for a total of $f(n)+O(n)$ time. The query then takes $f(n)+O(1)$, and we have completed the proof of the reduction. ■

An Algorithm for RMQ

Observe that RMQ has a solution with complexity $\langle O(n^2),\ O(1)\rangle$: build a table storing answers to all of the $\binom{n}{2}$ possible queries. To achieve $O(n^2)$ preprocessing rather than the $O(n^3)$ naive preprocessing, we apply a trivial dynamic program. Notice that answering an RMQ query now requires just one array lookup.

To improve the $\langle O(n^2),\ O(1)\rangle$-time brute-force table algorithm for RMQ to $\langle O(n\log n),\ O(1)\rangle$, precompute the result of all queries with a range size that is a power of two. That is, for every i between 1 and n and every j between 1 and $\log n$, find the minimum element in the block starting at i and having length 2^j, that is, compute $M[i,j] = \text{argmin}_{k=i\ldots i+2^j-1}\{A[k]\}$. Table M therefore has size $O(n\log n)$, and it can be filled in time $O(n\log n)$ by using dynamic programming. Specifically, find the minimum in a block of size 2^j by comparing the two minima of its two constituent blocks of size 2^{j-1}. More formally, $M[i,j] = M[i,j-1]$ if $A[M[i,j-1]] \le M[i+2^{j-1}-1, j-1]$, and $M[i,j] = M[i+2^{j-1}-1, j-1]$ otherwise.

How do we use these blocks to compute an arbitrary $\text{RMQ}(i,j)$? We select two overlapping blocks that entirely cover the subrange: let 2^k be the size of the largest block that fits into the range from i to j, that is, let $k = \lfloor\log(j-i)\rfloor$. Then $\text{RMQ}(i,j)$ can be computed by comparing the minima of the following two blocks: i to $i+2^k-1$ ($M(i,k)$) and $j-2^k+1$ to j ($M(j-2^k+1,k)$). These values have already been computed, so we can find the RMQ in constant time.

This gives the *Sparse Table (ST)* algorithm for RMQ, with complexity $\langle O(n\log n),\ O(1)\rangle$.

LCA on Special Trees and RMQ on Special Arrays

In this section, we consider special cases of LCA and RMQ that have fast solutions. These can be used to build optimal algorithms.

Paths and Balanced Binary Trees

If a tree is a path, that is, every node has outdegree 1, then computing the LCA is quite trivial. In that case, the depth of each node can be computed in

$O(n)$ time, and the LCA of two nodes is the node with smaller depth.

For a complete binary tree, the optimal algorithm is somewhat more involved. In this case, each node can be assigned a label of length $O(\log n)$ from which the LCA can be computed in constant time. This labeling idea can be extended to general trees, as we will see in section "Labeling Schemes."

Consider the following node labeling: for any node v of depth $d(v)$, the label $\mathcal{L}(v)$ is obtained by assigning to the first $d(v)$ bits of the code the left-right path from the root, where a left edge is coded with a 0 and a right edge is coded with a 1. The $d(v)+1^{\text{st}}$ bit is 1, and all subsequent bits, up to $1+\log n$ are 0.

Now let $x = \mathcal{L}(u)$ XOR $\mathcal{L}(v)$ and let $w = \text{LCA}(u, v)$. The first $d(w)$ bits of x are 0, since u and v share the same path until then. The next bit differs so the $d(w)+1$st bit of x is the first bit that is 1. Thus, by computing $\lfloor \log x \rfloor$, we find the depth of w. Then we can construct the label of w by taking the first $d(w)$ bits of $\mathcal{L}(u)$, then a 1 at position $d(w)+1$, and then 0s. All these operations take constant time, and all labels can be computed in linear time.

The first optimal LCA algorithm, by Harel and Tarjan [7], decomposed arbitrary trees into paths and balanced binary trees, using optimal labeling algorithms for each part. Such labeling schemes have been used as components in many of the subsequent algorithms. It turns out that there is an $O(\log n)$-bit labeling scheme for arbitrary trees where the LCA can be computed in constant time just from labels. This algorithm will be discussed in section "Labeling Schemes."

An $\langle O(n), O(1)\rangle$-Time Algorithm for $\pm$1RMQ

We have already seen an $\langle O(n \log n), O(1)\rangle$-time algorithm for RMQ, which thus yields an LCA algorithm of the same complexity. However, it is possible to do better, via a simple observation, plus the Four Russians technique.

Consider the RMQ problem generated by the reduction given in Lemma 1. The level tour of a tree is not an arbitrary instance of RMQ. Rather, we note that all entries are integers and adjacent entries differ by one. We call this special case $\pm$1RMQ.

If we can show an $\langle O(n), O(1)\rangle$-time algorithm for $\pm$1RMQ, we directly get an algorithm of the same complexity for LCA, by Lemma 1, but we also get an algorithm of the same complexity for general RMQ by Lemma 2. Thus, to solve an arbitrary RMQ problem optimally, first compute the Cartesian tree and then the Euler and Level tours, thus reducing an arbitrary RMQ to a $\pm$1RMQ in linear time.

In order to improve the preprocessing of $\pm$1RMQ, we will use a table lookup technique to precompute answers on small subarrays, for a log-factor speedup. To this end, partition A into blocks of size $\frac{\log n}{2}$. Define an array $A'[1, \ldots, 2n/\log n]$, where $A'[i]$ is the minimum element in the ith block of A. Define an equal size array B, where $B[i]$ is a position in the ith block in which value $A'[i]$ occurs. Recall that RMQ queries return the position of the minimum and that the LCA to RMQ reduction uses the position of the minimum, rather than the minimum itself. Thus, we will use array B to keep track of where the minima in A' came from.

The ST algorithm runs on array A' in time $\langle O(n), O(1)\rangle$. Having preprocessed A' for RMQ, consider how we answer any query $\text{RMQ}(i, j)$ in A. The indices i and j might be in the same block, so we have to preprocess each block to answer RMQ queries. If $i < j$ are in different blocks, then we can answer the query $\text{RMQ}(i, j)$ as follows. First compute the values:

1. The minimum from i forward to the end of its block
2. The minimum of all the blocks between i's block and j's block
3. The minimum from the beginning of j's block to j

The query will return the position of the minimum of the three values computed. The second minimum is found in constant time by an RMQ on A', which has been preprocessed using the ST algorithm. But we need to know how to answer range minimum queries inside blocks to compute the first and third minima and thus to finish off the

algorithm. Thus, the in-block queries are needed whether i and j are in the same block or not.

Therefore, we focus now only on in-block RMQs. If we simply performed RMQ preprocessing on each block, we would spend too much time in preprocessing. If two blocks were identical, then we could share their preprocessing. However, it is too much to hope for that blocks would be so repeated. The following observation establishes a much stronger shared-preprocessing property.

Observation 2 *If two arrays $X[1,\ldots,k]$ and $Y[1,\ldots,k]$ differ by some fixed value at each position, that is, there is a c such that $X[i] = Y[i] + c$ for every i, then all RMQ answers will be the same for X and Y. In this case, we can use the same preprocessing for both arrays.*

Thus, we can *normalize* a block by subtracting its initial offset from every element. We now use the ± 1 property to show that there are very few kinds of normalized blocks.

Lemma 3 *There are $O(\sqrt{n})$ kinds of normalized blocks.*

Proof Adjacent elements in normalized blocks differ by $+1$ or -1. Thus, normalized blocks are specified by a ± 1 vector of length $(1/2 \cdot \log n) - 1$. There are $2^{(1/2\cdot\log n)-1} = O(\sqrt{n})$ such vectors. ■

We are now basically done. We create $O(\sqrt{n})$ tables, one for each possible normalized block. In each table, we put all $(\frac{\log n}{2})^2 = O(\log^2 n)$ answers to all in-block queries. This gives a total of $O(\sqrt{n}\log^2 n)$ total preprocessing of normalized block tables and $O(1)$ query time. Finally, compute, for each block in A, which normalized block table it should use for its RMQ queries. Thus, each in-block RMQ query takes a single table lookup.

Overall, the total space and preprocessing used for normalized block tables and A' tables is $O(n)$ and the total query time is $O(1)$. This gives an optimal algorithm for LCA and RMQ. This algorithm was first presented as a PRAM algorithm by Berkman et al. [3]. Although this algorithm is quite simple, and easily implementable, for many years after its publication, LCA computation was still considered to be too complicated to implement. The algorithm presented here is somewhat streamlined compared to Berkman et al.'s algorithm, because they had the added goal of making a parallelizable algorithm. Thus, for example, they used two levels of encoding to remove the log factor from the preprocessing, with the first level breaking the RMQ into blocks of size $\log n$ and in the second level breaking the blocks up into mini-blocks of size $\log\log n$. Similarly, the sparse table algorithm was somewhat different and required binary-tree LCA as a subroutine.

It is possible, even probably, that the slight complexities of the PRAM version of this algorithm obscured its elegance. This theory was tested by Bender and Farach-Colton [2], who presented the sequential version of the same algorithm with the simplified Sparse Table and RMQ blocking scheme presented here, with the goal of establishing the practicality of LCA computation. This seems to have done the trick, and many variants and implementations of this algorithm now exist.

Labeling Schemes

We have already seen a labeling scheme that allows for the fast computation of LCA on complete binary trees. But labels can be used to solve the LCA problem on arbitrary trees, as shown by Alstrup et al. [1].

To be specific, the goal is to assign to every node an $O(\log n)$-bit label so that the label of the LCA of two nodes can be computed in constant time from their labels. We have seen how to do this for a complete binary tree, but the problem there was simplified by the fact that the depth of such a tree is $O(\log n)$ and the branching factor is two. Here, we consider the general case of arbitrary depth and degree.

Begin by decomposing the tree into *heavy paths*. To do so, let the weight $w(v)$ of any node v be the number of nodes in the subtree rooted at that node. All edges between a node and its heaviest child are called *heavy edges* and all other edges are called *light edges*.

Ties are broken arbitrarily. Note that there are $O(\log n)$ light edges on any root-leaf path.

A path to a node can be specified by alternately specifying how far one traverses a heavy path before exiting to a light edge and the rank of the light edge at the point of exit. Each such code takes $O(\log n)$ bits yielding $O(\log^2 n)$ bits in total.

To reduce the number of bits to $O(\log n)$, Alstrup et al. applied alphabetic codes, which preserve lexicographic ordering but are of variable length. In particular, they used a code with the following properties:

- Let $Y = y_1, y_2, \ldots, y_k$ be a sequence of integers with $\sum_{i=1}^{k} y_i = s$.
- There exists an alphabetic sequence $B = b_1, b_2, \ldots, b_k$ for Y such that, for all i, $|b_i| \leq \lceil \log s \rceil - \lfloor \log y_i \rfloor$.

The idea now is that large trees get short codes and small trees get large codes. By cleverly building alphabetic codes that depend on the size of the trees, the code lengths telescope, giving a final code of length $O(\log n)$. We refer the reader to the paper or to the excellent presentation by Bille [4] for details.

Succinct Representations

There have been several extensions of the LCA/RMQ problem. The one that has received the most attention is that of succinct structures to compute the LCA or RMQ. These structures are succinct because they use $O(n)$ bits of extra space, as opposed to the structures above, which use $\Omega(n)$ words of memory, or $\Omega(n \log n)$ bits.

The first succinct solution for LCA was due to Sadakane [10], who gave an optimal LCA algorithm using $2n + O(n(\log \log n)^2 / \log n)$ bits. The main approach of this algorithm is to replace the Sparse Table algorithm with a more bit-efficient variant.

The first succinct solution for RMQ was also due to Sadakane [9]. His algorithm takes $4n + o(n)$ bits. The main idea, once again, follows the Sparse Table algorithm. By using a ternary Cartesian tree which stores values not in internal nodes but in leaves, the preorders of nodes coincide the orders of the values.

The current best solution is by Fischer [5], who uses $2n + o(n)$ bits, which is shown to be optimal up to lower-order terms. The structure using the least known $o(n)$-bit term [8] uses $2n + O(n/\log^c n)$ bits, for any constant c. All the succinct solutions are $O(1)$ time.

Cross-References

▸ Compressed Range Minimum Queries

Recommended Reading

1. Alstrup S, Gavoille C, Kaplan H, Rauhe T (2004) Nearest common ancestors: a survey and a new algorithm for a distributed environment. Theory Comput Syst 37(3):441–456
2. Bender MA, Farach-Colton M (2000) The LCA problem revisited. In: Proceedings of Latin American theoretical informatics (LATIN), Montevideo, pp 88–94
3. Berkman O, Breslauer D, Galil Z, Schieber B, Vishkin U (1989) Highly parallelizable problems. In: Proceedings of the 21st annual ACM symposium on theory of computing, New Orleans, pp 309–319
4. Bille P (2014) Nearest common ancestors. http://massivedatasets.files.wordpress.com/2014/02/nearestcommonancestors_2014.pdf
5. Fischer J (2010) Optimal succinctness for range minimum queries. In: Proceedings of LATIN, Oaxaca, pp 158–169
6. Gabow HN, Bentley JL, Tarjan RE (1984) Scaling and related techniques for geometry problems. In: Proceedings of the 16th annual ACM symposium on theory of computing, New York, vol 67, pp 135–143
7. Harel D, Tarjan RE (1984) Fast algorithms for finding nearest common ancestors. SIAM J Comput 13(2):338–355
8. Navarro G, Sadakane K (2014) Fully functional static and dynamic succinct trees. ACM Trans Algorithms 10(3)
9. Sadakane K (2002) Space-efficient data structures for flexible text retrieval systems. In: International symposium on algorithms and computation (ISAAC), Vancouver, pp 14–24
10. Sadakane K (2002) Succinct representations of lcp information and improvements in the compressed suffix arrays. In: Proceedings of the 13th annual ACM-SIAM symposium on discrete algorithms, San Francisco, pp 225–232

LP Based Parameterized Algorithms

M.S. Ramanujan
Department of Informatics, University of Bergen, Bergen, Norway

Keywords

Above guarantee parameterizations; FPT algorithms; Linear programming

Years and Authors of Summarized Original Work

2013; Cygan, Pilipczuk, Pilipczuk, Wojtaszczyk
2014; Lokshtanov, Narayanaswamy, Raman, Ramanujan, Saurabh
2014; Wahlstrom

Problem Definition

Linear and integer programs have played a crucial role in the theory of approximation algorithms for combinatorial optimization problems. While they have also been central in identifying polynomial time solvable problems, it is only recently that these tools have been put to use in designing exact algorithms for NP-complete problems. Following the paradigm of above-guarantee parameterization in fixed-parameter tractability, these efforts have focused on designing algorithms where the exponential component of the running time depends only on the excess of the solution above the optimum value of a linear program for the problem.

Method Description

The linear program obtained from a given integer linear program (ILP) by relaxing the integrality conditions on the variables is called the *standard relaxation* of the ILP or the standard LP. Similarly, the linear program obtained from the ILP by restricting the domain of the variables to the set of all half integers is called the *half-integral relaxation* of the ILP or the half-integral LP. The standard LP is said to have a half-integral optimum if the optimum values of the standard relaxation and half-integral relaxation coincide.

The VERTEX COVER problem provides one of the simplest illustrations of this method. In this problem, the objective is to find a minimum-sized subset of vertices whose removal makes a given graph edgeless. In the well-known integer linear programming formulation (ILP) for VERTEX COVER, given a graph G, a *feasible solution* is defined as a function $x : V \to \{0, 1\}$ satisfying the edge constraints $x(u) + x(v) \geq 1$ for every edge (u, v). The objective of the linear program is to minimize $\Sigma_{u \in V} x(u)$ over all feasible solutions x. The value of the optimum solution to this ILP is denoted by $vc(G)$. In the standard relaxation of the above ILP, the constraint $x(v) \in \{0, 1\}$ is replaced with $x(v) \geq 0$, for all $v \in V$. For a graph G, this relaxation is denoted by LPVC(G), and the minimum value of LPVC(G) is denoted by $vc^*(G)$.

It is known that LPVC(G) has a half-integral optimum [10] and that LPVC(G) is *persistent* [11], that is, if a variable is assigned 0 (respectively 1) in an optimum solution to the standard LP, then it can be discarded from (respectively included into) an optimum vertex cover of G. Based on the persistence of LPVC(G), a polynomial time preprocessing procedure for VERTEX COVER immediately follows. More precisely, as long as there is an optimum solution to the standard LP which assigns 0 or 1 to a vertex of G, one may discard or include this vertex in the optimum solution. When this procedure cannot be executed any longer, an arbitrary vertex of G is selected, and the algorithm branches into 2 exhaustive cases based on this vertex being included or excluded in an optimum vertex cover of G. Standard analysis shows that this is an algorithm running in time $O(4^{(vc(G)-vc^*(G))}|G|^{O(1)})$.

Key Results

This method was first used in the context of fixed-parameter tractability by Guillemot [5]

who used it to give FPT algorithms for path-transversal and cycle-transversal problems. Subsequently, Cygan et al. [4] improved upon this result to give an FPT algorithm for the MULTIWAY CUT problem parameterized above the LP value. In this problem, the objective is to find a smallest set of vertices which pair-wise separates a given subset of vertices of a graph. As a consequence of this algorithm, they were able to obtain the current fastest FPT algorithm for MULTIWAY CUT parameterized by the solution size.

Theorem 1 ([4]) *There is an algorithm for* MULTIWAY CUT *running in time* $O(2^k|G|^{O(1)})$*, where* k *is the size of the solution.*

Following this work, Narayanaswamy et al. [9] and Lokshtanov et al. [8] considered the VERTEX COVER problem and built upon these methods with several additional problem-specific steps to obtain improved FPT algorithms for several problems, with the most notable among them being the ODD CYCLE TRANSVERSAL problem – the problem of finding a smallest set of vertices to delete in order to obtain a bipartite graph. These results were the first improvements over the very first FPT algorithm for this problem given by Reed, Smith, and Vetta [12].

Theorem 2 ([8]) *There is an algorithm for* ODD CYCLE TRANSVERSAL *running in time* $O(2.32^k|G|^{O(1)})$*, where* k *is the size of the solution.*

Iwata et al. [7] applied this method to several problems to improve the polynomial dependence of the running times on the input size. Using network-flow-based *linear time* algorithms for solving the half-integral Vertex Cover LP (LPVC), they obtained an FPT algorithm for ODD CYCLE TRANSVERSAL with a linear dependence on the input size. Most recently, using tools from the theory of constraint satisfaction, Wahlstrom [13] extended this approach to a much broader class of problems with half-integral LPs and obtained improved FPT algorithms for a number of problems including node-deletion UNIQUE LABEL COVER and GROUP FEEDBACK VERTEX SET. The UNIQUE LABEL COVER problem plays a central role in the theory of approximation and was studied from the point of view of parameterized complexity by Chitnis et al. [2]. The GROUP FEEDBACK VERTEX SET problem is a generalization of the classical FEEDBACK VERTEX SET problem. The fixed-parameter tractability of this problem was proved in [5] and [3].

Theorem 3 ([2]) *There is an algorithm for node-deletion* UNIQUE LABEL COVER *running in time* $O(|\Sigma|^{2k}|G|^{O(1)})$ *and an algorithm for* GROUP FEEDBACK VERTEX *running in time* $O(4^k|G|^{O(1)})$*. In the first case,* Σ *denotes the size of the alphabet, and in either case,* k *denotes the size of the solution.*

Applications

This method relies crucially on the half integrality of a certain LP for the problem at hand. The most well-known problems with this property are VERTEX COVER, MULTIWAY CUT, and certain problems for which Hochbaum [6] defined a particular kind of ILPs, referred to as IP2. However, the work of Wahlstrom [13] lifts this approach to a more general class of problems by interpreting a half-integral relaxation as a polynomial-time solvable problem on the discrete search space of $\{0, \frac{1}{2}, 1\}$.

Open Problems

A primary challenge here is to build upon these LP-based tools to design an FPT algorithm for ODD CYCLE TRANSVERSAL with a provably optimal dependence on the parameter under appropriate complexity theoretic assumptions.

Experimental Results

Experimental results comparing algorithms for VERTEX COVER based on this method with other state-of-the art empirical methods are given in [1].

Cross-References

- ▸ Kernelization, Max-Cut Above Tight Bounds
- ▸ Kernelization, MaxLin Above Average

Recommended Reading

1. Akiba T, Iwata Y (2015) Branch-and-reduce exponential/FPT algorithms in practice: a case study of vertex cover. In: Proceedings of the seventeenth workshop on algorithm engineering and experiments, ALENEX 2015, San Diego, 5 Jan 2015, pp 70–81
2. Chitnis RH, Cygan M, Hajiaghayi M, Pilipczuk M, Pilipczuk M (2012) Designing FPT algorithms for cut problems using randomized contractions. In: 53rd annual IEEE symposium on foundations of computer science, FOCS 2012, New Brunswick, 20–23 Oct 2012, pp 460–469
3. Cygan M, Pilipczuk M, Pilipczuk M (2012) On group feedback vertex set parameterized by the size of the cutset. In: Graph-theoretic concepts in computer science – 38th international workshop, WG 2012, Jerusalem, 26–28 June 2012, revised selected papers, pp 194–205
4. Cygan M, Pilipczuk M, Pilipczuk M, Wojtaszczyk JO (2013) On multiway cut parameterized above lower bounds. Trans Comput Theory 5(1):3
5. Guillemot S (2011) FPT algorithms for path-transversal and cycle-transversal problems. Discret Optim 8(1):61–71
6. Hochbaum DS (2002) Solving integer programs over monotone inequalities in three variables: a framework for half integrality and good approximations. Eur J Oper Res 140(2):291–321
7. Iwata Y, Oka K, Yoshida Y (2014) Linear-time FPT algorithms via network flow. In: Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms, SODA 2014, Portland, 5–7 Jan 2014, pp 1749–1761
8. Lokshtanov D, Narayanaswamy NS, Raman V, Ramanujan MS, Saurabh S (2014) Faster parameterized algorithms using linear programming. ACM Trans Algorithms 11(2):15
9. Narayanaswamy NS, Raman V, Ramanujan MS, Saurabh S (2012) LP can be a cure for parameterized problems. In: 29th international symposium on theoretical aspects of computer science, STACS 2012, Paris, 29th Feb – 3rd Mar 2012, pp 338–349
10. Nemhauser GL, Trotter LE (1974) Properties of vertex packing and independence system polyhedra. Math Program 6:48–61
11. Nemhauser GL, Trotter LE (1975) Vertex packings: structural properties and algorithms. Math Program 8:232–248
12. Reed BA, Smith K, Vetta A (2004) Finding odd cycle transversals. Oper Res Lett 32(4):299–301
13. Wahlström M (2014) Half-integrality, LP-branching and FPT algorithms. In: Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms, SODA 2014, Portland, 5–7 Jan 2014, pp 1762–1781

LP Decoding

Jonathan Feldman
Google, Inc., New York, NY, USA

Keywords

Belief propagation; Error-correcting codes; LDPC codes; Low-density parity-check codes; LP decoding; Pseudocodewords

Years and Authors of Summarized Original Work

2002 and later; Feldman, Karger, Wainwright

Problem Definition

Error-correcting codes are fundamental tools used to transmit digital information over unreliable channels. Their study goes back to the work of Hamming and Shannon, who used them as the basis for the field of information theory. The problem of decoding the original information up to the full error-correcting potential of the system is often very complex, especially for modern codes that approach the theoretical limits of the communication channel.

LP decoding [4, 5, 8] refers to the application of *linear programming* (LP) *relaxation* to the problem of decoding an error-correcting code. Linear programming relaxation is a standard technique in approximation algorithms and operations research, and is central to the study of efficient algorithms to find good (albeit suboptimal) solutions to very difficult optimization problems [13]. LP decoders have tight combinatorial

characterizations of decoding success that can be used to analyze error-correcting performance.

The codes for which LP decoding has received the most attention are *low-density parity-check* (LDPC) codes [9], due to their excellent error-correcting performance. The LP decoder is particularly attractive for analysis of these codes because the standard message-passing algorithms such as *belief propagation* (see [15]) used for decoding are often difficult to analyze, and indeed the performance of LP decoding is closely tied to these methods.

Error-Correcting Codes and Maximum-Likelihood Decoding

This section begins with a very brief overview of error-correcting codes, sufficient for formulating the LP decoder. Some terms are not defined for space reasons; for a full treatment of error-correcting codes in context, the reader is referred to textbooks on the subject (e.g., [11]).

A *binary error-correcting code* is a subset $C \subseteq \{0,1\}^n$. The *rate* of the code C is $r = \log(|C|)/n$. A *linear* binary code is a linear subspace of $\{0,1\}^n$. A *codeword* is a vector $y \in C$. Note that 0^n is always a codeword of a linear code, a fact that will be useful later. When the code is used for communication, a codeword $\dot{y} \in C$ is transmitted over a *noisy channel*, resulting in some *received word* $\hat{y} \in \Sigma^n$, where Σ is some alphabet that depends on the channel model. Generally in LP decoding a *memoryless, symmetric* channel is assumed. One common such channel is the *binary symmetric channel (BSC) with parameter p*, which will be referred to as BSC_p, where $0 < p < 1/2$. In the BSC_p, the alphabet is $\Sigma = \{0,1\}$, and for each i, the received symbol $\hat{y}_i$ is equal to $\dot{y}_i$ with probability p, and $\hat{y}_i = 1 - \dot{y}_i$ otherwise. Although LP decoding works with more general channels, this chapter will focus on the BSC_p.

The *maximum-likelihood (ML) decoding problem* is the following: given a received word $\hat{y} \in \{0,1\}^n$, find the codeword $y^* \in C$ that is most likely to have been sent over the channel. Defining the vector $\gamma \in \{-1,+1\}^n$ where $\gamma_i = 1 - 2\hat{y}_i$, it is easy to show:

$$y^* = \arg\min_{y \in C} \sum_i \gamma_i y_i . \tag{1}$$

The complexity of the ML decoding problem depends heavily on the code being used. For simple codes such as a *repetition code* $C = \{0^n, 1^n\}$, the task is easy. For more complex (and higher-rate) codes such as LDPC codes, ML decoding is NP-hard [1].

LP Decoding

Since ML decoding can be very hard in general, one turns to sub-optimal solutions that can be found efficiently. LP decoding, instead of trying to solve (1), relaxes the constraint $y \in C$, and instead requires that $y \in \mathcal{P}$ for some succinctly describable linear polytope $\mathcal{P} \subseteq [0,1]^n$, resulting in the following linear program:

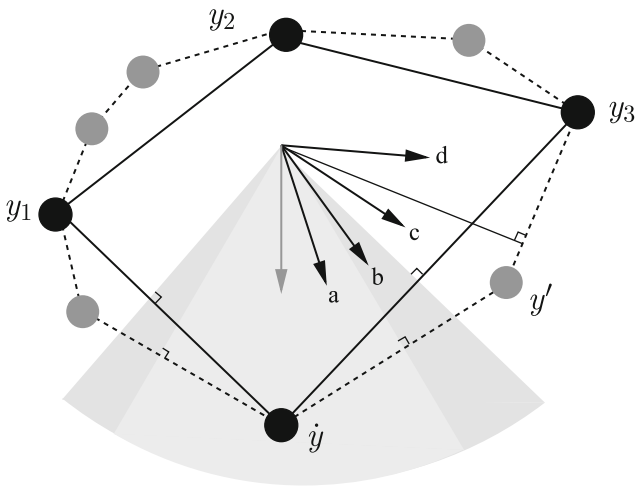
$$y_{\mathrm{LP}} = \arg\min_{y \in \mathcal{P}} \sum_{i=1}^{n} \gamma_i y_i . \tag{2}$$

It should be the case that the polytope includes all the codewords, and does not include any integral non-codewords. As such, a polytope $\mathcal{P}$ is called *proper* for code C if $\mathcal{P} \cap \{0,1\}^n = C$.

The LP decoder works as follows. Solve the LP in (2) to obtain $y_{\mathrm{LP}} \in [0,1]^n$. If y_{LP} is integral (i.e., all elements are 0 or 1), then output y_{LP}. Otherwise, output "error". By the definition of a proper polytope, if the LP decoder outputs a codeword, it is guaranteed to be equal to the ML codeword y^*. This fact is known as the *ML certificate* property.

Comparing with ML Decoding

A successful decoder is one that outputs the original codeword transmitted over the channel, and so the quality of an algorithm is measured by the likelihood that this happens. (Another common non-probabilistic measure is the *worst-case* performance guarantee, which measures how many bit-flips an algorithm can tolerate and still be guaranteed to succeed.) Note that y^* is the one *most likely* to be the transmitted codeword $\dot{y}$, but it is not always the case that $y^* = \dot{y}$. However, no



LP Decoding, Fig. 1 A decoding polytope $\mathcal{P}$ (*dotted line*) and the convex hull C (*solid line*) of the codewords $\dot{y}$, y_1, y_2, and y_3. Also shown are the four possible cases (a–d) for the objective function, and the normal cones to both $\mathcal{P}$ and C

decoder can perform better than an ML decoder, and so it is useful to use ML decoding as a basis for comparison.

Figure 1 provides a geometric perspective of LP decoding, and its relation to exact ML decoding. Both decoders use the same LP objective function, but over different constraint sets. In exact ML decoding, the constraint set is the convex hull C of codewords (i.e., the set of points that are convex combinations of codewords from C), whereas relaxed LP decoding uses the larger polytope $\mathcal{P}$. In Fig. 1, the four arrows labeled (a)–(d) correspond to different "noisy" versions of the LP objective function. (a) If there is very little noise, then the objective function points to the transmitted codeword $\dot{y}$, and thus both ML decoding and LP decoding succeed, since both have the transmitted codeword $\dot{y}$ as the optimal point. (b) If more noise is introduced, then ML decoding succeeds, but LP decoding fails, since the fractional vertex y' is optimal for the relaxation. (c) With still more noise, ML decoding fails, since y_3 is now optimal; LP decoding still has a fractional optimum y', so this error is in some sense "detected". (d) Finally, with a lot of noise, both ML decoding and LP decoding have y_3 as the optimum, and so both methods fail and the error is "undetected". Note that in the last two cases (c, d), when ML decoding fails, the failure of the LP decoder is in some sense the fault of the code itself, as opposed to the decoder.

Normal Cones and C-Symmetry

The (negative) *normal cones* at $\dot{y}$ (also called the *fundamental cone* [10]) is defined as follows:

$$N_{\dot{y}}(\mathcal{P}) = \left\{\gamma \in \mathbb{R}^n : \sum_i \gamma_i (y_i - \dot{y}_i) \geq 0 \text{ for all } y \in \mathcal{P}\right\},$$

$$N_{\dot{y}}(C) = \left\{\gamma \in \mathbb{R}^n : \sum_i \gamma_i (y_i - \dot{y}_i) \geq 0 \text{ for all } y \in C\right\}.$$

Note that $N_{\dot{y}}(\mathcal{P})$ corresponds to the set of cost vectors γ such that $\dot{y}$ is an optimal solution to (2). The set $N_{\dot{y}}(C)$ has a similar interpretation as the set of cost vectors γ for which $\dot{y}$ is the ML codeword. Since $\mathcal{P} \subset C$, it is immediate from the definition that $N_y(C) \supset N_y(\mathcal{P})$ for all $y \in C$. Fig. 1 shows these two cones and their relationship.

The success probability of an LP decoder is equal to the total probability mass of $N_{\dot{y}}(\mathcal{P})$, under the distribution on cost vectors defined by the channel. The success probability of ML decoding is similarly related to the probability mass in the normal cone $N_y(C)$. Thus, the discrepancy between the normal cones of $\mathcal{P}$ and C is a measure of the gap between exact ML and relaxed LP decoding.

This analysis is specific to a particular transmitted codeword $\dot{y}$, but one would like to apply it in general. When dealing with linear codes, for most decoders one can usually assume that an arbitrary codeword is transmitted, since the de-

cision region for decoding success is symmetric. The same holds true for LP decoding (see [4] for proof), as long as the polytope $\mathcal{P}$ is *C-symmetric*, defined as follows:

Definition 1 A proper polytope $\mathcal{P}$ for the binary code C is C-**symmetric** if, for all $y \in \mathcal{P}$ and $\dot{y} \in C$, it holds that $y' \in \mathcal{P}$, where $y'_i = |y_i - \dot{y}_i|$.

Using a Dual Witness to Prove Error Bounds

In order to prove that LP decoding succeeds, one must show that $\dot{y}$ is the optimal solution to the LP in (2). If the code C is linear, and the relaxation is proper and C-symmetric, one can assume that $\dot{y} = 0^n$, and then show that 0^n is optimal. Consider the *dual* of the decoding LP in (2). If there is a feasible point of the dual LP that has the same cost (i.e., zero) as the point 0^n has in the primal, then 0^n must be an optimal point of the decoding LP. Therefore, to prove that the LP decoder succeeds, it suffices to exhibit a zero-cost point in the dual. (Actually, since the existence of the zero-cost dual point only proves that 0^n is one of possibly many primal optima, one needs to be a bit more careful, a minor issue deferred to more complete treatments of this material.)

Key Results

LP decoders have mainly been studied in the context of Low-Density Parity-Check codes [9], and their generalization to expander codes [12]. LP decoders for Turbo codes [2] have also been defined, but the results are not as strong. This summary of key results gives bounds on the *word error rate (WER)*, which is the probability, over the noise in the channel, that the decoder does not output the transmitted word. These bounds are relative to specific *families* of codes, which are defined as infinite set of codes of increasing length whose rate is bounded from below by some constant. Here the bounds are given in asymptotic form (without constants instantiated), and only for the binary symmetric channel.

Many other important results that are not listed here are known for LP decoding and related notions. Some of these general areas are surveyed in the next section, but there is insufficient space to reference most of them individually; the reader is referred to [3] for a thorough bibliography.

Low-Density Parity-Check Codes

The polytope $\mathcal{P}$ for LDPC codes, first defined in [4, 8, 10], is based on the underlying *Tanner graph* of the code, and has a linear number of variables and constraints. If the Tanner graph expands sufficiently, it is known that LP decoding can correct a constant fraction of errors in the channel, and thus has an inverse exponential error rate. This was proved using a dual witness:

Theorem 1 ([6]) *For any rate $r > 0$, there is a constant $\epsilon > 0$ such that there exists a rate r family of low-density parity-check codes with length n where the LP decoder succeeds as long as at most ϵn bits are flipped by the channel. This implies that there exists a constant $\epsilon' > 0$ such that the word error rate under the* BSC_p *with $p < \epsilon'$ is at most $2^{-\Omega(n)}$.*

Expander Codes

The *capacity* of a communication channel bounds from above the rate one can obtain from a family of codes and still get a word error rate that goes to zero as the code length increases. The notation C_p is used to denote the capacity of the BSC_p. Using a family of codes based on expanders [12], LP decoding can achieve rates that approach capacity. Compared to LDPC codes, however, this comes at the cost of increased decoding complexity, as the size of the LP is exponential in the gap between the rate and capacity.

Theorem 2 ([7]) *For any $p > 0$, and any rate $r < C_p$, there exists a rate r family of expander codes with length n such that the word error rate of LP decoding under the* BSC_p *is at most $2^{-\Omega(n)}$.*

Turbo Codes

Turbo codes [2] have the advantage that they can be encoded in linear time, even in a streaming

fashion. *Repeat-accumulate* codes are a simple form of Turbo code. The LP decoder for Turbo codes and their variants was first defined in [4, 5], and is based on the *trellis* structure of the component *convolutional* codes. Due to certain properties of turbo codes it is impossible to prove bounds for turbo codes as strong as the ones for LDPC codes, but the following is known:

Theorem 3 ([5]) *There exists a rate* $1/2 - o(1)$ *family of repeat-accumulate codes with length n, and a constant* $\epsilon > 0$*, such that under the* BSC_p *with* $p < \epsilon$*, the LP decoder has a word error rate of at most* $n^{-\Omega(1)}$.

Applications

The application of LP decoding that has received the most attention so far is for LDPC codes. The LP for this family of codes not only serves as an interesting alternative to more conventional iterative methods [15], but also gives a useful tool for analyzing those methods, an idea first explored in [8, 10, 14]. Iterative methods such as *belief propagation* use local computations on the Tanner graph to update approximations of the marginal probabilities of each code bit. In this type of analysis, the vertices of the polytope $\mathcal{P}$ are referred to as *pseudocodewords*, and tend to coincide with the fixed points of this iterative process. Other notions of pseudocodeword-like structures such as *stopping sets* are also known to coincide with these polytope vertices. Understanding these structures has also inspired the design of new codes for use with iterative and LP decoding. (See [3] for a more complete bibliography of this work).

The decoding method itself can be extended in many ways. By adding redundant information to the description of the code, one can derive tighter constraint sets to improve the error-correcting performance of the decoder, albeit at an increase in complexity. Adaptive algorithms that try to add constraints "on the fly" have also been explored, using branch-and-bound or other techniques. Also, LP decoding has inspired the use of other methods from optimization theory in decoding error-correcting codes. (Again, see [3] for references.)

Open Problems

The LP decoding method gives a simple, efficient and analytically tractable approach to decoding error-correcting codes. The results known to this point serve as a proof of concept that strong bounds are possible, but there are still important questions to answer. Although LP decoders can achieve capacity with decoding time polynomial in the length of the code, the complexity of the decoder still depends exponentially on the gap between the rate and capacity (as is the case for all other known provably efficient capacity-achieving decoders). Decreasing this dependence would be a major accomplishment, and perhaps LP decoding could help. Improving the fraction of errors correctable by LP decoding is also an important direction for further research.

Another interesting question is whether there exist constant-rate linear-distance code families for which one can formulate a polynomial-sized exact decoding LP. Put another way, is there a constant-rate linear-distance family of codes whose convex hulls have a polynomial number of facets? If so, then LP decoding would be equivalent to ML decoding for this family. If not, this is strong evidence that suboptimal decoding is inevitable when using good codes, which is a common belief.

An advantage to LP decoding is the *ML certificate* property mentioned earlier, which is not enjoyed by most other standard suboptimal decoders. This property opens up the possibility for a wide range of heuristics for improving decoding performance, some of which have been analyzed, but largely remain wide open.

LP decoding has (for the most part) only been explored for LDPC codes under memoryless symmetric channels. The LP for turbo codes has been defined, but the error bounds proved so far are not a satisfying explanation of the excellent performance observed in practice. Other codes and channels have gotten little, if any, attention.

Cross-References

- Decoding Reed–Solomon Codes
- Learning Heavy Fourier Coefficients of Boolean Functions
- Linearity Testing/Testing Hadamard Codes
- List Decoding near Capacity: Folded RS Codes

Recommended Reading

1. Berlekamp E, McEliece R, van Tilborg H (1978) On the inherent intractability of certain coding problems. IEEE Trans Inf Theory 24:384–386
2. Berrou C, Glavieux A, Thitimajshima P (1993) Near Shannon limit error-correcting coding and decoding: turbo-codes. In: Proceedings of the IEEE international conference on communications (ICC), Geneva, 23–26 May 1993, pp 1064–1070
3. Boston N, Ganesan A, Koetter R, Pazos S, Vontobel P Papers on pseudocodewords. HP Labs, Palo Alto. http://www.pseudocodewords.info
4. Feldman J (2003) Decoding error-correcting codes via linear programming. Ph.D. thesis, Massachusetts Institute of Technology
5. Feldman J, Karger DR (2002) Decoding turbo-like codes via linear programming. In: Proceedings of the 43rd annual IEEE symposium on foundations of computer science (FOCS), Vancouver, 16–19 Nov 2002
6. Feldman J, Malkin T, Servedio RA, Stein C, Wainwright MJ (2004) LP decoding corrects a constant fraction of errors. In: Proceedings of the IEEE international symposium on information theory, Chicago, 27 June – 2 July 2004
7. Feldman J, Stein C (2005) LP decoding achieves capacity. In: Symposium on discrete algorithms (SODA '05), Vancouver, Jan 2005
8. Feldman J, Wainwright MJ, Karger DR (2003) Using linear programming to decode linear codes. In: 37th annual conference on information sciences and systems (CISS '03), Baltimore, 12–14 Mar 2003
9. Gallager R (1962) Low-density parity-check codes. IRE Trans Inf Theory (IT) 8:21–28
10. Koetter R, Vontobel P (2003) Graph covers and iterative decoding of finite-length codes. In: Proceedings of the 3rd international symposium on turbo codes and related topics, Brest, Sept 2003, pp 75–82
11. MacWilliams FJ, Sloane NJA (1981) The theory of error correcting codes. North-Holland, Amsterdam
12. Sipser M, Spielman D (1996) Expander codes. IEEE Trans Inf Theory 42:1710–1722
13. Vazirani VV (2003) Approximation algorithms. Springer, Berlin
14. Wainwright M, Jordan M (2003) Variational inference in graphical models: the view from the marginal polytope. In: Proceedings of the 41st Allerton conference on communications, control, and computing, Monticello Oct 2003
15. Wiberg N (1996) Codes and decoding on general graphs. Ph.D. thesis, Linkoping University

M

Majority Equilibrium

Qizhi Fang
School of Mathematical Sciences, Ocean University of China, Qingdao, Shandong Province, China

Keywords

Condorcet winner

Years and Authors of Summarized Original Work

2003; Chen, Deng, Fang, Tian

Problem Definition

Majority rule is arguably the best decision mechanism for public decision-making, which is employed not only in public management but also in business management. The concept of majority equilibrium captures such a democratic spirit in requiring that no other solutions would please more than half of the voters in comparison to it. The work of Chen, Deng, Fang, and Tian [1] considers a public facility location problem decided via a voting process under the majority rule on a discrete network. This work distinguishes itself from previous work by applying the computational complexity approach to the study of majority equilibrium. For the model with a single public facility located in trees, cycles, and cactus graphs, it is shown that the majority equilibrium can be found in linear time. On the other hand, when the number of public facilities is taken as the input size (not a constant), finding a majority equilibrium is shown to be $\mathcal{NP}$-hard.

Consider a network $G = ((V, \omega), (E, l))$ with vertex and edge- weight functions $\omega : V \to R^+$ and $l : E \to R^+$, respectively. Each vertex $i \in V$ represents a community, and $\omega(i)$ represents the number of voters that reside there. For each $e \in E$, $l(e) > 0$ represents the length of the road $e = (i, j)$ connecting two communities i and j. For two vertices $i, j \in V$, the distance between i and j, denoted by $d_G(i, j)$, is the length of a shortest path joining them. The location of a public facility such as a library, community center, etc., is to be determined by the public via a voting process under the majority rule. Here, each member of the community desires to have the public facility close to himself, and the decision has to be agreed upon by a majority of the voters. Denote the vertex set of G by $V = \{v_1, v_2, \ldots, v_n\}$. Then each $v_i \in V$ has a preference order $\succeq_i$ on V induced by the distance on G. That is, $x \succeq_i y$ if and only if $d_G(v_i, x) \leq d_G(v_i, y)$ for two vertices $x, y \in V$; similarly, $x \succ_i y$ if and only if $d_G(v_i, x) < d_G(v_i, y)$. Based on such a preference profile, four types of majority equilibrium, called Condorcet winners, are defined as follows.

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

Definition 1 Let $v_0 \in V$, then v_0 is called:

1. *A weak quasi-Condorcet winner*, if for every $u \in V$ distinct of v_0,

$$\omega(\{v_i \in V : v_0 \succeq_i u\}) \geq \sum_{v_i \in V} \omega(v_i)/2;$$

2. *A strong quasi-Condorcet winner*, if for every $u \in V$ distinct of v_0,

$$\omega(\{v_i \in V : v_0 \succeq_i u\}) > \sum_{v_i \in V} \omega(v_i)/2;$$

3. *A weak Condorcet winner*, if for every $u \in V$ distinct of v_0,

$$\omega(\{v_i \in V : v_0 \underset{i}{\succeq} u\}) \geq \omega(\{v_i \in V : u \underset{i}{\succeq} v_0\});$$

4. *A strong Condorcet winner*, if for every $u \in V$ distinct of v_0,

$$\omega(\{v_i \in V : v_0 \underset{i}{\succeq} u\}) > \omega(\{v_i \in V : u \underset{i}{\succeq} v_0\}).$$

Under the majority voting mechanism described above, the problem is to develop efficient ways for determining the existence of Condorcet winners and finding such a winner when one exists.

Problem 1 (Finding Condorcet Winners) INPUT: A network $G = ((V, w), (E, l))$. OUTPUT: A Condorcet winner $v \in V$ or nonexistence of Condorcet winners.

Key Results

The mathematical results on the Condorcet winners depend deeply on the understanding of combinatorial structures of underlying networks. Theorems 1–3 below are given for weak quasi-Condorcet winners in the model with a single facility to be located. Other kinds of Condorcet winners can be discussed similarly.

Theorem 1 *Every tree has one weak quasi-Condorcet winner or two adjacent weak quasi-Condorcet winners, which can be found in linear time.*

Theorem 2 *Let C_n be a cycle of order n with vertex-weight function $\omega : V(C_n) \to R^+$. Then $v \in V(C_n)$ is a weak quasi-Condorcet winner of C_n if and only if the weight of each $\lfloor \frac{n+1}{2} \rfloor$-interval containing v is at least $\frac{1}{2} \sum_{v \in C_n} \omega(v)$. Furthermore, the problem of finding a weak quasi-Condorcet winner of C_n is solvable in linear time.*

Given a graph $G = (V, E)$, a vertex v of G is a *cut vertex* if $E(G)$ can be partitioned into two nonempty subsets E_1 and E_2 such that the induced graphs $G[E_1]$ and $G[E_2]$ have just the vertex v in common. A *block* of G is a connected subgraph of G that has no cut vertices and is maximal with respect to this property. Every graph is the union of its blocks. A graph G is called a *cactus graph*, if G is a connected graph in which each block is an edge or a cycle.

Theorem 3 *The problem of finding a weak quasi-Condorcet winner of a cactus graph with vertex-weight function is solvable in linear time.*

In general, the problem can be extended to the cases where a number of public facilities are required to be located during one voting process, and the definitions of Condorcet winners can also be extended accordingly. In such cases, the public facilities may be of the same type or different types; and the utility functions of the voters may be of different forms.

Theorem 4 *If there are a bounded constant number of public facilities to be located at one voting process under the majority rule, then the problem of finding a Condorcet winner (any of the four types) can be solved in polynomial time.*

Theorem 5 *If the number of public facilities to be located is not a constant but considered as the input size, the problem of finding a Condorcet winner is $\mathcal{NP}$-hard; and the corresponding decision problem: deciding whether a candidate set of public facilities is a Condorcet winner is co-$\mathcal{NP}$-complete.*

Applications

Damange [2] first reviewed continuous and discrete spatial models of collective choice, aiming at characterizing the public facility location problem as a result of the pubic voting process. Although the network models in Chen et al. [1] have been studied for some problems in economics [3,4], the main point of Chen et al.'s work is the computational complexity and algorithmic approach. This approach can be applied to more general public decision-making processes.

For example, consider a public road repair problem, pioneered by Tullock [5] to study redistribution of tax revenue under a majority rule system. An edge-weighted graph $G = (V, E, w)$ represents a network of local roads, where the weight of each edge represents the cost of repairing the road. There is also a distinguished vertex $s \in V$ representing the entry point to the highway system. The majority decision problem involves a set of agents $A \subseteq V$ situated at vertices of the network who would choose a subset F of edges. The cost of repairing F, which is the sum of the weights of edges in F, will be shared by all n agents, each an n-th of the total. In this model, a majority stable solution under the majority rule is a subset $F \subseteq E$ that connects s to a subset $A_1 \subset A$ of agents with $|A_1| > |A|/2$ such that no other solution H connecting s to a subset of agents $A_2 \subset A$ with $|A_2|A|/2$ satisfies the conditions that $\sum_{e \in H} w(e) \leq \sum_{e \in F} w(e)$, and for each agent in A_2, its shortest path to s in solution H is not longer than that in solution F, and at least one of the inequalities is strict. It is shown in Chen et al. [1] that for this model, finding a majority equilibrium is $\mathcal{NP}$-hard for general networks and is polynomially solvable for tree networks.

Cross-References

- ▸ General Equilibrium
- ▸ Leontief Economy Equilibrium
- ▸ Local Search for K-medians and Facility Location

Recommended Reading

1. Chen L, Deng X, Fang Q, Tian F (2002) Majority equilibrium for public facility allocation. Lect Notes Comput Sci 2697:435–444
2. Demange G (1983) Spatial models of collective choice. In: Thisse JF, Zoller HG (eds) Locational analysis of public facilities. North-Holland Publishing Company, Amsterdam
3. Hansen P, Thisse JF (1981) Outcomes of voting and planning: condorcet, weber and rawls locations. J Publ Econ 16:1–15
4. Schummer J, Vohra RV (2002) Strategy-proof location on a network. J Econ Theory 104:405–428
5. Tullock G (1959) Some problems of majority voting. J Polit Econ 67:571–579

Manifold Reconstruction

Siu-Wing Cheng
Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China

Keywords

Čzech complex; Delaunay complex; Homology; Homeomorphism; Implicit function; Voronoi diagram

Years and Authors of Summarized Original Work

2005; Cheng, Dey, Ramos
2008; Niyogi, Smale, Weinberger
2014; Boissonnat, Ghosh
2014; Cheng, Chiu

Problem Definition

With the widespread of sensing and Internet technologies, a large number of numeric attributes for a physical or cyber phenomenon can now be collected. If each attribute is viewed as a coordinate, an instance in the collection can be viewed as a point in $\mathbb{R}^d$ for some large d. When the physical or cyber phenomenon is governed by

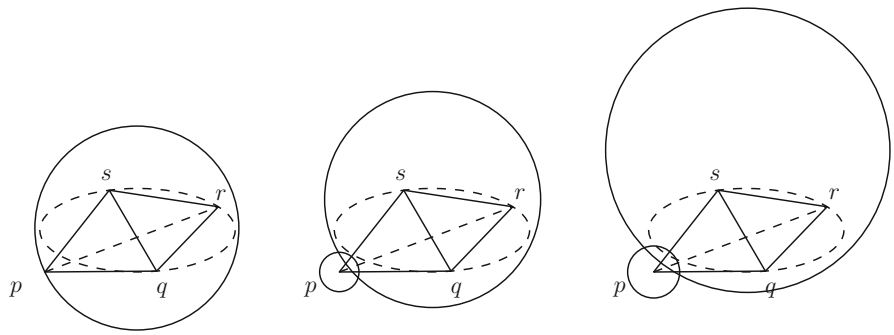
only a few latent parameters, it is often postulated that the data points lie on some unknown smooth compact manifold $\mathcal{M}$ of dimension k, where $k \ll d$. The goal is to reconstruct a faithful representation of $\mathcal{M}$ from the data points. Reconstruction problem are ill-posed in general. Therefore, the data points are assumed to be dense enough so that it becomes theoretically possible to obtain a faithful reconstruction. The quality of the reconstruction is measured in several ways: the Hausdorff distance between the reconstruction and $\mathcal{M}$, the deviation between the normal spaces of the reconstruction and $\mathcal{M}$ at nearby points, and whether the reconstruction and $\mathcal{M}$ are topologically equivalent.

Key Results

It is clear that more data points are needed in some parts of $\mathcal{M}$ than others, and this is captured well by the concepts of *medial axis* and *local feature size*. The medial axis of $\mathcal{M}$ is the closure of the set of points in $\mathbb{R}^d$ that are at the same distances from two or more closest points in $\mathcal{M}$. For every point $x \in \mathcal{M}$, its local feature $f(x)$ is the distance from x to the medial axis of $\mathcal{M}$. The input set S of data points in $\mathcal{M}$ is an *ϵ-sample* if for every point $x \in \mathcal{M}$, $d(x, S) \leq \epsilon f(x)$. The input set S is a *uniform* ϵ-sample if for every point $x \in \mathcal{M}$, $d(x, S) \leq \varepsilon$, assuming that the ambient space has been scaled such that $\min_{x \in \mathcal{M}} f(x) = 1$. Furthermore, for every pair of points $p, q \in S$, if $d(p, q) \geq \delta f(p)$ or $d(p, q) \geq \delta$ for some constant δ, then we call S an (ϵ, δ)-sample or a uniform (ϵ, δ)-sample, respectively. Most reconstruction results in the literature are about what theoretical guarantees can be offered when ϵ is sufficiently small.

Cocone Complex

Cheng, Dey, and Ramos [7] gave the first proof that a faithful homeomorphic simplicial complex can be constructed from the data points. First, the dimension k of $\mathcal{M}$ and the tangent spaces at the data points are estimated using known algorithms in the literature (e.g., [4, 8, 10]). Let θ be some appropriately small constant angle. The *θ-cocone* at a point $p \in S$ is the subset of points $z \in \mathbb{R}^d$ such that pz makes an angle at most θ with the estimated tangent space at p. Consider the Delaunay triangulation of S. Let σ be a Delaunay simplex and let V_σ be its dual Voronoi cell. The *cocone complex* $\mathcal{K}$ of S is the collection of Delaunay simplices σ such that V_σ intersects the θ-cocones of the vertices of σ. It turns out that if the simplices in the Delaunay triangulation of S have bounded aspect ratio, then $\mathcal{K}$ is a k-dimensional simplicial complex homeomorphic to $\mathcal{M}$. Moreover, the Hausdorff distance between $\mathcal{K}$ and $\mathcal{M}$ is at most ε times the local feature size, and the angle deviation between the normal spaces of $\mathcal{K}$ and $\mathcal{M}$ at nearby points is $O(\epsilon)$. A difficult step of the algorithm is the generalization of sliver removal [6] in $\mathbb{R}^3$ to $\mathbb{R}^d$ in order to ensure that the simplices have bounded aspect ratio. The intuition is simple, but the analysis is quite involved. Figure 1 shows a sliver $pqrs$. View the Delaunay triangulation as a weighted Delaunay triangulation with all weights



Manifold Reconstruction, Fig. 1 As the weight of p increases, the orthoball becomes larger and moves away from p

equal to zero. If the weight of p is increased, the orthoball of $pqrs$ – the ball at zero power distances from all its vertices – moves away from p and becomes larger. Therefore, the ball will become so large that it contains some other points in S, thereby eliminating the sliver $pqrs$ from the weighted Delaunay triangulation.

Theorem 1 ([7]) *There exist constants ϵ and δ such that if S is an (ϵ, δ)-sample of a smooth compact manifold of dimension k in $\mathbb{R}^d$, a simplicial complex $\mathcal{K}$ can be constructed such that:*

- *$\mathcal{K}$ is homeomorphic to $\mathcal{M}$.*
- *Let τ be a j-dimensional simplex of $\mathcal{K}$. Let p be a vertex of τ. For every point $q \in \mathcal{M}$, if $d(p,q) = O(\epsilon f(p))$, then for every normal vector $\mathbf{n}$ at q, τ has a normal vector $\mathbf{v}$ that makes an $O(\epsilon)$ angle with $\mathbf{n}$.*
- *For every point x in $\mathcal{K}$, its distance from the nearest point $y \in \mathcal{M}$ is $O(\epsilon f(y))$.*

The running time of the algorithm is exponential in d.

Boissonnat, Guibas, and Oudot [3] show that the Voronoi computation can be avoided if one switches to the *weighted witness complex*. Moreover, given an ϵ-sample without the lower bound on the interpoint distances, they can construct a family of plausible reconstructions of dimensions $1, 2, \ldots, k$ and let the user choose an appropriate one. The sliver issue is also encountered [3] and resolved in an analogous manner.

Čech Complex

Betti numbers are informative topological invariants of the shape. There are $d + 1$ betti numbers, β_i for $i \in [0, d]$. The zeroth betti number β_0 is the number of connected components in $\mathcal{M}$. In three dimensions (i.e., $d = 3$), β_2 is the number of voids in $\mathcal{M}$ (i.e., bounded components in $\mathbb{R}^3 \setminus \mathcal{M}$), and β_1 is the number of independent one-dimensional cycles in $\mathcal{M}$ that cannot be contracted within $\mathcal{M}$ to a single point. Two cycles are *homologous* if one can be deformed into the other continuously. Two overlapping cycles can be combined by eliminating the overlap. A set of cycles are independent if no cycle can be obtained by continuously deforming and combining some other cycles in the set. A tunnel is physically accommodated in the complement of $\mathcal{M}$, and its existence is witnessed by a one-dimensional cycle in $\mathcal{M}$ that "circles around" the tunnel (i.e., cannot be contracted in $\mathcal{M}$ to a single point). Therefore, the number of independent one-dimensional cycles that cannot be contracted to a single point measures the number of "independent tunnels." In fact, voids live in the complement of $\mathcal{M}$ too and their number is measured by the number of independent 2-dimensional cycles that cannot be contracted to a single point. In general, β_i is the number of independent i-dimensional cycles that cannot be contracted in $\mathcal{M}$ to a single point. Alternatively, β_i is the rank of the ith homology group. Note that $\beta_i = 0$ for $i > k$ as $\mathcal{M}$ is k-dimensional.

Numerical procedures are known to compute the betti numbers of a simplicial complex (e.g., [11]) and only the incidence relations among the elements in the complex are needed. Therefore, given a homeomorphic simplicial complex $\mathcal{K}$ of $\mathcal{M}$, the betti numbers of $\mathcal{K}$ and hence of $\mathcal{M}$ can be computed. In fact, $\mathcal{K}$ and $\mathcal{M}$ have the same homology (groups). But requiring a homeomorphic complex is an overkill. Let $\mathcal{B}$ be a set of balls of the same radius r. For every subset of balls in $\mathcal{B}$, connect the ball centers in the subset to form a simplex if these balls have a nonempty common intersection. The resulting collection of simplices is known as a *Čech complex* $\mathcal{C}$. Niyogi, Smale, and Weinberger [12] proved that if S is a dense sample from a uniform probability distribution on $\mathcal{M}$ and r is set appropriately, then $\mathcal{C}$ has the same homology as $\mathcal{M}$ with high probability.

Theorem 2 ([12]) *Let S be a set of n points drawn in i.i.d. fashion according to the uniform probability measure on $\mathcal{M}$. Assume that $r < 1/2$. There exists constants α_1 and α_2 depending on $\mathcal{M}$ such that if $n > \alpha_1(\log \alpha_2 + \log(1/\delta))$, then $\mathcal{C}$ has the same homology as $\mathcal{M}$ with probability greater than $1 - \delta$.*

In general the Čech complex contains many simplices. Niyogi, Smale, and Weinberger

showed that the same result also holds when the probability distribution has support near $\mathcal{M}$ instead of exactly on $\mathcal{M}$. Subsequently, similar results have also been obtained for the *Vietoris-Rips complex* [1].

Tangential Delaunay Complex

Although the cocone complex gives a homeomorphic reconstruction, the running time is exponential in d. It is natural to ask whether the running time can be made to depend exponentially on k instead. Boissonnat and Ghosh [2] answered this question affirmatively by introducing a new local Delaunay reconstruction.

Let S be a dense (ϵ, δ)-sample of $\mathcal{M}$. Suppose that the tangent spaces at the data points in S have been estimated with an $O(\epsilon)$ angular error. Take a point $p \in S$. Let H_p be the estimated tangent space at p. Let V_p be the Voronoi cell owned by p. Identify the set star(p) of Delaunay simplices that are incident to p and whose dual Voronoi faces intersect H_p. The collection of all such stars form the *tangential Delaunay complex*. The bisectors between p and the other points in S intersect H_p in $(k-1)$-dimensional affine subspaces. These affine subspaces define a Voronoi diagram in H_p, and star(p) is determined by the cell owned by p in this Voronoi diagram in H_p. Therefore, no data structure of dimension higher than k is needed in the computation.

The tangential Delaunay complex is not a triangulated manifold in general though due to some inconsistencies. For example, if σ is a simplex in star(p), it is not necessarily true that σ is in star(q) for another vertex q of σ. Such inconsistencies can be removed by assigning weights to the points in S appropriately as in the case of eliminating slivers from the cocone complex.

Theorem 3 ([2]) *The guarantees in Theorem 1 can be obtained by an algorithm that runs in $O(dn^2 + d2^{O(k^2)}n)$ time, where n is the number of data points.*

More theoretical guarantees are provided in [2].

Implicit Function

The complexes in the previous methods are either very large or not so easy to compute in practice. An alternative approach is to approximate $\mathcal{M}$ by the zero-set of an implicit function $\varphi : \mathbb{R}^d \to \mathbb{R}^{d-k}$ that is defined using the data points in S, assuming that S is a uniform ϵ-sample:

$$\varphi(x) = \sum_{p \in S} \omega(x, p) \cdot B^t_{\varphi,x} \cdot (x - p),$$

- Let $c \geq 3$. Define $\omega(x, p) = \gamma(d(x, p))/\sum_{p \in S} \gamma(d(x, q))$, where

$$\gamma(s) = \begin{cases} \left(1 - \frac{s}{kc\epsilon}\right)^{2k} \left(\frac{2s}{c\epsilon} + 1\right), & \text{if } s \in [0, kc\epsilon], \\ 0, & \text{if } s > kc\epsilon. \end{cases}$$

  Notice that $\varphi(x)$ depends only on the points in S within a distance $kc\epsilon$ from x.
- For every point $p \in S$, let T_p be a $d \times k$ matrix with orthonormal columns such that its column space is an approximate tangent space at p with angular error $O(\epsilon)$. For every point $x \in \mathbb{R}^d$, let $C_x = \sum_{p \in S} \omega(x, p) \cdot T_p T_p^t$. Therefore, the space L_x spanned by the eigenvectors of C_x corresponding to the smallest $d - k$ eigenvalues is a "weighted average" of the approximate normal spaces at the data points. Define $B_{\varphi,x}$ to be a $d \times (d-k)$ matrix with linearly independent columns such that its column space is L_x. It turns out the zero-set of φ is independent of the choices of $B_{\varphi,x}$.

The weight function ω makes local reconstruction possible without a complete sampling of $\mathcal{M}$. Moreover, the construction of φ is computational less intensive than the construction of a complex. The following guarantees are offered.

Theorem 4 ([5]) *Let $\hat{\mathcal{M}}$ be the set of points at distance ϵ^τ or less from $\mathcal{M}$ for any fixed $\tau \in (1, 2)$. Let S_φ denote the zero-set of φ. Let ν denote the map that sends a point in $\mathbb{R}^d$ to the nearest point in $\mathcal{M}$.*

- *For a small enough ϵ, the restriction of ν to $S_\varphi \cap \hat{\mathcal{M}}$ is a homeomorphism between $S_\varphi \cap \hat{\mathcal{M}}$ and $\mathcal{M}$.*

- *For every $x \in S_\varphi \cap \hat{\mathcal{M}}$, the angle between the normal space of $\mathcal{M}$ at $v(x)$ and the normal space of S_φ at x is $O(\epsilon^{(\tau-1)/2})$.*

A provably good iterative projection operator is also known for φ [9].

Cross-References

- ▸ Curve Reconstruction
- ▸ Delaunay Triangulation and Randomized Constructions
- ▸ Surface Reconstruction
- ▸ Voronoi Diagrams and Delaunay Triangulations

Recommended Reading

1. Attali D, Lieutier A, Salinas D (2013) Vietoris-Rips complexes also provide topologically correct reconstructions of sampled shapes. Comput Geom Theory Appl 46(4):448–465
2. Boissonnat J-D, Ghost A (2014) Manifold reconstruction using tangential Delaunay complexes. Discret Comput Geom 51(1):221–267
3. Boissonnat J-D, Guibas LJ, Oudot S (2009) Manifold reconstruction in arbitrary dimensions using witness complexes. Discret Comput Geom 42(1):37–70
4. Cheng S-W, Chiu, M-K (2009) Dimension detection via slivers. In: Proceedings of the ACM-SIAM symposium on discrete algorithms, New York, pp 1001–1010
5. Cheng S-W, Chiu M-K (2014) Implicit manifold reconstruction. In: Proceedings of the ACM-SIAM symposium on discrete algorithms, Portland, pp 161–173
6. Cheng S-W, Dey TK, Edelsbrunner H, Facello MA, Teng S-H (2000) Sliver exudation. J ACM 17(1–2):51–68
7. Cheng S-W, Dey TK, Ramos EA (2005) Manifold reconstruction from point samples. In: Proceedings of the ACM-SIAM symposium on discrete algorithms, Vancouver, pp 1018–1027
8. Cheng S-W, Wang Y, Wu Z (2008) Provable dimension detection using principal component analysis. Int J Comput Geom Appl 18(5):415–440
9. Chiu M-K (2013) Manifold reconstruction from discrete point sets. PhD dissertation, Hong Kong University of Science and Technology
10. Dey TK, Giesen J, Goswami S, Zhao W (2003) Shape dimension and approximation from samples. Discret Comput Geom 29(3):419–343
11. Friedman J (1998) Computing Betti numbers via combinatorial laplacians. Algorithmica 21(4): 331–346
12. Niyogi P, Smale S, Weinberger S (2008) Finding the homology of submanifolds with high confidence from random samples. Discret Comput Geom 39(1):419–441

Market Games and Content Distribution

Vahab S. Mirrokni
Theory Group, Microsoft Research, Redmond, WA, USA

Keywords

Congestion games; Market sharing games; Stable matching; Valid-Utility games

Years and Authors of Summarized Original Work

2005; Mirrokni

Problem Definition

This chapter studies market games for their performance and convergence of the equilibrium points. The main application is the content distribution in cellular networks in which a service provider needs to provide data to users. The service provider can use several cache locations to store and provide the data. The assumption is that cache locations are selfish agents (resident subscribers) who want to maximize their own profit. Most of the results apply to a general framework of monotone two-sided markets.

Uncoordinated Two-Sided Markets

Various economic interactions can be modeled as two-sided markets. A two-sided market consists of two disjoint groups of agents: active agents and passive agents. Each agent has a preference list over the agents of the other side, and can be matched to one (or many) of the agents in the other side. A central solution concept to

these markets are *stable matchings*, introduced by Gale and Shapley [5]. It is well known that stable matchings can be achieved using a centralized polynomial-time algorithm. Many markets, however, do not have any centralized matching mechanism to match agents. In those markets, matchings are formed by actions of self-interested agents. Knuth [9] introduced uncoordinated two-sided markets. In these markets, cycles of better or best responses exist, but random better response and best response dynamics converge to a stable matching with probability one [2, 10, 14]. Our model for content distribution corresponds to a special class of uncoordinated two-sided markets that is called *the distributed caching games*.

Before introducing the distributed caching game as an uncoordinated two-sided market, the distributed caching problem and some game theoretic notations are defined.

Distributed Caching Problem

Let U be a set of n cache locations with given available capacities A_i and given available bandwidths B_i for each cache location i. There are k request types; (Request type can be thought of as different files that should be delivered to clients.) each request type t has a size a_t $(1 \leq t \leq k)$. Let H be a set of m requests with a reward R_j, a required bandwidth b_j, a request type t_j for each request j, and a cost c_{ij} for connecting each cache location i to each request j. The profit of providing request j by cache location i is $f_{ij} = R_j - c_{ij}$. A cache location i can service a set of requests S_i, if it satisfies the *bandwidth constraint*: $\sum_{j \in S_i} b_j \leq B_i$, and the *capacity constraint*: $\sum_{t \in \{t_j | j \in S_i\}} a_t \leq A_i$ (this means that the sum of the sizes of the request types of the requests in cache location i should be less than or equal to the available capacity of cache location i). A set S_i of requests is feasible for cache location i if it satisfies both of these constraints. The goal of the DCP problem is to find a feasible assignment of requests to cache locations to maximize the total profit; i.e., the total reward of requests that are provided minus the connection costs of these requests.

Strategic Games

A *strategic game* $\mathcal{G}$ is defined as a tuple $\mathcal{G}(U, \{F_i | i \in U\}, \{\alpha_i() | i \in U\})$ where (i) U is the set of n players or agents, (ii) F_i is a family of feasible *(pure) strategies* or *actions* for player i and (iii) $\alpha_i : \Pi_{i \in U} F_i \rightarrow \mathbb{R}^+ \cup \{0\}$ is the (private) *payoff* or *utility* function for agent i, given the set of strategies of all players. Player i's strategy is denoted by $s_i \in F_i$. A *strategy profile* or a (strategy) *state*, denoted by $S = (s_1, s_2, \ldots, s_n)$, is a vector of strategies of players. Also let $S \oplus s'_i := (s_1, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_k)$.

Best-Response Moves

In a *non-cooperative* game, each agent wishes to maximize its own payoff. For a strategy profile $S = (s_1, s_2, \ldots, s_n)$, a *better response move* of player i is a strategy s'_i such that $\alpha_i(S \oplus s'_i) \geq \alpha_i(S)$. In a *strict better response move*, the above inequality is strict. Also, for a strategy profile $S = (s_1, s_2, \ldots, s_n)$ a *best response* of player i in S is a better response move $s^*_i \in F_i$ such that for any strategy $s_i \in F_i$, $\alpha_i(S \oplus s^*_i) \geq \alpha_i(S \oplus s_i)$.

Nash Equilibria

A pure strategy Nash equilibrium (PSNE) of a strategic game is a strategy profile in which each player plays his best response.

State Graph

The state graph, $\mathcal{D} = (\mathcal{F}, \mathcal{E})$, of a strategic game $\mathcal{G}$, is an arc-labeled directed graph, where the vertex set $\mathcal{F}$ corresponds to the set of strategy profiles or states in $\mathcal{G}$, and there is an arc from state S to state S' with label i if the only difference between S and S' is in the strategy of player i; and player i plays one of his best responses in strategy profile S'. A *best-response walk* is a directed walk in the state graph.

Price of Anarchy

Given a strategic game, $\mathcal{G}(U, \{F_i | i \in U\}, \{\alpha() | i \in U\})$, and a maximization social function $\gamma : \Pi_{i \in U} F_i \rightarrow \mathbb{R}$, the price of anarchy, denoted by $\mathsf{poa}(\mathcal{G}, \gamma)$, is the worst ratio between the

social value of a pure Nash equilibrium and the optimum.

Distributed Caching Games

The distributed caching game can be formalized as a two-sided market game: active agents correspond to n resident subscribers or cache locations, and passive agents correspond to m requests from transit subscribers. Formally, given an instance of the DCP problem, a strategic game $\mathcal{G}(U, \{F_i | i \in U\}, \{\alpha_i | i \in U\})$ is defined as follows. The set of players (or active agents) U is the set of cache locations. The family of feasible strategies F_i of a cache location i is the family of subsets s_i of requests such that $\sum_{j \in s_i} b_j \leq B_i$ and $\sum_{t \in \{t_j | j \in s_i\}} a_t \leq A_i$. Given a vector $S = (s_1, s_2, \ldots, s_n)$ of strategies of cache locations, the *favorite* cache locations for request j, denoted by $\mathsf{FAV}(j)$, is the set of cache locations i such that $j \in s_i$ and f_{ij} has the maximum profit among the cache locations that have request j in their strategy set, i.e., $f_{ij} \geq f_{i'j}$ for any i' such that $j \in s_{i'}$. For a strategy profile $S = (s_1, \ldots, s_n)$ $\alpha_i(S) = \sum_{j: i \in \mathsf{FAV}(j)} f_{ij} / |\mathsf{FAV}(j)|$. Intuitively, the above definition implies that the profit of each request goes to the cache locations with the minimum connection cost (or equivalently with the maximum profit) among the set of cache locations that provide this request. If more than one cache location have the maximum profit (or minimum connection cost) for a request j, the profit of this request is divided equally between these cache locations. The payoff of a cache location is the sum of profits from the requests it actually serves. A player i *serves* a request j if $i \in \mathsf{FAV}(j)$. The social value of strategy profile S, denoted by $\gamma(S)$, is the sum of profits of all players. This value $\gamma(S)$ is a measure of the efficiency of the assignment of requests and request types to cache locations.

Special Cases

In this paper, the following variants and special cases of the DCP problem are also studied: The CapDCP problem is a special case of DCP problem without bandwidth constraints. The BanDCP problem is a special case of DCP problem without capacity constraints. In the uniform BanDCP problem, the bandwidth consumption of all requests is the same. In the uniform CapDC problem, the size of all request types is the same.

Many-to-One Two-Sided Markets with Ties

In the distributed caching game, active and passive agents correspond to cache locations and requests respectively. The set of feasible strategies for each active agent correspond to a set of solutions to a packing problem. Moreover, the preferences of both active and passive agents is determined from the profit of requests to cache locations. In many-to-one two-sided markets, the preference of passive and active agents as well as the feasible family of strategies are arbitrary. The preference list of agents may have ties as well.

Monotone and Matroid Markets

In *monotone many-to-one two-sided markets*, the preferences of both active and passive agents are determined based on payoffs $p_{ij} = p_{ji}$ for each active agent i and passive agent j (similar to the DCP game). An agent i prefers j to j' if $p_{ij} > p_{ij'}$. In *matroid two-sided markets*, the feasible set of strategies of each active agent is the set of independent sets of a matroid. Therefore, uniform BanDCP game is a matroid two-sided market game.

Key Results

In this section, the known results for these problems are summarized.

Centralized Approximation Algorithm

The distributed caching problem generalizes the multiple knapsack problem and the generalized assignment problem [3] and as a result is an APX-hard problem.

Theorem 1 ([4]) *There exists a linear programming based* $1 - \frac{1}{e}$*-approximation algorithm and a local search* $\frac{1}{2}$*-approximation algorithm for the* DCP *problem.*

The $1 - \frac{1}{e}$-approximation for this problem is based on rounding an exponentially large configuration linear program [4]. On the basis of some reasonable complexity theoretic assumptions, this approximation factor of $1 - \frac{1}{e}$ is tight for this problem. More formally,

Theorem 2 ([4]) *For any $\epsilon > 0$, there exists no $1 - \frac{1}{e} - \epsilon$-approximation algorithm for the* DCP *problem unless $NP \subseteq DTIME(n^{O(\log \log n)})$.*

Price of Anarchy

Since the DCP game is a strategic game, it possesses mixed Nash equilibria [12]. The DCP game is a valid-utility game with a submodular social function as defined by Vetta [16]. This implies that the performance of any mixed Nash equilibrium of this game is at least $\frac{1}{2}$ of the optimal solution.

Theorem 3 ([4, 11]) *The* DCP *game is a valid-utility game and the price of anarchy for mixed Nash equilibria is $\frac{1}{2}$. Moreover, this result holds for all monotone many-to-one two-sided markets with ties.*

A direct proof of the above price of anarchy bound for the DCP game can be found in [11].

Pure Nash Equilibria: Existence and Convergence

This part surveys known results for existence and convergence of pure Nash equilibria.

Theorem 4 ([11]) *There are instances of the* IBDC *game that have no pure Nash equilibrium.*

Since, IBDC is a special case of CapDCP, the above theorem implies that there are instances of the CapDCP game that have no pure Nash equilibrium. In the above theorem, the bandwidth consumption of requests are not uniform, and this was essential in finding the example. The following gives theorems for the uniform variant of these games.

Theorem 5 ([1, 11]) *Any instance of the uniform* BanDCP *game does not contain any cycle of strict best-response moves, and thus possess a pure Nash equilibrium. On the other hand, there are instances of the uniform* CapDCP *game with no pure Nash equilibria.*

The above result for the uniform BanDCP game can be generalized to matroid two-sided markets with ties as follows.

Theorem 6 ([1]) *Any instance of the monotone matroid two-sided market game with ties is a potential game, and possess pure Nash equilibria. Moreover, any instance of the matroid two-sided market game with ties possess pure Nash equilibria.*

Convergence Time to Equilibria

This section proves that there are instances of the uniform CapDCP game in which finding a pure Nash equilibrium is PLS-hard [8]. The definition of PLS-hard problems can be found in papers by Yannakakis et al. [8, 15].

Theorem 7 ([11]) *There are instances of the uniform* CapDCP *game with pure Nash equilibria (It is also possible to say that finding a* sink equilibrium *is PLS-hard. A sink equilibrium is a set of strategy profiles that is closed under best-response moves. A pure equilibrium is a sink equilibrium with exactly one profile. This equilibrium concept is formally defined in [7].) for which finding a pure Nash equilibrium is PLS-hard.*

Using the above proof and a result of Schaffer and Yannakakis [13, 15], it is possible to show that in some instances of the uniform CapDCP game, there are states from which all paths of best responses have exponential length.

Corollary 1 ([11]) *There are instances of the uniform* CapDCP *game that have pure Nash equilibria with states from which any sequence of best-response moves to any pure Nash equilibrium (or sink equilibrium) has an exponential length.*

The above theorems show exponential convergence to pure Nash equilibria in general DCP

games. For the special case of the uniform BanDCP game, the following is a positive result for the convergence time to equilibria.

Theorem 8 ([2]) *The expected convergence time of a random best-response walk to pure Nash equilibria in matroid monotone two-sided markets (without ties) is polynomial.*

Since the uniform BanDCP game is a special case of matroid monotone two-sided markets with ties, the above theorem indicates that for the BanDCP game with no tie in the profit of requests, the convergence time of a random best-response walk is polynomial. Finally, we state a theorem about the convergence time of the general (non-monotone) matroid two-sided market games.

Theorem 9 ([2]) *In the matroid two-sided markets (without ties), a random best response dynamic of players may cycle, but it converges to a Nash equilibrium with probability one. However, it may take exponential time to converge to a pure Nash equilibrium.*

Pure Nash equilibria of two-sided market games correspond to stable matchings in two-sided markets and vice-versa [2]. The fact that better response dynamics of players in two-sided market games may cycle, but will converge to a stable matching has been proved in [9, 14]. Ackermann et al. [2] extend these results for best-response dynamics, and show an exponential lower bound for expected convergence time to pure Nash equilibria.

Applications

The growth of the Internet, the World Wide Web, and wide-area wireless networks allow an increasing number of users to access vast amounts of information in different geographic areas. As one of the most important functions of the service provider, content delivery can be performed by caching popular items in cache locations close to the users. Performing such a task in a decentralized manner in the presence of self-interested entities in the system can be modeled as an uncoordinated two-sided market game.

The 3G subscriber market can be categorized into groups with shared interest in location-based services, e.g., the preview of movies in a theater or scenes of the mountain nearby. Since the 3G radio resources are limited, it is expensive to repeatedly transmit large quantities of data over the air interface from the base station (BS). It is more economical for the service provider to offload such repeated requests on to the ad-hoc network comprised of its subscribers where some of them recently acquired a copy of the data. In this scenario, the goal for the service provider is to give incentives for peer subscribers in the system to cache and forward the data to the requesting subscribers. Since each data item is large in size and transit subscribers are mobile, we assume that the data transfer occurs in a close range of a few hops.

In this setting, envision a system consisting of two groups of subscribers: resident and transit subscribers. Resident subscribers are less mobile and mostly confined to a certain geographical area. Resident subscribers have incentives to cache data items that are specific to this geographical region since the service provider gives monetary rewards for satisfying the queries of transit subscribers. Transit subscribers request their favorite data items when they visit a particular region. Since the service provider does not have knowledge of the spatial and temporal distribution of requests, it is difficult if not impossible for the provider to stipulate which subscriber should cache which set of data items. Therefore, the decision of what to cache is left to each individual subscriber. The realization of this content distribution system depends on two main issues. First, since subscribers are selfish agents, they may act to increase their individual payoff and decrease the performance of the system. Here, we provide a framework for which we can prove that in an equilibrium situation of this framework, we use the performance of the system efficiently. The second issue is that the payoff of each request for each agent must be a function of the set of agents that have this request in their strategy, since these agents compete on this request and

the profit of this request should be divided among these agents in an appropriate way. Therefore, each selfish agent may change the set of items it cached in response to the set of items cached by others. This model leads to a non-cooperative caching scenario that can be modeled on a two-sided market game, studied and motivated in the context of market sharing games and distributed caching games [4, 6, 11].

Open Problems

It is known that there exist instances of the distributed caching game with no pure Nash equilibria. It is also known that best response dynamics of players may take exponential time to converge to pure Nash equilibria. An interesting question is to study the performance of sink equilibria [7, 11] or the price of sinking [7, 11] for these games. The distributed caching game is a valid-utility game. Goemans, Mirrokni, and Vetta [7] show that despite the price of anarchy of $\frac{1}{2}$ for valid-utility games, the performance of sink equilibria (or price of sinking) for these games is $\frac{1}{n}$. We conjecture that the price of sinking for DCP games is a constant. Moreover, it is interesting to show that after polynomial rounds of best responses of players the approximation factor of the solution is a constant. We know that one round of best responses of players is not sufficient to get constant-factor solutions. It might be easier to show that after a polynomial number of random best responses of players, the expected total profit of players is at least a constant factor of the optimal solution. Similar positive results for sink equilibria and random best responses of players are known for congestion games [7, 11].

The complexity of verifying if a given state of the distributed caching game is in a sink equilibrium or not is an interesting question to explore. Also, given a distributed caching game (or a many-to-one two-sided market game), an interesting problem is to check if the set of all sink equilibria is pure Nash equilibria or not. Finally, an interesting direction of research is to classify classes of two-sided market games for which pure Nash equilibria exists or best-response dynamics of players converge to a pure Nash equilibrium.

Cross-References

- ▸ Stable Marriage
- ▸ Stable Marriage with Ties and Incomplete Lists
- ▸ Best Response Algorithms for Selfish Routing

Recommended Reading

1. Ackermann H, Goldberg P, Mirrokni V, Röglin H, Vöcking B (2007) A unified approach to congestion games and twosidedmarkets. In: 3rd workshop of internet economics (WINE), pp 30–41
2. Ackermann H, Goldberg P, Mirrokni V, Röglin H, Vöcking B (2008) Uncoordinated two-sided markets. ACM Electronic Commerce (ACMEC)
3. Chekuri C, Khanna S (2000) A PTAS for the multiple knapsack problem. In: Proceedings of the 11th annual ACM-SIAM symposium on discrete algorithms (SODA), pp 213–222
4. Fleischer L, Goemans M, Mirrokni VS, Sviridenko M (2006) Tight approximation algorithms for maximum general assignment problems. In: Proceedings of the 16th annual ACM–SIAM symposium on discrete algorithms (SODA), pp 611–620
5. Gale D, Shapley L (1962) College admissions and the stability of marriage. Am Math Mon 69:9–15
6. Goemans M, Li L, Mirrokni VS, Thottan M (2004) Market sharing games applied to content distribution in ad-hoc networks. In: Proceedings of the 5th ACM international symposium on mobile ad hoc networking and computing (MobiHoc), pp 1020–1033
7. Goemans M, Mirrokni VS, Vetta A (2005) Sink equilibria and convergence. In: Proceedings of the 46th conference on foundations of computer science (FOCS), pp 123–131
8. Johnson D, Papadimitriou CH, Yannakakis M (1988) How easy is local search? J Comput Syst Sci 37:79–100
9. Knuth D (1976) Marriage Stables et leurs relations avec d'autres problèmes Combinatories. Les Presses de l'Université de Montréal
10. Kojima F, Unver Ü (2006) Random paths to pairwise stability in many-to-many matching problems: a study on market equilibration. Int J Game Theory
11. Mirrokni VS (2005) Approximation algorithms for distributed and selfish agents. Ph.D. thesis, Massachusetts Institute of Technology
12. Nash JF (1951) Non-cooperative games. Ann Math 54:268–295

13. Papadimitriou CH, Schaffer A, Yannakakis M (1990) On the complexity of local search. In: Proceedings of the 22nd symposium on theory of computing (STOC), pp 438–445
14. Roth AE, Vande Vate JH (1990) Random paths to stability in two sided matching. Econometrica 58(6):1475–1480
15. Schaffer A, Yannakakis M (1991) Simple local search problems that are hard to solve. SIAM J Comput 20(1):56–87
16. Vetta A (2002) Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In: Proceedings of the 43rd symposium on foundations of computer science (FOCS), pp 416–425

Matching in Dynamic Graphs

Surender Baswana[1], Manoj Gupta[2], and Sandeep Sen[2]
[1]Department of Computer Science and Engineering, Indian Institute of Technology (IIT), Kanpur, India
[2]Indian Institute of Technology (IIT) Delhi, Hauz Khas, New Delhi, India

Keywords

Dynamic graph; Matching; Maximum matching; Maximal matching; Randomized algorithms

Years and Authors of Summarized Original Work

2011; Baswana, Gupta, Sen

Problem Definition

Let $G = (V, E)$ be an undirected graph on $n = |V|$ vertices and $m = |E|$ edges. A matching in G is a set of edges $\mathcal{M} \subseteq E$ such that no two edges in $\mathcal{M}$ share any vertex. Matching has been one of the most well-studied problems in algorithmic graph theory for decades [4]. A matching $\mathcal{M}$ is called *maximum matching* if the number of edges in $\mathcal{M}$ is maximum. The fastest known algorithm for maximum matching, due to Micali and Vazirani [5], runs in $O(m\sqrt{n})$. A matching is said to be *maximal* if it is not strictly contained in any other matching. It is well known that a maximal matching achieves a factor 2 approximation of the maximum matching.

Key Result

We address the problem of maintaining maximal matching in a fully dynamic environment – allowing updates in the form of both insertion and deletion of edges. Ivković and Llyod [3] designed the first fully dynamic algorithm for maximal matching with $O((n+m)^{0.7072})$ update time. In this entry, we present a fully dynamic algorithm for maximal matching that achieves $O(\log n)$ expected amortized time per update.

Ideas Underlying the Algorithm

We begin with some terminologies and notations that will facilitate our description and also provide some intuition behind our approach. Let $\mathcal{M}$ denote a matching in the given graph at any instant – an edge $(u, v) \in \mathcal{M}$ is called a *matched* edge where u is referred to as a *mate* of v and vice versa. An edge in $E \backslash \mathcal{M}$ is an *unmatched* edge. A vertex x is *matched* if there exists an edge $(x, y) \in \mathcal{M}$; otherwise it is *free* or *unmatched*.

In order to maintain a maximal matching, it suffices to ensure that there is no edge (u, v) in the graph such that both u and v are free with respect to the matching $\mathcal{M}$. Therefore, a natural technique for maintaining a maximal matching is to keep track of each vertex if it is matched or free. When an edge (u, v) is inserted, we add (u, v) to the matching if u and v are free. For the case when an unmatched edge (u, v) is deleted, no action is required. Otherwise, for both u and v, we search their neighborhoods for any free vertex and update the matching accordingly. It follows that each update takes $O(1)$ computation time except when it involves deletion of a matched edge; in this case the computation time is of the order of the sum of the degrees of the two endpoints of the deleted edge. So this trivial algorithm is quite

efficient for *small* degree vertices, but could be expensive for *large* degree vertices. An alternate approach could be to match a free vertex u with a randomly chosen neighbor, say v. Following the standard adversarial model, it can be observed that an expected $\deg(u)/2$ edges incident to u will be deleted before deleting the matched edge (u, v). So the expected amortized cost per edge deletion for u is roughly $O\left(\frac{\deg(u)+\deg(v)}{\deg(u)/2}\right)$. If $\deg(v) < \deg(u)$, this cost is $O(1)$. But if $\deg(v) \gg \deg(u)$, then it can be as bad as the trivial algorithm. To circumvent this problem, we introduce an important notion, called *ownership* of edges. Intuitively, we assign an edge to that endpoint which has *higher* degree.

The idea of choosing a random mate and the trivial algorithm described above can be combined together to design a simple algorithm for maximal matching. This algorithm maintains a partition of the vertices into two levels. Level 0 consists of vertices which own *fewer* edges, and we handle the updates there using the trivial algorithm. Level 1 consists of vertices (and their mates) which own *larger* number of edges, and we use the idea of random mate to handle their updates. This $2-$LEVEL algorithm achieves $O(\sqrt{n})$ expected amortized time per update. A careful analysis of the $2-$LEVEL algorithm suggests that a *finer* partition of vertices could help in achieving a faster update time. This leads to our $\log_2 n -$ LEVEL algorithm that achieves expected amortized $O(\log n)$ time per update.

Our algorithm uses randomization very crucially in order to handle the updates efficiently. The matching maintained (based on the random bits) by the algorithm at any stage is not known to the adversary for it to choose the updates adaptively. This oblivious adversarial model is no different from randomized data structures like universal hashing.

The 2-LEVEL Algorithm

The algorithm maintains a partition of the set of vertices into two levels. Each edge present in the graph will be owned by one or both of its endpoints as follows. If both the endpoints of an edge are at level 0, then it is owned by both of them. Otherwise it will be owned by exactly that endpoint which lies at a higher level. If both the endpoints are at level 1, the tie will be broken suitably by the algorithm. Let $\mathcal{O}_u$ denote the set of edges owned by a vertex u at any instant of the algorithm. With a slight abuse of the notation, we will also use $\mathcal{O}_u$ to denote $\{v|(u, v) \in \mathcal{O}_u\}$. As the algorithm proceeds, the vertices will make transition from one level to another and the ownership of edges will also change accordingly.

The algorithm maintains the following three invariants after each update:

1. Every vertex at level 1 is matched. Every free vertex at level 0 has all its neighbors matched.
2. Every vertex at level 0 owns less than $\sqrt{n}$ edges at any stage.
3. Both endpoints of every matched edge are at the same level.

It follows from the first invariant that the matching $\mathcal{M}$ is maximal at each stage. The second and third invariants help in incorporating the two ideas of our algorithm efficiently.

Handling Insertion of an Edge

Let (u, v) be the edge being inserted. If either u or v are at level 1, there is no violation of any invariant. However, if both u and v are at level 0, then we proceed as follows. Both u and v become the owner of the edge (u, v). If u and v are free, then we add (u, v) to $\mathcal{M}$. Notice that the insertion of (u, v) also leads to increase of $|\mathcal{O}_u|$ and $|\mathcal{O}_v|$ by one and so may lead to violation of Invariant 2. We process the vertex that owns more edges; let u be that vertex. If $|\mathcal{O}_u| = \sqrt{n}$, then Invariant 2 has got violated. In order to restore it, u moves to level 1 and gets matched to some vertex, say y, selected uniformly at random from $\mathcal{O}_u$. Vertex y also moves to level 1 to satisfy Invariant 3. If w and x were, respectively, the earlier mates of u and y at level 0, then the matching of u with y has rendered w and x free. Both w and x search for free neighbors at level 0 and update the matching accordingly. It is easy to observe that in all these cases, it takes $O(\sqrt{n})$ time to handle an edge insertion.

Handling Deletion of an Edge

Let (u, v) be an edge that is deleted. If $(u, v) \notin \mathcal{M}$, all the invariants are still valid. Let us consider the more important case of $(u, v) \in \mathcal{M}$ – the deletion of (u, v) has caused u and v to become free. Therefore, the first invariant might have got violated for u and v. If edge (u, v) was at level 0, then both u and v search for a free neighbor and update the matching accordingly. This takes $O(\sqrt{n})$ time. If edge (u, v) was at level 1, then u (similarly v) is processed as follows.

First, u disowns all its edges whose other endpoint is at level 1. If $|\mathcal{O}_u|$ is still greater than or equal to $\sqrt{n}$, then u stays at level 1 and selects a random mate from $\mathcal{O}_u$. However, if $|\mathcal{O}_u|$ has fallen below $\sqrt{n}$, then u moves to level 0 and gets matched to a free neighbor (if any). For each neighbor of u at level 0, the transition of u from level 1 to 0 is, effectively, like insertion of a new edge. This transition leads to an increase in the number of owned edges by each neighbor of u at level 0. As a result, the second invariant for each such neighbor at level 0 may get violated if the number of edges it owns now becomes $\sqrt{n}$. To take care of these scenarios, we proceed as follows. We scan each neighbor of u at level 0, and for each neighbor w, with $|\mathcal{O}_w| = \sqrt{n}$, a mate is selected randomly from $\mathcal{O}_w$ and w is moved to level 1 along with its mate. This concludes the deletion procedure of edge (u, v).

Analysis of the Algorithm

It may be noted that, unlike insertion, the deletion of an edge could potentially lead to moving of many vertices from level 0 to 1 and this may involve significant computation. However, we will show that the expected amortized computation per update is $O(\sqrt{n})$.

We analyze the algorithm using the concept of *epochs*.

Definition 1 At any time t, let (u, v) be any edge in $\mathcal{M}$. Then the **epoch** of (u, v) at time t is the maximal time interval containing t during which $(u, v) \in \mathcal{M}$.

The entire life span of an edge (u, v) can be viewed as a sequence of epochs when it is matched, separated by periods when it is unmatched. Any edge update that does not change the matching is processed in O(1) time. An edge update that changes the matching results in the start of new epoch(s) or the termination of some existing epoch(s). And it is only during the creation or termination of an epoch that significant computation is involved. For the purpose of analyzing the update time (when matching is affected), we assign the computation performed to the corresponding epochs created or terminated. It is easy to see that the computation associated with an epoch at level 0 is $O(\sqrt{n})$. The computation associated with an epoch at level 1 is of the order of sum of the degrees of the endpoints of the corresponding matched edge which may be $\Omega(n)$. When a vertex moves from level 0 to 1, although it owns $\sqrt{n}$ edges, this may grow later to $O(n)$. So the computation associated with an epoch at level 1 can be quite high. We will show that the expected number of such epochs that get terminated during any arbitrary sequence of edge updates will be relatively small. The following lemma plays a key role.

Lemma 1 *The deletion of an edge (u, v) at level 1 terminates an epoch with probability $\leq 1/\sqrt{n}$.*

Proof The deletion of edge (u, v) will lead to termination of an epoch only if $(u, v) \in \mathcal{M}$. If edge (u, v) was owned by u at the time of its deletion, note that u owned at least $\sqrt{n}$ edges at the moment of start of its epoch. Since u selected its matched edge uniformly at random from these edges, the (conditional) probability is $\frac{1}{\sqrt{n}}$. The same argument applies if v was the owner, so (u, v) is a matched edge at the time of deletion of (u, v) with probability at most $1/\sqrt{n}$. □

Consider any sequence of m edge updates. We analyze the computation associated with all the epochs that get terminated during these m updates. It follows from Lemma 1 and the linearity of expectation that the expected number of epochs terminated at level 1 will be $m/\sqrt{n}$. As discussed above, computation associated with each epoch at level 1 is $O(n)$. So the expected computation associated with the termination of all epochs at level 1 is $O(m\sqrt{n})$. The number of epochs destroyed at level 0 is trivially bounded by

$O(m)$. Each epoch at level 0 has $O(\sqrt{n})$ computation associated with it, so the total computation associated with these epochs is also $O(m\sqrt{n})$. We conclude the following.

Theorem 1 *Starting with a graph on n vertices and no edges, we can maintain maximal matching for any sequence of m updates in expected $O(m\sqrt{n})$ time.*

The $\log_2 n -$ LEVEL Algorithm

The key idea for improving the update time lies in the second invariant of our $2 -$ LEVEL algorithm. Let $\alpha(n)$ be the threshold for the maximum number of edges that a vertex at level 0 can own. Consider an epoch at level 1 associated with some edge, say (u, v). The computation associated with this epoch is of the order of the number of edges u and v own which can be $\Theta(n)$ in the worst case. However, the expected duration of the epoch is of the order of the minimum number of edges u can own at the time of its creation, i.e., $\Theta(\alpha(n))$. Therefore, the expected amortized computation per edge deletion at level 1 is $O(n/\alpha(n))$. Balancing this with the $\alpha(n)$ update time at level 0 yields $\alpha(n) = \sqrt{n}$.

In order to improve the running time of our algorithm, we need to decrease the ratio between the maximum and the minimum number of edges a vertex can own during an epoch at any level. It is this ratio that determines the expected amortized time per edge deletion. This observation leads us to a finer partitioning of the ownership classes. When a vertex creates an epoch at level i, it owns at least 2^i edges, and during the epoch, it is allowed to own at most $2^{i+1} - 1$ edges. As soon as it owns 2^{i+1} edges, it migrates to a higher level. Notice that the ratio of maximum to minimum edges owned by a vertex during an epoch gets reduced from $\sqrt{n}$ to a constant leading to about $\log_2 n$ levels. Though the $\log_2 n -$LEVEL algorithm can be seen as a natural generalization of our $2 -$ LEVEL algorithm, there are many intricacies that make the algorithm and its analysis quite involved. For example, a single edge update may lead to a sequence of falls and rise of many vertices across the levels of the data structure. Moreover, there may be several vertices trying to fall or rise at any time while processing an update. Taking a top-down approach in processing these vertices simplifies the description of the algorithm. The analysis of the algorithm becomes easier when we analyze each level separately. This analysis at any level is quite similar to the analysis of LEVEL-1 in our $2-$LEVEL algorithm. We recommend the interested reader to refer to the journal version of this paper in order to fully comprehend the algorithm and its analysis. The final result achieved by our $\log_2 n -$ LEVEL algorithm is stated below.

Theorem 2 *Starting with a graph on n vertices and no edges, we can maintain maximal matching for any sequence of m updates in expected $O(m \log n)$ time.*

Using standard probability tools, it can be shown that the bound on the update time as stated in Theorem 2 holds with high probability, as well as with limited independence.

Open Problems

There have been new results on maintaining approximate weighted matching [2] and $(1 + \epsilon)$-approximate matching [1, 6] for $\epsilon < 1$. The interested reader should study these results. For any $\epsilon < 1$, whether it is possible to maintain $(1 + \epsilon)$-approximate matching in poly-logarithmic update time is still an open problem.

Recommended Reading

1. Anand A, Baswana S, Gupta M, Sen S (2012) Maintaining approximate maximum weighted matching in fully dynamic graphs. In: FSTTCS, Hyderabad, pp 257–266
2. Gupta M, Peng R (2013) Fully dynamic (1+e)-approximate matchings. In: FOCS, Berkeley, pp 548–557
3. Ivkovic Z, Lloyd EL (1994) Fully dynamic maintenance of vertex cover. In: WG '93: proceedings of the 19th international workshop on graph-theoretic concepts in computer science, Utrecht. Springer, London, pp 99–111
4. Lovasz L, Plummer M (1986) Matching theory. AMS Chelsea Publishing/North-Holland, Amsterdam/New York
5. Micali S, Vazirani VV (1980) An $O(\sqrt{(|V|)}|E|)$ algorithm for finding maximum matching in general graphs. In: FOCS, Syracuse, pp 17–27

6. Neiman O, Solomon S (2013) Simple deterministic algorithms for fully dynamic maximal matching. In: STOC, Palo Alto, pp 745–754

Matching Market Equilibrium Algorithms

Ning Chen[1] and Mengling Li[2]
[1]Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore
[2]Division of Mathematical Sciences, Nanyang Technological University, Singapore, Singapore

Keywords

Competitive equilibrium; Matching market; Maximum competitive equilibrium; Minimum competitive equilibrium

Years and Authors of Summarized Original Work

1971; Shapley, Shubik
1982; Kelso, Crawford
1986; Demange, Gale and Sotomayor

Problem Definition

The study of matching market equilibrium was initiated by Shapley and Shubik [13] in an assignment model. A classical instance of the matching market involves a set B of n unit-demand buyers and a set Q of m indivisible items, where each buyer wants to buy at most one item and each item can be sold to at most one buyer. Each buyer i has a valuation $v_{ij} \geq 0$ for each item j, representing the maximum amount that i is willing to pay for item j. Each item j has a reserve price $r_j \geq 0$, below which it won't be sold. Without loss of generality, one can assume there is a null item whose value is zero to all buyers and whose price is always zero.

An output of the matching market is a tuple $(\mathbf{p}, \mathbf{x})$, where $\mathbf{p} = (p_1, \ldots, p_m) \geq 0$ is a *price* vector with p_j denoting the price charged for item j and $\mathbf{x} = (x_1, \ldots, x_n) \geq 0$ is an *allocation* vector with x_i denoting the item that i wins. If i does not win any item, $x_i = \emptyset$. An output is essentially a bipartite matching with monetary transfers between the matched parties (buyers and items). A *feasible* price vector is any vector $\mathbf{p} \geq \mathbf{r} = (r_1 \ldots r_m)$. Given an outcome $(\mathbf{p}, \mathbf{x})$, the *utility* (*payoff*) to a buyer i who gets item j and pays p_j is $u_i(\mathbf{px}) = v_{ij} - p_j$ (assume linear surplus) and $u_i(\mathbf{px}) = 0$ if i gets nothing. At price $\mathbf{p}$, the *demand set* for buyer i is $D_i(\mathbf{p}) = \left\{ j \in \arg\max_{j'}(v_{ij'} - p_{j'}) \,\middle|\, v_{ij} - p_j \geq 0 \right\}$

A tuple $(\mathbf{p}, \mathbf{x})$ is called a *competitive equilibrium* if:

- For any item j, $p_j = r_j$ if no one wins j in allocation $\mathbf{x}$.
- If buyer i wins item j $(x_i = j)$, then $j \in D_i(\mathbf{p})$.
- If buyer i does not win any item $(x_i = \emptyset)$, then for every item j, $v_{ij} - p_j \leq 0$.

The first condition above is a market clearance (efficiency) condition, which says that all unallocated items are priced at the given reserve prices. The second and third conditions ensure envy-freeness (fairness), implying that each buyer is allocated with an item that maximizes his utility at these prices. In a market competitive equilibrium, all items with prices higher than the reserve prices are sold out and everyone gets his maximum utility at the corresponding allocation.

An outcome $(\mathbf{p}, \mathbf{x})$ is called a *minimum competitive equilibrium* if it is a competitive equilibrium and for any other competitive equilibrium $(\mathbf{p}', \mathbf{x}')$, $p_j \leq p'_j$ for every item j. It represents the interests of all buyers in terms of their total payment. Similarly, an outcome $(\mathbf{p}, \mathbf{x})$ is called a *maximum competitive equilibrium* if it is a competitive equilibrium and for any other competitive equilibrium $(\mathbf{p}', \mathbf{x}')$, $p_j \geq p'_j$ for every item j. It represents the interests of all sellers in terms of total payment received. Maximum and minimum equilibria represent the contradictory interests of the two parties in a two-sided matching market at the two extremes.

The solution concept of competitive equilibrium is closely related to the well-established stability solution concept initiated by Gale and Shapley [9] in pure two-sided matching markets without money. To define stability in a multi-item matching market with money, we need to have a definition of preferences. Buyers prefer items with larger utility $(v_{ij} - p_j)$, and sellers (items) prefer buyers with larger payment. Given an outcome $(\mathbf{p}, \mathbf{x})$, where p_j is the price of item j and x_i is the allocation of buyer i, we say (i, j) is a *blocking pair* if there is p'_j such that $p'_j > p_j$ (item j can receive more payment) and the utility that i obtains in $(\mathbf{p}, \mathbf{x})$ is less than $v_{ij} - p'_j$ (by payment p'_j to item j buyer i can get more utility). An outcome $(\mathbf{p}, \mathbf{x})$ is *stable* if it has no blocking pairs, that is, no unmatched buyer-seller pair can mutually benefit by trading with each other instead of their current partners.

Key Results

For any given matching market, Shapley and Shubik [13] formulate an efficient matching market outcome as a linear program of maximizing the social welfare of the allocation. The duality theorem then shows the existence of competitive equilibrium. They also prove that there is a unique minimum (maximum) equilibrium price vector.

Theorem 1 (Shapley and Shubik [13]) *A matching market competitive equilibrium always exits. The set of competitive equilibrium price vectors* $\mathbf{p}$ *form a lattice in* $\mathbb{R}^m_{\geq 0}$.

Shapley and Shubik [13] also establish the connections between stability and competitive equilibrium.

Theorem 2 (Shapley and Shubik [13]) *In a multi-item market, an outcome* $(\mathbf{p}, \mathbf{x})$ *is a competitive equilibrium if and only if it is stable.*

However, they did not define an adjustment process like the deferred-acceptance algorithm by Gale and Shapley [9]. Crawford and Knoer [5] study a more generalized setup with firms and workers, where firms can hire multiple workers while each worker can be employed by at most one firm. It is essentially a many-to-one matching framework, where firms can be considered as buyers with multi-unit demand while buyers can be considered as items. They describe a salary adjustment process, which is essentially a version of the deferred-acceptance algorithm. They also show that for arbitrary capacities of the firms (buyers), when ties are ruled out, it always converges to a minimum competitive equilibrium. They provide an alternative proof of Shapley and Shubik's [13] result and allow significant generalization of it. However, the analysis in Crawford and Knoer [5] is flawed by two unnecessarily restrictive assumptions. Later, Kelso and Crawford [10] relax these assumptions and propose a modification to the salary adjustment process, which works as follows (firms are essentially buyers with multi-unit demand and workers are the items):

Salary Adjustment Process (Kelso-Crawford [10])

1. Firms begin by facing a set of initially very low salaries (reserve prices).
2. Firms make offers to their most preferred set of workers. Any offer previously made by a firm to a worker that was not rejected must be honored.
3. Each worker who receives one or more offers rejects all but his favorite, which he tentatively accepts.
4. Offers not rejected in previous periods remain in force. For each rejected offer a firm made, increase the feasible salary for the rejecting worker. Firms continue to make offers to their favorite sets of workers.
5. The process stops when no rejections are made. Workers then accept the offers that remain in force from the firms they have not rejected.

An important assumption underlying the algorithm is that workers are gross substitutes from the standpoint of the firm. That is, when the price of one worker goes up, demand for another worker should not go down. As the salary adjustment process goes on, for firms, the set of

feasible offers is reduced as some offers are rejected. For workers, the set of offers grows as more offers come up. Therefore, there is a monotonicity of offer sets in opposite directions. Alternatively, we can have a worker-proposing algorithm, where workers all begin by offering their services at the highest possible wage at their most preferred firm.

Theorem 3 (Kelso and Crawford [10]) *The salary adjustment process converges to a minimum competitive equilibrium, provided that all workers are gross substitutes from each firm's standpoint. In other words the final competitive equilibrium allocation is weakly preferred by every firm to any other equilibrium allocation.*

Taking a different perspective, Demange, Gale, and Sotomayor [7] propose an ascending auction-based algorithm (called "exact auction mechanism" in their original paper) that converges to a minimum competitive equilibrium for the original one-to-one matching problem. It is a variant of the so-called Hungarian method by Kuhn [11] for solving the optimal assignment problem. The algorithm works as the following:

Exact Auction Mechanism (Demange-Gale-Sotomayor [7])

1. Assume (for simplicity) that valuations v_{ij} are all integers.
2. Set the initial price vector, $\mathbf{p}^0$, to the reserve prices, $\mathbf{p}^0 = \mathbf{r}$
3. At round t when current prices are $\mathbf{p}^t$, each buyer i declares his demand set, $D_i(\mathbf{p}^t)$.
4. If there is no over-demanded set of items, terminate the process. A market equilibrium at prices $\mathbf{p}^t$ exists, and a corresponding allocation can be found by maximum matching.
5. If there exits over-demanded set(s), find a minimal over-demanded set S, and for all $j \in S$, $p_j^{t+1} = p_j^t + 1$. Set $t = t + 1$, and go to step 3.

Theorem 4 (Demange, Gale, and Sotomayor [7]) *The exact auction mechanism always finds a competitive equilibrium. Moreover, the equilibrium it finds is the minimum competitive equilibrium.*

This auction outcome can be computed efficiently since we can compute (minimal) over-demanded sets in P-time. It is not necessary to increment prices by one unit in each iteration. Instead, one can raise prices in the over-demanded set until the demand set of one of the respective bidders enlarges. It turns out that the payments (minimum price equilibrium) are precisely the VCG payments. Therefore, the mechanism is incentive compatible, and for every buyer, it is a dominant strategy to specify his true valuations. Moreover, this mechanism is even group strategy-proof, meaning that no strict subset of buyers who can collude have an incentive to misrepresent their true valuations.

Demange, Gale, and Sotomayor [7] also propose an approximation algorithm, called "approximate auction mechanism" for computing a minimum competitive equilibrium. It is a version of the deferred-acceptance algorithm proposed by Crawford and Knoer [5], which in turn is a special case of the algorithm of Kelso and Crawford [10]. The algorithm works as follows.

Approximate Auction Mechanism (Demange-Gale-Sotomayor [7])

1. Set the initial price vector, $\mathbf{p}^0$, to the reserve prices, $\mathbf{p}^0 = \mathbf{r}$
2. At round t when current prices are $\mathbf{p}^t$, each buyer i may bid for any item. When he does so, he is committed to that item, which means he commits himself to possibly buying the item at the announced price. The item is (tentatively) assigned to that bidder.
3. At this point, any uncommitted bidder may:
   - Bid for some unassigned item, in which case he becomes committed to it at its initial price.
   - Bid for an assigned item, in which case he becomes committed to that item, its price increase by some fixed amount δ, and the bidder to whom it was assigned becomes uncommitted
   - Drop out of the bidding.
4. When there are no more uncommitted bidders, the auction terminates. Each committed bidder buys the item assigned to him at its current price.

This approximate auction mechanism would be appealing to the buyers since it does not require them to decide in advance exactly what their bidding behavior will be. Instead, at each stage, a buyer can make use of present and past stages of the auction to decide his next bid. If buyers behave in accordance with linear valuations, the final price will differ from the minimum equilibrium price by at most $k\delta$ units, where k is the minimum of the number of items and bidders. Thus, by making δ (the unit by which bids are increased) sufficiently small, one can get arbitrarily close to the minimum equilibrium price.

Theorem 5 (Demange, Gale, and Sotomayor [7]) *Under the approximate auction mechanism, the final price of an item will differ from the minimum equilibrium price by at most $k\delta$, where $k = \min(m, n)$.*

The mechanisms discussed so far (approximately) compute a minimum competitive equilibrium. These approaches can be easily transformed to compute a maximum competitive equilibrium. Chen and Deng [3] discuss a combinatorial algorithm which iteratively increases prices to converge to a maximum competitive equilibrium starting from an arbitrary equilibrium.

Applications

The assignment model is used by Becker [2] to study marriage and household economics. Based on the fact that stable outcomes all correspond to optimal assignments, he studies which men are matched to which women under different assumptions of the assignment matrix.

Extensions

The existence of competitive equilibrium has later been established by Crawford and Knoer [5], Gale [8], and Quinnzi [12] for more general utility functions rather than the linear surplus, provided $u_{ij}(\cdot)$ is strictly decreasing and continuous everywhere.

Under a minimum competitive equilibrium mechanism, it is a dominant strategy for every buyer to report his true valuation. On the other hand, under a maximum competitive equilibrium mechanism, while the sellers will be truthful, it is possible that some buyer bids a false value to obtain more utility. In a recent study, Chen and Deng [3] show the convergence from the maximum competitive equilibrium toward the minimum competitive equilibrium in a deterministic and dynamic setting.

Another strand of recent studies focus on the assignment model with budget constraints, which is applicable to many marketplaces such as online and TV advertising markets. An extra budget constraint introduces discontinuity in the utility function, which fundamentally changes the properties of competitive equilibria. In such setups, a competitive equilibrium does not always exist. Aggarwal et al. [1] study the problem of computing a weakly stable matching in the assignment model with quasi-linear utilities subject to a budget constraint. However, a weakly stable matching does not possess the envy-freeness property of a competitive equilibrium. Chen et al. [4] establish a connection between competitive equilibrium in the assignment model with budgets and strong stability. Then they give a strong polynomial time algorithm for deciding existence of and computing a minimum competitive equilibrium for a general class of utility functions in the assignment model with budgets.

Cross-References

- ▸ Hospitals/Residents Problem
- ▸ Optimal Stable Marriage
- ▸ Stable Marriage
- ▸ Stable Marriage and Discrete Convex Analysis
- ▸ Stable Marriage with Ties and Incomplete Lists

Recommended Reading

1. Aggarwal G, Muthukrishnan S, Pal D, Pal M (2009) General auction mechanism for search advertising. In: WWW, Madrid, pp 241–250

2. Becker GS (1981) A treatise on the family. Harvard University Press, Cambridge
3. Chen N, Deng X (2011) On Nash dynamics of matching market equilibria. CoRR abs/1103.4196
4. Chen N, Deng X, Ghosh A (2010) Competitive equilibria in matching markets with budgets. ACM SIGecom Exch 9(1):1–5
5. Crawford VP, Knoer EM (1981) Job matching with heterogeneous firms and workers. Econometrica 49(2):437–450
6. Demange G, Gale D (1985) The strategy structure of two-sided matching markets. Econometrica 53(4):873–883
7. Demange G, Gale D, Sotomayor M (1986) Multi-item auctions. J Pol Econ 94(4):863–872
8. Gale D (1984) Equilibrium in a discrete exchange economy with money. Int J Game Theory 13:61–64
9. Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69(1):9–15
10. Kelso AS, Crawford VP (1982) Job matching, coalition formation, and gross substitutes. Econometrica 50(6):1483–1504
11. Kuhn HW (1955) The Hungarian method for the assignment problem. Nav Reserv Logist Q 2(1):83–97
12. Quinnzi M (1984) Core and competitive equilibrium with indivisibilities. Int J Game Theory 13:41–60
13. Shapley LS, Shubik M (1971) The assignment game I: the core. Int J Game Theory 1(1):110–130

Matroids in Parameterized Complexity and Exact Algorithms

Fahad Panolan[1] and Saket Saurabh[1,2]
[1]Institute of Mathematical Sciences, Chennai, India
[2]University of Bergen, Bergen, Norway

Keywords

Exact algorithms; Matroids; Parameterized algorithms; Representative families

Years and Authors of Summarized Original Work

2009; Marx
2014; Fomin, Lokshtanov, Saurabh
2014; Fomin, Lokshtanov, Panolan, Saurabh
2014; Shachnai, Zehavi

Problem Definition

In recent years matroids have been used in the fields of parameterized complexity and exact algorithms. Many of these works mainly use a computation of *representative families*. Let $M = (E, \mathcal{I})$ be a matroid and $\mathcal{S} = \{S_1, \ldots, S_t\} \subseteq \mathcal{I}$ be a family of independent sets of size p. A subfamily $\hat{\mathcal{S}} \subseteq \mathcal{S}$ is called a *q-representative family* for $\mathcal{S}$ (denoted by $\hat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$), if for every $Y \subseteq E$ of size at most q, if there exists a set $S \in \mathcal{S}$ disjoint from Y with $S \cup Y \in \mathcal{I}$, then there exists a set $\hat{S} \in \hat{\mathcal{S}}$ disjoint from Y with $\hat{S} \cup Y \in \mathcal{I}$. The basic algorithmic question regarding representative families is, given a matroid $M = (E, \mathcal{I})$, a family $\mathcal{S} \subseteq \mathcal{I}$ of independent sets of size p and a positive integer q, compute $\hat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$ of size as small as possible in time as fast as possible.

The Two-Families Theorem of Bollobás [1] for extremal set systems implies that every family of independent sets of size p in a uniform matroid has a q-representative family with at most $\binom{p+q}{p}$ sets. The generalization of Two-Families Theorem to subspaces of a vector space by Lovász [5] implies that every family of independent sets of size p in a linear matroid has a q-representative family with at most $\binom{p+q}{p}$ sets. In fact one can show that the cardinality $\binom{p+q}{p}$ of a q-representative family of a family of independent sets of size p is optimal. It is important to note that the size of the q-representative family of a family of sets of size p in a uniform or linear matroid only depends on p and q and not on the cardinality of ground set of the matroid, and this fact is used to design parameterized and exact algorithms.

Key Results

For uniform matroids, Monien [8] gave an algorithm for computing a q-representative family of size at most $\sum_{i=0}^{q} p^i$ in time $\mathcal{O}(pq \cdot \sum_{i=0}^{q} p^i \cdot t)$ and Marx [6] gave another algorithm, for computing a q-representative families of size at most $\binom{p+q}{p}$ in time $\mathcal{O}(p^q \cdot t^2)$. For uniform matroids, Fomin et al. [2] proved the following theorem.

Theorem 1 ([2, 9]) *Let $\mathcal{S} = \{S_1, \ldots, S_t\}$ be a family of sets of size p over a universe of size n and let $0 < x < 1$. For a given q, a q-representative family $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$ with at most $x^{-p}(1-x)^{-q} \cdot 2^{o(p+q)} \log n$ sets can be computed in time $\mathcal{O}((1-x)^{-q} \cdot 2^{o(p+q)} \cdot t \cdot \log n)$.*

In [3], Fomin, Lokshtanov, and Saurabh proved Theorem 1 for $x = \frac{p}{p+q}$. That is, a q-representative family $\hat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$ with at most $\binom{p+q}{p} \cdot 2^{o(p+q)} \cdot \log n$ sets can be computed in time $\mathcal{O}((\frac{p+q}{q})^q \cdot 2^{o(p+q)} \cdot t \cdot \log n)$. Later Fomin et al. [2] observed that the proof in [3] can be modified to work for every $0 < x < 1$ and allows an interesting trade-off between the size of the computed representative families and the time taken to compute them, and this trade-off can be exploited algorithmically to speed up "representative families based" algorithms. Independently, at the same time, Shachnai and Zehavi [9] also observed that the proof in [3] can be generalized to get Theorem 1. We would like to mention that in fact a variant of Theorem 1 is proved in [2, 9] which computes a weighted q-representative family. The proof of Theorem 1 uses algorithmic variant of "random permutation" proof of Bollobás Lemma and an efficient construction of a variant of universal sets called n-p-q-separating collections.

For linear matroids, Marx [7] showed that Lovász's proof can be transformed into an algorithm computing a q-representative family:

Theorem 2 ([7]) *Given a linear representation A_M of a matroid $M = (E, \mathcal{I})$, a family $\{S_1, \ldots, S_t\}$ of independent sets of size p and a positive integer q, there is an algorithm which computes $\hat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$ of size $\binom{p+q}{p}$ in time $2^{\mathcal{O}(p \log(p+q))} \cdot \binom{p+q}{p}^{\mathcal{O}(1)} (||A_M||t)^{\mathcal{O}(1)}$ where $||A_M||$ is the size of A_M.*

Fomin, Lokshtanov, and Saurabh [3] gave an efficient computation of representative families in linear matroids:

Theorem 3 ([3]) *Let $M = (E, \mathcal{I})$ be a linear matroid of rank $p + q = k$ given together with its representation matrix A_M over a field $\mathbb{F}$. Let $\mathcal{S} = \{S_1, \ldots, S_t\}$ be a family of independent sets of size p. Then $\hat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$ with at most $\binom{p+q}{p}$ sets can be computed in $\mathcal{O}\left(\binom{p+q}{p} tp^\omega + t\binom{p+q}{q}^{\omega-1}\right)$ operations over $\mathbb{F}$, where $\omega < 2.373$ is the matrix multiplication exponent.*

We would like to draw attention of the reader that in Theorems 3 and 2, the cardinality of the computed q-representative family is optimal and polynomial in p and q if one of p or q is a constant, unlike in Theorem 1. As in the case of Theorem 1, Theorem 3 is also proved for a weighted q-representative family.

Most of the algorithms using representative families are dynamic programming algorithms. A class of families which often arise in dynamic programming are product families. A family $\mathcal{F}$ is called *product* of two families $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{A}$ and $\mathcal{B}$ are families of independent sets in a matroid $M = (E, \mathcal{I})$, if $\mathcal{F} = \{A \cup B \mid A \in \mathcal{A}, B \in \mathcal{B}, A \cap B = \emptyset, A \cup B \in \mathcal{I}\}$. Fomin et al. [2] gave two algorithms to compute q-representative family of a product family $\mathcal{F}$, one in case of uniform matroid and other in case of linear matroid. These algorithms significantly outperform the naive way of computing the product family $\mathcal{F}$ first and then a representative family of it.

Applications

Representative families are used to design efficient algorithms in parameterized complexity and exact algorithms.

Parameterized and Exact Algorithms

In this subsection we list some of the parameterized and exact algorithms obtained using representative families.

1. ℓ-Matroid Intersection. In this problem we are given ℓ matroids $M_1 = (E, \mathcal{I}_1), \ldots, M_\ell = (E, \mathcal{I}_\ell)$ along with their linear representations $A_{M_1}, \ldots, A_{M_\ell}$ over the same field $\mathbb{F}$ and a positive integer k. The objective is to find a k element subset of E, which is independent in all the matroids $M_1, \ldots, M_\ell$. Marx [7] gave a randomized

algorithm for the problem running in time $f(k,l)\left(\sum_{i=1}^{\ell}||A_{M_i}||\right)^{\mathcal{O}(1)}$, where f is a computable function. By giving an algorithm for deterministic truncation of linear matroids, Lokshtanov et al. [4] gave a deterministic algorithm for the problem running in time $2^{\omega k\ell}\left(\sum_{i=1}^{\ell}||A_{M_i}||\right)^{\mathcal{O}(1)}$, where ω is the matrix multiplication exponent.

2. Long Directed Cycle. In the Long Directed Cycle problem, we are interested in finding a cycle of length at least k in a directed graph. Fomin et al. [2] and Shachnai and Zehavi [9] gave an algorithm of running time $\mathcal{O}(6.75^{k+o(k)}mn^2\log^2 n)$ for this problem.
3. Short Cheap Tour. In this problem we are given an undirected n-vertex graph G, $w : E(G) \to \mathbb{N}$ and an integer k. The objective is to find a path of length k with minimum weight. Fomin et al. [2] and Shachnai and Zehavi [9] gave a $\mathcal{O}(2.619^k n^{\mathcal{O}(1)}\log W)$ time algorithm for Short Cheap Tour, where W is the largest edge weight in the given input graph.
4. Multilinear Monomial Detection. Here the input is an arithmetic circuit C over $\mathbb{Z}^+$ representing a polynomial $P(X)$ over $\mathbb{Z}^+$. The objective is to test whether $P(X)$ construed as a sum of monomials contain a multilinear monomial of degree k. For this problem Fomin et al. [2] gave an algorithm of running time $\mathcal{O}(3.8408^k 2^{o(k)}s(C)n\log^2 n)$, where $s(C)$ is the size of the circuit.
5. Minimum Equivalent Graph(MEG). In this problem we are seeking a spanning subdigraph D' of a given n-vertex digraph D with as few arcs as possible in which the reachability relation is the same as in the original digraph D. Fomin, Lokshtanov, and Saurabh [3] gave the first single-exponential exact algorithm, i.e., of running time $2^{\mathcal{O}(n)}$, for the problem.
6. **Dynamic Programming Over Graphs of Bounded Treewidth.** Fomin et al. [2] gave algorithms with running time $\mathcal{O}\left((1+2^{\omega-1}\cdot 3)^{\mathbf{tw}}\mathbf{tw}^{O(1)}n\right)$ for Feedback Vertex Set and Steiner Tree, where **tw** is the treewidth of the input graph, n is the number of vertices in the input graph, and ω is the matrix multiplication exponent.

Open Problems

1. Can we improve the running time for the computation of representative families in linear matroids or in specific matroids like graphic matroids?

Cross-References

- Shadowless Solutions for Fixed-Parameter Tractability of Directed Graphs
- Treewidth of Graphs

Recommended Reading

1. Bollobás B (1965) On generalized graphs. Acta Math Acad Sci Hungar 16:447–452
2. Fomin FV, Lokshtanov D, Panolan F, Saurabh S (2014) Representative sets of product families. In: Proceedings of 22nd Annual European Symposium on Algorithms (ESA 2014), Wroclaw, 8–10 Sept 2014, vol 8737, pp 443–454. doi:10.1007/978-3-662-44777-2_37
3. Fomin FV, Lokshtanov D, Saurabh S (2014) Efficient computation of representative sets with applications in parameterized and exact algorithms. In: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014), Portland, 5–7 Jan 2014, pp 142–151. doi:10.1137/1.9781611973402.10
4. Lokshtanov D, Misra P, Panolan F, Saurabh S (2015) Deterministic truncation of linear matroids. In: Proceedings of 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015), Kyoto, 6–10 July 2015, Part I, pp 922–934. doi:10.1007/978-3-662-47672-7_75
5. Lovász L (1977) Flats in matroids and geometric graphs. In: Combinatorial surveys (Proceedings of the Sixth British Combinatorial Conference, Royal Holloway College, Egham). Academic, London, pp 45–86
6. Marx D (2006) Parameterized coloring problems on chordal graphs. Theor Comput Sci 351(3):407–424
7. Marx D (2009) A parameterized view on matroid optimization problems. Theor Comput Sci 410(44):4471–4479
8. Monien B (1985) How to find long paths efficiently. In: Analysis and design of algorithms for combinatorial problems (Udine, 1982). North-Holland mathe-

matics studies, vol 109. North-Holland, Amsterdam, pp 239–254. doi:10.1016/S0304-0208(08)73110-4
9. Shachnai H, Zehavi M (2014) Representative families: a unified tradeoff-based approach. In: Proceedings of 22nd Annual European Symposium on Algorithms (ESA 2014), Wroclaw, 8–10 Sept 2014, vol 8737, pp 786–797. doi:10.1007/978-3-662-44777-2_65

Max Cut

Alantha Newman
CNRS-Université Grenoble Alpes and G-SCOP, Grenoble, France

Keywords

Approximation algorithms; Graph partitioning

Synonyms

Maximum bipartite subgraph

Year and Authors of Summarized Original Work

1994; 1995; Goemans, Williamson

Problem Definition

Given an undirected edge-weighted graph, $G = (V, E)$, the maximum cut problem (MAX CUT) is to find a bipartition of the vertices that maximizes the weight of the edges crossing the partition. If the edge weights are non-negative, then this problem is equivalent to finding a maximum weight subset of the edges that forms a bipartite subgraph, i.e., the maximum bipartite subgraph problem. All results discussed in this article assume non-negative edge weights. MAX CUT is one of Karp's original NP-complete problems [20]. In fact, it is NP-hard to approximate to within a factor better than $\frac{16}{17}$ [17, 35].

For nearly 20 years, the best-known approximation factor for MAX CUT was half, which can be achieved by a very simple algorithm: form a set S by placing each vertex in S with probability half. Since each edge crosses the cut $(S, V \setminus S)$ with probability half, the expected value of this cut is half the total edge weight. This implies that for any graph, there exists a cut with value at least half of the total edge weight. In 1976, Sahni and Gonzalez presented a deterministic half-approximation algorithm for MAX CUT, which is essentially a de-randomization of the aforementioned randomized algorithm [31]: iterate through the vertices and form sets S and $\bar{S}$ by placing each vertex in the set that maximizes the weight of cut $(S, \bar{S})$ thus far. After each iteration of this process, the weight of this cut will be at least half of the weight of the edges with both endpoints in $S \cup \bar{S}$.

This simple half-approximation algorithm uses the fact that for any graph with non-negative edge weights, the total edge weight of a given graph is an upper bound on the value of its maximum cut. There exist classes of graphs for which a maximum cut is arbitrarily close to half the total edge weight, i.e., graphs for which this "trivial" upper bound can be close to twice the true value of an optimal solution. An example of such a class of graphs is complete graphs on n vertices, K_n. In order to obtain an approximation factor better than half, one must be able to compute an upper bound on the value of a maximum cut that is better, i.e., smaller, than the trivial upper bound for such classes of graphs.

Linear Programming Relaxations

For many optimization (maximization) problems, linear programming has been shown to yield better (upper) bounds on the value of an optimal solution than can be obtained via combinatorial methods. There are several well-studied linear programming relaxations for MAX CUT. For example, a classical integer program has a variable x_e for each edge and a constraint for each odd cycle, requiring that an odd cycle C contribute at most $|C| - 1$ edges to an optimal solution.

$$\begin{aligned} \max \sum_{e \in E} w_e x_e & \\ \sum_{e \in C} x_e \;&\leq\; |C| - 1 \quad \forall \text{ odd cycles } C \\ x_e \;&\in\; \{0,1\}. \end{aligned}$$

The last constraint can be relaxed so that each x_e is required to lie between 0 and 1, but need not be integral, i.e., $0 \leq x_e \leq 1$. Although this relaxation may have exponentially many constraints, there is a polynomial-time separation oracle (equivalent to finding a minimum weight odd cycle), and thus, the relaxation can be solved in polynomial time [14]. Another classical integer program contains a variable x_{ij} for each pair of vertices. In any partition of the vertices, either zero or two edges from a three-cycle cross the cut. This requirement is enforced in the following integer program. If edge $(i, j) \notin E$, then w_{ij} is set to 0.

$$\begin{aligned} \max \sum_{i,j \in V} w_{ij} x_{ij} & \\ x_{ij} + x_{jk} + x_{ki} \;&\leq\; 2 \quad \forall i, j, k \in V \\ x_{ij} + x_{jk} - x_{ki} \;&\geq\; 0 \quad \forall i, j, k \in V \\ x_{ij} \;&\in\; \{0,1\}. \end{aligned}$$

Again, the last constraint can be relaxed so that each x_{ij} is required to lie between 0 and 1. In contrast to the aforementioned cycle-constraint-based linear program, this linear programming relaxation has a polynomial number of constraints.

Both of these relaxations actually have the same optimal value for any graph with non-negative edge weights [3,26,30]. (For a simplified proof of this, see [25].) Poljak showed that the integrality gap for each of these relaxations is arbitrarily close to 2 [26]. In other words, there are classes of graphs that have a maximum cut containing close to half of the edges, but for which each of the above relaxations yields an upper bound close to all the edges, i.e., no better than the trivial "all-edges" bound. In particular, graphs with a maximum cut close to half the edges and with high girth can be used to demonstrate this gap. A comprehensive look at these linear programming relaxations is contained in the survey of Poljak and Tuza [30].

Another natural integer program uses variables for vertices rather than edges:

$$\max \sum_{(i,j) \in E} w_{ij} \left(x_i (1 - x_j) + x_j (1 - x_i) \right) \quad (1)$$

$$x_i \;\in\; \{0,1\} \qquad \forall i \in V. \quad (2)$$

Replacing (2) with $x_i \in [0,1]$ results in a nonlinear relaxation that is actually just as hard to solve as the integer program. This follows from the fact that any fractional solution can be rounded to obtain an integer solution with at least the same value. Indeed, for any vertex $h \in V$ with fractional value x_h, we can rewrite the objective function (1) as follows. Edges adjacent to vertex h are denoted by $\delta(h)$. For ease of notation, let us momentarily assume the graph is unweighted, although the argument works for non-negative edge weights.

$$\sum_{(i,j) \in E \setminus \delta(h)} x_i (1 - x_j) + x_j (1 - x_i) + x_h \overbrace{\sum_{j \in \delta(h)} (1 - x_j)}^{A} + (1 - x_h) \overbrace{\sum_{j \in \delta(h)} x_j}^{B}. \quad (3)$$

If $A \geq B$, we round x_ℓ to 1, otherwise we round it to 0. Repeating this process for all vertices results in an integral solution whose objective value is no less than the objective value of the initial fractional solution.

Eigenvalue Upper Bounds

Delorme and Poljak [8] presented an eigenvalue upper bound on the value of a maximum cut, which was a strengthened version of a previous eigenvalue bound considered by Mohar and Poljak [24]. Computing Delorme and Poljak's upper bound is equivalent to solving an eigenvalue minimization problem. They showed that their bound is computable in polynomial time with arbitrary precision. In a series of work, Delorme, Poljak and Rendl showed that this upper bound behaves "differently" from the linear programming-based

upper bounds. For example, they studied classes of sparse random graphs (e.g., $G(n, p)$ with $p = 50/n$) and showed that their upper bound is close to optimal on these graphs [9]. Since graphs of this type can also be used to demonstrate an integrality gap arbitrarily close to 2 for the aforementioned linear programming relaxations, their work highlighted contrasting behavior between these two upper bounds. Further computational experiments on other classes of graphs gave more evidence that the bound was indeed stronger than previously studied bounds [27, 29]. Delorme and Poljak conjectured that the five cycle demonstrated the worst-case behavior for their bound: a ratio of $\frac{32}{25+5\sqrt{5}} \approx 0.88445$ between their bound and the optimal integral solution. However, they could not prove that their bound was strictly less than twice the value of a maximum cut in the worst case.

Key Result

In 1994, Goemans and Williamson presented a randomized 0.87856-approximation algorithm for MAX CUT [12]. Their breakthrough work was based on rounding a semidefinite programming relaxation and was the first use of semidefinite programming in approximation algorithms. Poljak and Rendl showed that the upper bound provided by this semidefinite relaxation is equivalent to the eigenvalue bound of Delorme and Poljak [28]. Thus, Goemans and Williamson proved that the eigenvalue bound of Delorme and Poljak is no more than 1.138 times the value of a maximum cut.

A Semidefinite Relaxation

MAX CUT can be formulated as the following quadratic integer program, which is NP-hard to solve. Each vertex $i \in V$ is represented by a variable y_i, which is assigned either 1 or -1 depending on which side of the cut it appears.

$$\begin{aligned} \max \ & \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - y_i y_j) \\ y_i \ & \in \ \{-1, 1\} \qquad \forall i \in V. \end{aligned}$$

Goemans and Williamson considered the following relaxation of this integer program, in which each vertex is represented by a unit vector.

$$\begin{aligned} \max \ & \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - v_i \cdot v_j) \\ v_i \cdot v_i \ & = \ 1 \qquad \forall i \in V \\ v_i \ & \in \ \mathcal{R}^n \qquad \forall i \in V. \end{aligned}$$

They showed that this relaxation is equivalent to a semidefinite program. Specifically, consider the following semidefinite relaxation:

$$\begin{aligned} \max \ & \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - y_{ij}) \\ y_{ii} \ & = \ 1 \qquad \forall i \in V \\ Y \ & \qquad \text{positive semidefinite.} \end{aligned}$$

The equivalence of these two relaxations is due to the fact that a matrix Y is positive semidefinite if and only if there is a matrix B such that $B^T B = Y$. The latter relaxation can be solved to within arbitrary precision in polynomial time via the ellipsoid algorithm, since it has a polynomial-time separation oracle [15]. Thus, a solution to the first relaxation can be obtained by finding a solution to the second relaxation and finding a matrix B such that $B^T B = Y$. If the columns of B correspond to the vectors $\{v_i\}$, then $y_{ij} = v_i \cdot v_j$, yielding a solution to the first relaxation.

Random-Hyperplane Rounding

Goemans and Williamson showed how to round the semidefinite programming relaxation of MAX CUT using a new technique that has since become known as "random-hyperplane rounding" [12]. First obtain a solution to the first relaxation, which consists of a set of unit vectors $\{v_i\}$, one vector for each vertex. Then choose a random vector $r \in \mathcal{R}^n$ in which each coordinate of r is chosen from the standard normal distribution. Finally, set $S = \{i \mid v_i \cdot r \geq 0\}$ and output the cut $(S, V \setminus S)$.

The probability that a particular edge $(i, j) \in E$ crosses the cut is equal to the probability that the dot products $v_i \cdot r$ and $v_j \cdot r$ differ in sign. This probability is exactly equal to θ_{ij}/π, where θ_{ij} is the angle between vectors v_i and v_j. Thus, the expected weight of edges crossing the cut is equal to $\sum_{(i,j)\in E} \theta_{ij}/\pi$. How large is this compared to the objective value given by the semidefinite programming relaxation, i.e., what is the approximation ratio?

Define α_{gw} as the worst-case ratio of the expected contribution of an edge to the cut, to its contribution to the objective function of the semidefinite programming relaxation. In other words: $\alpha_{gw} = \min_{0\le\theta\le\pi} \frac{2}{\pi}\frac{\theta}{1-\cos\theta}$. It can be shown that $\alpha_{gw} > 0.87856$. Thus, the expected value of a cut is at least $\alpha_{gw} \cdot SDP_{OPT}$, resulting in an approximation ratio of at least 0.87856 for MAX CUT. The same analysis applies to weighted graphs with non-negative edge weights.

This algorithm was de-randomized by Mahajan and Hariharan [23]. Goemans and Williamson also applied their random-hyperplane rounding techniques to give improved approximation guarantees for other problems such as MAX-DICUT and MAX-2SAT.

Integrality Gap and Hardness

Karloff showed that there exist graphs for which the best hyperplane is only a factor α_{gw} of the maximum cut [19], showing that there are graphs for which the analysis in [12] is tight. Since the optimal SDP value for such graphs equals the optimal value of a maximum cut, these graphs cannot be used to demonstrate an integrality gap. However, Feige and Schechtman showed that there exist graphs for which the maximum cut is a α_{gw} fraction of the SDP bound [10], thereby establishing that the approximation guarantee of Goemans and Williamson's algorithm matches the integrality gap of their semidefinite programming relaxation. Recently, Khot, Kindler, Mossel, and O'Donnell [22] showed that if the Unique Games Conjecture of Khot [21] is true, then it is NP-hard to approximate MAX CUT to within any factor larger than α_{gw}.

Better-than-Half Approximations Without SDPs

Since Goemans and Williamson presented an α_{gw}-approximation algorithm for MAX CUT, it has been an open question if one can obtain a matching approximation factor or even an approximation ratio of $\frac{1}{2} + \epsilon$ for some constant $\epsilon > 0$ without using SDPs. Trevisan presented an algorithm based on spectral partitioning with running time $\tilde{O}(n^2)$ and an approximation guarantee of 0.531 [34], which was subsequently improved to 0.614 by Soto [32].

Applications

The work of Goemans and Williamson paved the way for the further use of semidefinite programming in approximation algorithms, particularly for graph partitioning problems. Methods based on the random-hyperplane technique have been successfully applied to many optimization problems that can be categorized as partition problems. A few examples are 3-COLORING [18], MAX-3-CUT [7,11,13], MAX-BISECTION [16], CORRELATION CLUSTERING [5,33], and SPARSEST CUT [2]. Additionally, some progress has been made in extending semidefinite programming techniques outside the domain of graph partitioning to problems such as BETWEENNESS [6], BANDWIDTH [4], and LINEAR EQUATIONS mod p [1].

Cross-References

- Exact Algorithms for Maximum Two-Satisfiability
- Graph Bandwidth
- Graph Coloring
- Sparsest Cut

Recommended Reading

1. Andersson G, Engebretsen L, Håstad J (2001) A new way to use semidefinite programming with applications to linear equations mod p. J Algorithms 39:162–204

2. Arora S, Rao S, Vazirani U (2004) Expander flows, geometric embeddings and graph partitioning. In: Proceedings of the 36th annual symposium on the theory of computing (STOC), Chicago, pp 222–231
3. Barahona F (1993) On cuts and matchings in planar graphs. Math Program 60:53–68
4. Blum A, Konjevod G, Ravi R, Vempala S (2000) Semi-definite relaxations for minimum bandwidth and other vertex-ordering problems. Theor Comput Sci 235:25–42
5. Charikar M, Guruswami V, Wirth A (2003) Clustering with qualitative information. In: Proceedings of the 44th annual IEEE symposium on foundations of computer science (FOCS), Boston, pp 524–533
6. Chor B, Sudan M (1998) A geometric approach to betweeness. SIAM J Discret Math 11:511–523
7. de Klerk E, Pasechnik DV, Warners JP (2004) On approximate graph colouring and MAX-k-CUT algorithms based on the θ function. J Comb Optim 8(3):267–294
8. Delorme C, Poljak S (1993) Laplacian eigenvalues and the maximum cut problem. Math Program 62:557–574
9. Delorme C, Poljak S (1993) The performance of an eigenvalue bound in some classes of graphs. Discret Math 111:145–156. Also appeared in Proceedings of the conference on combinatorics, Marseille, 1990
10. Feige U, Schechtman G (2002) On the optimality of the random hyperplane rounding technique for MAX-CUT. Random Struct Algorithms 20(3):403–440
11. Frieze A, Jerrum MR (1997) Improved approximation algorithms for MAX-k-CUT and MAX BISECTION. Algorithmica 18:61–77
12. Goemans MX, Williamson DP (1995) Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J ACM 42:1115–1145
13. Goemans MX, Williamson DP (2004) Approximation algorithms for MAX-3-CUT and other problems via complex semidefinite programming. STOC 2001 Spec Issue J Comput Syst Sci 68:442–470
14. Grötschel M, Lovász L, Schrijver A (1981) The ellipsoid method and its consequences in combinatorial optimization. Combinatorica 1:169–197
15. Grötschel M, Lovász L, Schrijver A (1988) Geometric algorithms and combinatorial optimization. Springer, Berlin
16. Halperin E, Zwick U (2002) A unified framework for obtaining improved approximation algorithms for maximum graph bisection problems. Random Struct Algorithms 20(3):382–402
17. Håstad J (2001) Some optimal inapproximability results. J ACM 48:798–869
18. Karger DR, Motwani R, Sudan M (1998) Improved graph coloring via semidefinite programming. J ACM 45(2):246–265
19. Karloff HJ (1999) How good is the Goemans-Williamson MAX CUT algorithm? SIAM J Comput 29(1):336–350
20. Karp RM (1972) Reducibility among combinatorial problems. In: Complexity of computer computations. Plenum, New York, pp 85–104
21. Khot S (2002) On the power of unique 2-prover 1-round games. In: Proceedings of the 34th annual symposium on the theory of computing (STOC), Montreal, pp 767–775
22. Khot S, Kindler G, Mossel E, O'Donnell R (2004) Optimal inapproximability results for MAX CUT and other 2-variable CSPs? In: Proceedings of the 45th annual IEEE symposium on foundations of computer science (FOCS), Rome, pp 146–154
23. Mahajan R, Hariharan R (1995) Derandomizing semidefinite programming based approximation algorithms. In: Proceedings of the 36th annual IEEE symposium on foundations of computer science (FOCS), Milwaukee, pp 162–169
24. Mohar B, Poljak S (1990) Eigenvalues and the max-cut problem. Czech Math J 40(115): 343–352
25. Newman A (2004) A note on polyhedral relaxations for the maximum cut problem. Unpublished manuscript
26. Poljak S (1992) Polyhedral and eigenvalue approximations of the max-cut problem. Sets Graphs Numbers Colloqiua Mathematica Societatis Janos Bolyai 60:569–581
27. Poljak S, Rendl F (1994) Node and edge relaxations of the max-cut problem. Computing 52:123–137
28. Poljak S, Rendl F (1995) Nonpolyhedral relaxations of graph-bisection problems. SIAM J Optim 5:467–487
29. Poljak S, Rendl F (1995) Solving the max-cut using eigenvalues. Discret Appl Math 62(1–3):249–278
30. Poljak S, Tuza Z (1995) Maximum cuts and large bipartite subgraphs. DIMACS Ser Discret Math Theor Comput Sci 20:181–244
31. Sahni S, Gonzalez T (1976) P-complete approximation problems. J ACM 23(3):555–565
32. Soto JA (2015) Improved analysis of a max-cut algorithm based on spectral partitioning. SIAM J Discret Math 29(1):259–268
33. Swamy C (2004) Correlation clustering: maximizing agreements via semidefinite programming. In: Proceedings of 15th annual ACM-SIAM symposium on discrete algorithms (SODA), New Orleans, pp 526–527
34. Trevisan L (2012) Max cut and the smallest eigenvalue. SIAM J Comput 41(6):1769–1786
35. Trevisan L, Sorkin GB, Sudan M, Williamson DP (2000) Gadgets, approximation, and linear programming. SIAM J Comput 29(6): 2074–2097

Max Leaf Spanning Tree

Frances Rosamond
Parameterized Complexity Research Unit, University of Newcastle, Callaghan, NSW, Australia

Keywords

Connected dominating set; Extremal structure; Maximum leaf spanning tree

Years and Authors of Summarized Original Work

2005; Estivill-Castro, Fellows, Langston, Rosamond

Problem Definition

The MAX LEAF SPANNING TREE problem asks us to find a spanning tree with at least k leaves in an undirected graph. The decision version of parameterized MAX LEAF SPANNING TREE is the following:

MAX LEAF SPANNING TREE
INPUT: A connected graph G, and an integer k.
PARAMETER: An integer k.
QUESTION: Does G have a spanning tree with at least k leaves?

The parameterized complexity of the nondeterministic polynomial-time complete MAX LEAF SPANNING TREE problem has been extensively studied [2, 3, 9, 11] using a variety of kernelization, branching and other fixed-parameter tractable (FPT) techniques. The authors are the first to propose an extremal structure method for hard computational problems. The method, following in the sense of Grothendieck and in the spirit of the graph minors project of Robertson and Seymour, is that a mathematical project should unfold as a series of small steps in an overall trajectory that is described by the appropriate "mathematical machine." The authors are interested in statements of the type: Every connected graph on n vertices that satisfies a certain set of properties has a spanning tree with at least k leaves, and this spanning tree can be found in time $O(f(k) + n^c)$, where c is a constant (independent of k) and f is an arbitrary function.

In parameterized complexity, the value k is called the *parameter* and is used to capture some structure of the input or other aspect of the computational objective. For example, k might be the number of edges to be deleted in order to obtain a graph with no cycles, or k might be the number of DNA sequences to be aligned in an alignment, or k may be the maximum type-declaration nesting depth of a compiler, or $k = 1/\epsilon$ may be the parameterization in the analysis of approximation, or k might be a composite of several variables.

There are two important ways of comparing FPT algorithms, giving rise to two FPT *races*. In the "$f(k)$" race, the competition is to find ever more slowing growing parameter functions $f(k)$ governing the complexity of FPT algorithms. The "kernelization race" refers to the following lemma stating that a problem is in FPT if and only if the input can be preprocessed (*kernelized*) in "ordinary" polynomial time into an instance whose size is bounded by a function of k only.

Lemma 1 *A parameterized problem* Π *is in FPT if and only if there is a polynomial-time transformation* (*in both n and k*) *that takes* (x, k) *to* (x', k') *such that:*

(1) (x, k) *is a yes-instance of* Π *if and only if* (x', k') *is a yes-instance of* Π,
(2) $k' \leq k$, *and*
(3) $|x'| \leq g(k)$ *for some fixed function* g.

In the situation described by the lemma, say that we can *kernelize* to instances of size at most $g(k)$. Although the two races are often closely related, the result is not always the same. The current best FPT algorithm for MAX LEAF is due to Bonsma [1] (following the extremal structure approach outlined by the authors) with a running

time of $O^*(8.12^k)$ to determine whether a graph G on n vertices has a spanning tree with at least k leaves; however the authors present the FPT algorithm with the smallest kernel size.

The authors list five independent deliverables associated to the extremal structure theory, and illustrate all of the objectives for the MAX LEAF problem. The five objectives are:

(A) Better FPT algorithms as a result of deeper structure theory, more powerful reduction rules associated with that structure theory, and stronger inductive proofs of improved kernelization bounds.
(B) Powerful preprocessing (*data reduction/kernelization*) rules and *combinations* of rules that can be used regardless of whether the parameter is small and that can be combined with other approaches, such as approximation and heuristics. These are usually easy to program.
(C) Gradients and transformation rules for local search heuristics.
(D) Polynomial-time approximation algorithms and performance bounds proved in a systematic way.
(E) Structure to exploit for solving other problems.

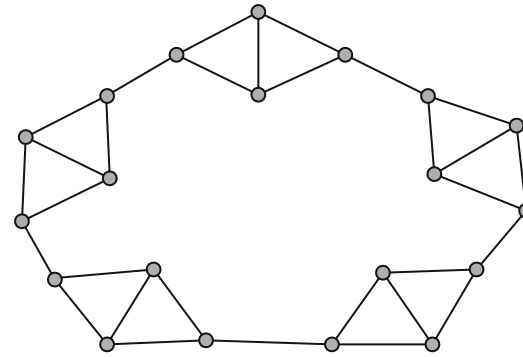
Key Results

The key results are programmatic, providing a *method of extremal structure* as a systematic method for designing FPT algorithms. The five interrelated objectives listed above are surveyed, and each is illustrated using the MAX LEAF SPANNING TREE problem.

Objective A: FPT Algorithms

The objective here is to find polynomial-time preprocessing (*kernelization*) rules where $g(k)$ is as small as possible. This has a direct payoff in terms of program objective B.

Rephrased as a structure theory question, the crucial issue is: *What is the structure of graphs that do not have a subgraph with k leaves?*



Max Leaf Spanning Tree, Fig. 1 Reduction rules were developed in order to reduce this Kleitman–West graph structure

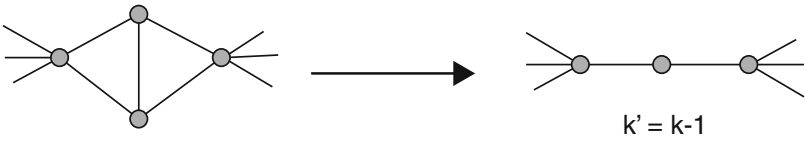
A graph theory result due to Kleitman and West shows that a graph of minimum degree at least 3, that excludes a k-leaf subgraph, has at most $4(k-3)$ vertices. Figure 1 shows that this is the best possible result for this hypothesis. However, investigating the structure using extremal methods reveals the need for the reduction rule of Fig. 2. About 20 different polynomial-time reduction rules (some much more complex and "global" in structure than the simple local reduction rule depicted) are sufficient to kernelize to a graph of minimum degree 2 having at most $3.5k$ vertices.

In general, an instance of a parameterized problem consists of a pair (x, k) and a "boundary" which is located by holding x fixed and varying k and regarding whether the outcome of the decision problem is *yes* or *no*. Of interest is the boundary when x is reduced. A typical boundary lemma looks like the following.

Lemma 2 *Suppose* (G, k) *is a reduced instance of* MAX LEAF, WITH (G, k) *a yes-instance and* $(G, k+1)$ *a no-instance. Then* $|G| \leq ck$. *(Here c is a small constant that becomes clarified during the investigation.)*

A proof of a boundary lemma is by minimum counterexample. A counterexample would be a graph such that (1) (G, k) is reduced, (2) (G, k) is a *yes*-instance of MAX LEAF, (3) $(G, k+1)$ is a *no*-instance, and (4) $|G| > ck$.

The proof of a boundary lemma unfolds gradually. Initially, it is not known what bound will



Max Leaf Spanning Tree, Fig. 2 A reduction rule for the Kleitman–West graph

eventually succeed and it is not known exactly what is meant by *reduced*. In the course of an attempted proof, these details are worked out. As the arguments unfold, structural situations will suggest new reduction rules. Strategic choices involved in a boundary lemma include:

(1) Determining the polarity of the boundary, and setting up the boundary lemma.
(2) Choosing a witness structure.
(3) Setting inductive priorities.
(4) Developing a series of structural claims that describe the situation at the boundary.
(5) Discovering reduction rules that can act in polynomial-time on relevant structural situations at the boundary.
(6) As the structure at the boundary becomes clear, filling in the blank regarding the kernelization bound.

The overall structure of the argument is "by minimum counterexample" according to the priorities established by choice 3, which generally make reference to choice 2. The proof proceeds by a series of small steps consisting of structural claims that lead to a detailed structural picture at the "boundary"– and thereby to the bound on the size of G that is the conclusion of the lemma. The complete proof assembles a series of claims made against the witness tree, various sets of vertices, and inductive priorities and sets up a master inequality leading to a proof by induction, and a $3.5k$ problem kernel.

Objective B: Polynomial-Time Preprocessingand Data-Reduction Routines

The authors have designed a table for tracing each possible *boundary state* for a possible solution. Examples are given that show the surprising power of cascading data-reduction rules on real input distributions and that describe a variety of mathematical phenomena relating to reduction rules. For example, some reduction rules, such as the *Kleitman–West dissolver rule* for MAX LEAF (Fig. 2), have a fixed "boundary size" (in this case 2), whereas crown-type reduction rules do not have a fixed boundary size.

Objective C: Gradients and Solution Transformations for Local Search

A generalization of the usual setup for local search is given, based on the mathematical power of the more complicated gradient in obtaining superior kernelization bounds. Idea 1 is that local search be conducted based on maintaining a "current witness structure" rather than a full solution (spanning tree). Idea 2 is to use the list of inductive priorities to define a "better solution" gradient for the local search.

Objective D: Polynomial-Time Approximation Algorithms

The polynomial-time extremal structure theory leads directly to a constant-factor p-time approximation algorithm for MAX LEAF. First, reduce G using the kernelization rules. The rules are approximation-preserving. Take *any* tree T (not necessarily spanning) in G. If all of the structural claims hold, then (by the boundary lemma arguments) the tree T must have at least n/c leaves for $c = 3.75$. Therefore, lifting T back along the reduction path, we obtain a c-approximation.

If at least one of the structural claims does not hold, then the tree T can be improved against one of the inductive priorities. Notice that each claim is proved by an argument that can be interpreted as a polynomial-time routine that improves T, when the claim is contradicted.

These consequences can be applied to the original T (and its successors) only a polynomial number of times (determined by the list of

Max Leaf Spanning Tree, Table 1 The complexity ecology of parameters

	TW	BW	VC	DS	G	ML
TW	*FPT*	*W*[1]-hard	*FPT*	*FPT*	?	*FPT*
BW	*FPT*	*W*[1]-hard	*FPT*	*FPT*	?	*FPT*
VC	*FPT*	?	*FPT*	*FPT*	?	*FPT*
DS	?	?	*W*[1]-hard	*W*[1]-hard	?	?
G	*W*[1]-hard	*W*[1]-hard	*W*[1]-hard	*W*[1]-hard	*FPT*	?
ML	*FPT*	?	*FPT*	*FPT*	*FPT*	?

inductive priorities) until one arrives at a tree T' for which all of the various structural claims hold. At that point, we must have a c-approximate solution.

Objective E: Structure To Exploitin The Ecology of Complexity

The objective here is to understand how every input-governing problem parameter affects the complexity of every other problem. As a small example, consider Table 1 using the shorthand TW is TREEWIDTH, BW is BANDWIDTH, VC is VERTEX COVER, DS is DOMINATING SET, G is GENUS and ML is MAX LEAF. The entry in the second row and fourth column indicates that there is an *FPT* algorithm to optimally solve the DOMINATING SET problem for a graph G of bandwidth at most k. The entry in the fourth row and second column indicates that it is unknown whether BANDWIDTH can be solved optimally by an *FPT* algorithm when the parameter is a bound on the domination number of the input.

MAX LEAF applies to the last row of the table. For graphs of *max leaf number* bounded by k, the maximum size of an independent set can be computed in time $O^*(2.972^k)$ based on a reduction to a kernel of size at most $7k$. There is a practical payoff for using the output of one problem as the input to another.

Applications

The MAX LEAF SPANNING TREE problem has motivations in computer graphics for creating triangle strip representations for fast interactive rendering [5]. Other applications are found in the area of traffic grooming and network design, such as the design of optical networks and the utilization of wavelengths in order to minimize network cost, either in terms of the line-terminating equipment deployed or in terms of electronic switching [6]. The minimum-energy problem in wireless networks consists of finding a transmission radius vector for all stations in such a way that the total transmission power of the whole network is the least possible. A restricted version of this problem is equivalent to the MAX LEAF SPANNING TREE problem [7]. Finding spanning trees with many leaves is equivalent to finding small connected dominating sets and is also called the MINIMUM CONNECTED DOMINATING problem [13].

Open Problems

Branching Strategies

While extremal structure is in some sense *the right way* to design an FPT algorithm, this is not the only way. In particular, the recipe is silent on what to do with the kernel. An open problem is to find general strategies for employing "parameter-appropriate structure theory" in branching strategies for sophisticated problem kernel analysis.

Turing Kernelizability

The polynomial-time transformation of (x, k) to the simpler reduced instance (x', k') is a many:1 transformation. One can generalize the notion of many:1 reduction to Turing reduction. How should the quest for p-time extremal theory unfold under this "more generous" FPT?

Algorithmic Forms of The Boundary Lemma Approach

The hypothesis of the boundary lemma that (G, k) is a *yes*-instance implies that there exists a witness structure to this fact. There is no assumption that one has algorithmic access to this structure, and when reduction rules are discovered, these have to be transformations that can be applied to (G, k) and a structure that can be discovered in (G, k) in polynomial time. In other words, reduction rules cannot be *defined* with respect to the witness structure. Is it possible to describe more general approaches to kernelization where the witness structure used in the proof of the boundary lemma is polynomial-time computable, and this structure provides a conditional context for some reduction rules? How would this change the extremal method recipe?

Problem Annotation

One might consider a generalized MAX LEAF problem where vertices and edges have various annotations as to whether they *must* be leaves (or internal vertices) in a solution, etc. Such a generalized form of the problem would generally be expected to be "more difficult" than the vanilla form of the problem. However, several of the "best known" FPT algorithms for various problems, are based on these generalized, annotated forms of the problems. Examples include PLANAR DOMINATING SET and FEEDBACK VERTEX SET [4]. Should annotation be part of the recipe for the best possible polynomial-time kernelization?

Cross-References

- ▸ Connected Dominating Set
- ▸ Data Reduction for Domination in Graphs

Recommended Reading

1. Bonsma P (2006) Spanning trees with many leaves: new extremal results and an improved FPT algorithm, vol 1793. Memorandum Department of Applied Mathematics, University of Twente, Enschede
2. Bonsma P, Brueggemann T, Woeginger G (2003) A faster FPT algorithm for finding spanning trees with many leaves. In: Proceedings of MFCS 2003. Lecture notes in computer science, vol 2747. Springer, Berlin, pp 259–268
3. Downey RG, Fellows MR (1999) Parameterized complexity, Monographs in computer science. Springer, New York
4. Dehne F, Fellows M, Langston M, Rosamond F, Stevens K (2005) An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. In: Proceedings COCOON 2005. Lecture notes in computer science, vol 3595. Springer, Berlin, pp 859–869
5. Diaz-Gutierrez P, Bhushan A, Gopi M, Pajarola R (2006) Single-strips for fast interactive rendering. J Vis Comput 22(6):372–386
6. Dutta R, Savage C (2005) A Note on the complexity of converter placement supporting broadcast in WDM optical networks. In: Proceedings of the international conference on telecommunication systems-modeling and analysis, Dallas, Nov 2005, pp 23–31. ISBN: 0-9716253-3-6. American Telecommunication Systems Management Association, Nashville
7. Egecioglu O, Gonzalez T (2001) Minimum-energy broadcast in simple graphs with limited node power. In: Proceedings of the IASTED international conference on parallel and distributed computing and systems (PDCS), Anaheim, Aug 2001, pp 334–338
8. Estivill-Castro V, Fellows MR, Langston MA, Rosamond FA (2005) FPT is P-time extremal structure I. In: Algorithms and complexity in Durham 2005. Texts in algorithmics, vol 4. Kings College Publications, London, pp 1–41
9. Fellows M, Langston M (1992) On well-partial-order theory and its applications to combinatorial problems of VLSI design. SIAM J Discret Math 5:117–126
10. Fellows M (2003) Blow-ups, win/win's and crown rules: some new directions in FPT. In: Proceedings of the 29th workshop on graph theoretic concepts in computer science (WG 2003). Lecture notes in computer science, vol 2880. Springer, Berlin, pp 1–12
11. Fellows M, McCartin C, Rosamond F, Stege U (2000) Coordinatized kernels and catalytic reductions: an improved FPT algorithm for max leaf spanning tree and other problems. In: Proceedings of the 20th conference on foundations of software technology and theoretical computer science (FST-TCS 2000). Lecture notes in theoretical computer science, vol 1974. Springer, Berlin, pp 240–251
12. Kleitman DJ, West DB (1991) Spanning trees with many leaves. SIAM J Discret Math 4:99–106
13. Kouider M, Vestergaard PD (2006) Generalized connected domination in graphs. Discret Math Theory Comput Sci (DMTCS) 8:57–64
14. Lu H-I, Ravi R (1998) Approximating maximum leaf spanning trees in almost linear time. J Algorithm 29:132–141

15. Niedermeier R (2006) Invitation to fixed parameter algorithms. Lecture series in mathematics and its applications. Oxford University Press, Oxford
16. Prieto-Rodriguez E (2005) Systematic kernelization in FPT algorithm design. Dissertation, School of Electrical Engineering and Computer Science, University of Newcastle
17. Solis-Oba R (1998) 2-approximation algorithm for finding a spanning tree with the maximum number of leaves. In: Proceedings of the 6th annual European symposium on algorithms (ESA'98). Lecture notes in computer science, vol 1461. Springer, Berlin, pp 441–452

Maximizing the Minimum Machine Load

Csanad Imreh
Institute of Informatics, University of Szeged, Szeged, Hungary

Keywords

Approximation algorithms; Multiprocessor systems; Scheduling

Years and Authors of Summarized Original Work

1982; Deuermeyer, Friesen, Langston
1997; Woeginger
1998; Azar, Epstein

Problem Definition

In a scheduling problem we have to find an optimal schedule of jobs. Here we consider the parallel machines case, where m machines are given, and we can use them to schedule the jobs. In the most fundamental model, each job has a known processing time, and to schedule the job we have to assign it to a machine, and we have to give its starting time and a completion time, where the difference between the completion time and the starting time is the processing time. No machine may simultaneously run two jobs. If no further assumptions are given then the machines can schedule the jobs assigned to them without an idle time and the total time required to schedule the jobs on a machine is the sum of the processing times of the jobs assigned to it. We call this value the load of the machine.

Concerning the machine environment three different models are considered. If the processing time of a job is the same for each machine, then we call the machines identical machines. If each machine has a speed s_i, the jobs have a processing weight p_j and the processing time of job j on the i-th machine is p_j/s_i, then we call the machines related machines. If the processing time of job j is given by an arbitrary positive vector $P_j = (p_j(1), \ldots, p_j(m))$, where the processing time of the job on the i-th machine is $p_j(i)$, then we call the machines unrelated machines. Here we consider the identical machine case unless it is stated otherwise.

Many objective functions are considered for scheduling problems. Here we consider only such models where the goal is the maximization of the minimal load which problem was proposed in [4]. We note that the most usual objective function is minimizing the maximal load which is called makespan. This objective is the dual of the makespan in some sense but both objective functions require to balance the loads of the machines.

A straightforward reduction from the well-known NP-hard partition problem shows that the investigated problem is NP-hard. Therefore one main research question is to develop polynomial time approximation algorithms which cannot ensure the optimal solution but always give a solution which is not much worse than the optimal one. These approximation algorithms are usually evaluated by the approximation ratio. In case of maximization problems an algorithm is called c-approximation if the objective value given by the algorithm is at least c-times as large than the optimal objective value. If we have a polynomial time $1 - \varepsilon$-approximation algorithm for every $\varepsilon > 0$, then this class of algorithms is called polynomial approximation scheme (PTAS in short). If these algorithms are also polynomial

in $1/\varepsilon$, then we call them fully polynomial approximation scheme (FPTAS in short).

Key Results

Approximation Algorithms

Since we plan to balance the load on the machines, a straightforward idea to find a solution is to use some greedy method to schedule the jobs. If we schedule the jobs one by one, then a greedy algorithm assigns the job to the machine with the smallest load. Unfortunately if we schedule the jobs in arbitrary order, then this algorithm does not have constant approximation ratio. In the worst inputs the problem is caused by the large jobs which are scheduled at the end of the input. Therefore, the next idea is to order the jobs by decreasing size and schedule them one by one assigning the actual job to the machine with the smallest load. This algorithm is called LPT (longest processing time) and analyzed in [4] and [3]. The first analysis was presented in [4], where the authors proved that the algorithm is $3/4$-approximation and also proved that no greater approximation ratio can be given for the algorithm as the number of machines tends to ∞. In [3] a more sophisticated analysis is given, the authors proved that the exact competitive ratio is $(3m - 1)/(4m - 2)$. Later in [7] a PTAS was presented for the problem. The time complexity of the algorithm is $O(c_\varepsilon \cdot n \cdot m)$ where c_ε is a constant which depends exponentially on ε. Thus the presented class of algorithms is not an FPTAS. But it worths noting that we cannot expect an FPTAS for the problem. It belongs to the class of strongly NP-complete problems; thus an FPTAS would yield P = NP. The case of unrelated machines is much more difficult. In [2] it is proved that no better than $1/2$ approximation algorithm exists unless P = NP; therefore we cannot expect a PTAS in this case.

Online and Semi-online Problem

In many applications we do not have a priori knowledge about the input and the algorithms must make their decision online based only on the past information. These algorithms are called online algorithms. In scheduling problems this means that the jobs arrive one by one and the algorithm has to schedule the arriving job without any knowledge about the further ones. This area is very similar to the area of approximation algorithms, again we cannot expect algorithms which surely find optimal solutions. In approximation algorithms the problem is that we do not have exponential time for computation; in online algorithms it is the lack of information. The algorithms are also analyzed by a similar method, but in the area of online algorithms it is called competitive ratio. For maximization problems an online algorithm is called c-competitive if the objective value given by the algorithm is at least c-times as large than the optimal objective value.

The online version of scheduling maximizing the minimal load is studied in [1]. The most straightforward online algorithm is the above-mentioned List algorithm which assigns the actual job to the machine with the smallest load. It is $1/m$-competitive and it is easy to see (considering m jobs of size 1 and if they are assigned to different machines $m-1$ further jobs of size m) that no better deterministic online algorithm can be given. In [1] randomized algorithms are studied, the authors presented an $1/O(\sqrt{m}\log m)$-competitive randomized algorithm and proved that no randomized algorithm can have better competitive ratio than $1/\Omega(\sqrt{m})$. The case of related machines is also studied and it is proved that no algorithm exists which has a competitive ratio depending on the number of machines.

In semi-online problems usually some extra information is given to the algorithm. The first such model is also studied in [1]. The authors studied the version where the optimal value is known in advance and they presented an $m/(2m - 1)$-competitive algorithm and they proved that if $m = 2$ or $m = 3$ then no semi-online algorithm in this model with better competitive ratio exists. In case of related machines and known optimal value, an $1/m$-competitive algorithm is given. Several further semi-online version is studied in the literature. In [5] the semi-online version where the maximal job size is known in advance, in [6] the version where total processing time of all jobs and the largest processing time is known in advance is studied.

Applications

The first paper [4] mentions an application as the motivation of the model. It is stated that the problem was motivated by modeling the sequencing of maintenance actions for modular gas turbine aircraft engines. If a fleet of M identical machines (engines) are given and they must be kept operational for as long as possible, then we obtain the objective to maximize the minimal load.

Cross-References

▸ List Scheduling

Recommended Reading

1. Azar Y, Epstein L (1998) On-line machine covering. J Sched 1(2):67–77
2. Bezáková I, Dani V (2005) Nobody left behind: fair allocation of indivisible goods. ACM SIGecom Exch 5.3
3. Csirik J, Kellerer H, Woeginger GJ (1992) The exact LPT-bound for maximizing the minimum completion time. Oper Res Lett 11:281–287
4. Deuermeyer BL, Friesen DK, Langston MA (1982) Scheduling to maximize the minimum processor finish time in a multiprocessor system. SIAM J Discret Methods 3:190–196
5. He Y, Zhang GC (1999) Semi on-line scheduling on two identical machines. Computing 62:179–187
6. Tan Z, Wu Y (2007) Optimal semi-online algorithms for machine covering. Theor Comput Sci 372: 69–80
7. Woeginger GJ (1997) A polynomial time approximation scheme for maximizing the minimum machine completion time. Oper Res Lett 20(4):149–154

Maximum Agreement Subtree (of 2 Binary Trees)

Ramesh Hariharan
Strand Life Sciences, Bangalore, India

Keywords

Isomorphism; Tree agreement

Years and Authors of Summarized Original Work

1996; Cole, Hariharan
2002; Farach, Hariharan, Przytycka, Thorup

Problem Definition

Consider two rooted trees T_1 and T_2 with n leaves each. The internal nodes of each tree have at least two children each. The leaves in each tree are labeled with the same set of labels, and further, no label occurs more than once in a particular tree. An *agreement subtree* of T_1 and T_2 is defined as follows. Let L_1 be a subset of the leaves of T_1 and let L_2 be the subset of those leaves of T_2 which have the same labels as leaves in L_1. The subtree of T_1 *induced* by L_1 is an agreement subtree of T_1 and T_2 if and only if it is *isomorphic* to the subtree of T_2 induced by L_2. The maximum agreement subtree problem (henceforth called *MAST*) asks for the largest agreement subtree of T_1 and T_2.

The terms *induced subtree* and *isomorphism* used above need to be defined. Intuitively, the subtree of T induced by a subset L of the leaves of T is the topological subtree of T restricted to the leaves in L, with branching information relevant to L preserved. More formally, for any two leaves a, b of a tree T, let $\text{lca}_T(a, b)$ denote their lowest common ancestor in T. If $a = b$, $\text{lca}_T(a, b) = a$. The *subtree* U of T *induced* by a subset L of the leaves is the tree with leaf set L and interior node set $\{\text{lca}_T(a, b) \mid a, b \in L\}$ inheriting the ancestor relation from T, that is, for all $a, b \in L$, $\text{lca}_U(a, b) = \text{lca}_T(a, b)$.

Intuitively, two trees are isomorphic if the children of each node in one of the trees can be reordered so that the leaf labels in each tree occur in the same order and the shapes of the two trees become identical. Formally, two trees U_1 and U_2 with the same leaf labels are said to be *isomorphic* if there is a 1–1 mapping μ between their nodes, mapping leaves to leaves with the same labels, and such that for any two different leaves a, b of U_1, $\mu(\text{lca}_{U1}(a, b)) = \text{lca}_{U2}(\mu(a), \mu(b))$.

Key Results

Previous Work

Finden and Gordon [8] gave a heuristic algorithm for the *MAST* problem on binary trees which had an $O(n^5)$ running time and did not guarantee an optimal solution. Kubicka, Kubicki, and McMorris [12] gave an $O(n^{(5+\epsilon)\log n})$ algorithm for the same problem. The first polynomial time algorithm for this problem was given by Steel and Warnow [14]; it had a running time of $O(n^2)$. Steel and Warnow also considered the case of nonbinary and unrooted trees. Their algorithm takes $O(n^2)$ time for fixed-degree rooted and unrooted trees and $O(n^{4.5}\log n)$ for arbitrary-degree rooted and unrooted trees. They also give a linear reduction from the rooted to the unrooted case. Farach and Thorup gave an $O\left(nc^{\sqrt{\log n}}\right)$ time algorithm for the *MAST* problem on binary trees; here c is a constant greater than 1. For arbitrary-degree trees, their algorithm takes $O\left(n^2c^{\sqrt{\log n}}\right)$ time for the unrooted case [6] and $O(n^{1.5}\log n)$ time for the rooted case [7]. Farach, Przytycka, and Thorup [4] obtained an $O(n\log^3 n)$ algorithm for the *MAST* problem on binary trees. Kao [11] obtained an algorithm for the same problem which takes $O(n\log^2 n)$ time. This algorithm takes $O(\min\{nd^2\log d\log^2 n, nd^{\frac{3}{2}}\log^3 n\})$ for degree d trees.

The *MAST* problem for more than two trees has also been studied. Amir and Keselman [1] showed that the problem is *NP*-hard for even 3 unbounded degree trees. However, polynomial bounds are known [1, 5] for three or more bounded degree trees.

Our Contribution

An $O(n\log n)$ algorithm for the *MAST* problem for two binary trees is presented here. This algorithm is obtained by improving upon the $O(n\log^3 n)$ algorithm from [4] (in fact, the final journal version [3] combines both papers). The $O(n\log^3 n)$ algorithm of [4] can be viewed as taking the following approach (although the authors do not describe it this way). It identifies two special cases and then solves the general case by interpolating between these cases.

Special Case 1

The internal nodes in both trees form a path. The *MAST* problem reduces to essentially a size n Longest Increasing Subsequence problem in this case. As is well known, this can be solved in $O(n\log n)$ time.

Special Case 2

Both trees T_1 and T_2 are complete binary trees. For each node v in T_2, only certain nodes u in T_1 can be usefully mapped to v, in the sense that the subtree of T_1 rooted at u and the subtree of T_2 rooted at v have a nonempty agreement subtree. There are $O(n\log^2 n)$ such pairs (u, v). This can be seen as follows. Note that for (u, v) to be such a pair, the subtree of T_1 rooted at u and the subtree of T_2 rooted at v must have a leaf label in common. For each label, there are only $O(\log^2 n)$ such pairs obtained by pairing each ancestor of the leaf with this label in T_1 with each ancestor of the leaf with this label in T_2. The total number of interesting pairs is thus $O(n\log^2 n)$. For each pair, computing the *MAST* takes $O(1)$ time, as it is simply a question of deciding the best way of pairing their children.

The interpolation process takes a centroid decomposition of the two trees and compares pairs of centroid paths, rather than individual nodes as in the complete tree case. The comparison of a pair of centroid paths requires finding matchings with special properties in appropriately defined bipartite graphs, a nontrivial generalization of the Longest Increasing Subsequence problem. This process creates $O(n\log^2 n)$ interesting (u, v) pairs, each of which takes $O(\log n)$ time to process.

This work provides two improvements, each of which gains a $\log n$ factor.

Improvement 1

The complete tree special case is improved to $O(n\log n)$ time as follows. A pair of nodes $(u, v), u \in T_1, v \in T_2$, is said to be *interesting* if there is an agreement subtree mapping u to v. As is shown below, for complete trees, the

total number of interesting pairs (u, v) is just $O(n \log n)$. Consider a node v in T_2. Let L_2 be the set of leaves which are descendants of v. Let L_1 be the set of leaves in T_1 which have the same labels as the leaves in L_2. The only nodes that may be mapped to v are the nodes u in the subtree of T_1 induced by L_1. The number of such nodes u is $O(\text{size of the subtree of } T_2 \text{ rooted at } v)$. The total number of interesting pairs is thus the sum of the sizes of all subtrees of T_2, which is $O(n \log n)$.

This reduces the number of interesting pairs (u, v) to $O(n \log n)$. Again, processing a pair takes $O(1)$ time (this is less obvious, for identifying the descendants of u which root the subtrees with which the two subtrees of v can be matched is nontrivial). Constructing the above induced subtree itself can be done in $O(|L_1|)$ time, as will be detailed later. The basic tool here is to preprocess trees T_1 and T_2 in $O(n)$ time so that the least common ancestor queries can be answered in $O(1)$ time.

Improvement 2

As in [4], when the trees are not complete binary trees, the algorithm takes centroid paths and matches pairs of centroid paths. The $O(\log n)$ cost that the algorithm in [4] incurs in processing each such interesting pair of paths arises when there are large (polynomial in n size) instances of the generalized Longest Increasing Subsequence problem. At first sight, it is not clear that large instances of these problems can be created for sufficiently many of the interesting pairs; unfortunately, this turns out to be the case. However, these problem instances still have some useful structure. By using (static) weighted trees, pairs of interesting vertices are processed in $O(1)$ time per pair, on the average, as is shown by an appropriately parametrized analysis.

The Multiple Degree Case

The techniques can be generalized to higher degree bounds $d > 2$ by combining it with techniques from [6, Sect. 2] for unbounded degrees. This appears to yield an algorithm with running time $O(\min\{n\sqrt{d}\log^2 n, nd \log n \log d\})$. The conjecture, however, is that there is an algorithm with running time $O(n\sqrt{d}\log n)$.

Applications

Motivation

The *MAST* problem arises naturally in biology and linguistics as a measure of consistency between two evolutionary trees over species and languages, respectively. An evolutionary tree for a set of *taxa*, either species or languages, is a rooted tree whose leaves represent the taxa and whose internal nodes represent ancestor information. It is often difficult to determine the true phylogeny for a set of taxa, and one way to gain confidence in a particular tree is to have different lines of evidence supporting that tree. In the biological taxa case, one may construct trees from different parts of the DNA of the species. These are known as *gene trees*. For many reasons, these trees need not entirely agree, and so one is left with the task of finding a consensus of the various gene trees. The maximum agreement subtree is one method of arriving at such a consensus. Notice that a gene is usually a binary tree, since DNA replicates by a binary branching process. Therefore, the case of binary trees is of great interest.

Another application arises in automated translation between two languages (Grishman and Yangarber, NYU, Private Communication). The two trees are the parse trees for the same meaning sentences in the two languages. A complication that arises in this application (due in part to imperfect dictionaries) is that words need not be uniquely matched, i.e., a word at the leaf of one tree could match a number (usually small) of words at the leaves of the other tree. The aim is to find a maximum agreement subtree; this is done with the goal of improving context-using dictionaries for automated translation. So long as each word in one tree has only a constant number of matches in the other tree (possibly with differing weights), the algorithm given here can be used, and its performance remains $O(n \log n)$. More generally, if there are m word matches in all, the performance becomes $O((m+n)\log n)$. Note, however, that if there are two

collections of equal-meaning words in the two trees of sizes k_1 and k_2 respectively, they induce $k_1 k_2$ matches.

Cross-References

- Maximum Agreement Subtree (of 3 or More Trees)
- Maximum Agreement Supertree

Recommended Reading

1. Amir A, Keselman D (1997) Maximum agreement subtree in a set of evolutionary trees. SIAM J Comput 26(6):1656–1669
2. Cole R, Hariharan R (1996) An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. In: Proceedings of 7th ACM-SIAM SODA, Atlanta, pp 323–332
3. Cole R, Farach-Colton M, Hariharan R, Przytycka T, Thorup M (2000) An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. SIAM J Comput 30(5):1385–1404
4. Farach M, Przytycka T, Thorup M (1995) The maximum agreement subtree problem for binary trees. In: Proceedings of 2nd ESA
5. Farach M, Przytycka T, Thorup M (1995) Agreement of many bounded degree evolutionary trees. Inf Process Lett 55(6):297–301
6. Farach M, Thorup M (1995) Fast comparison of evolutionary trees. Inf Comput 123(1):29–37
7. Farach M, Thorup M (1997) Sparse dynamic programming for evolutionary-tree comparison. SIAM J Comput 26(1):210–230
8. Finden CR, Gordon AD (1985) Obtaining common pruned trees. J Classific 2:255–276
9. Fredman ML (1975) Two applications of a probabilistic search technique: sorting $X + Y$ and building balanced search trees. In: Proceedings of the 7th ACM STOC, Albuquerque, pp 240–244
10. Harel D, Tarjan RE (1984) Fast algorithms for finding nearest common ancestors. SIAM J Comput 13(2):338–355
11. Kao M-Y (1998) Tree contractions and evolutionary trees. SIAM J Comput 27(6):1592–1616
12. Kubicka E, Kubicki G, McMorris FR (1995) An algorithm to find agreement subtrees. J Classific 12:91–100
13. Mehlhorn K (1977) A best possible bound for the weighted path length of binary search trees. SIAM J Comput 6(2):235–239
14. Steel M, Warnow T (1993) Kaikoura tree theorems: computing the maximum agreement subtree. Inf Process Lett 48:77–82

Maximum Agreement Subtree (of 3 or More Trees)

Teresa M. Przytycka
Computational Biology Branch, NCBI, NIH, Bethesda, MD, USA

Keywords

Tree alignment

Years and Authors of Summarized Original Work

1995; Farach, Przytycka, Thorup

Problem Definition

The maximum agreement subtree problem for k trees (k-MAST) is a generalization of a similar problem for two trees (MAST). Consider a tuple of k rooted leaf-labeled trees $(T_1, T_2 \ldots T_k)$. Let $A = \{a_1, a_2, \ldots a_n\}$ be the set of leaf labels. Any subset $B \subseteq A$ uniquely determines the so-called topological restriction $T|B$ of the three T to B. Namely, $T|B$ is the topological subtree of T spanned by all leaves labeled with elements from B and the lowest common ancestors of all pairs of these leaves. In particular, the ancestor relation in $T|B$ is defined so that it agrees with the ancestor relation in T. A subset B of A such $T^1 \mid B, \ldots, T^k \mid B$ are isomorphic is called an *agreement set*.

Problem 1 (k-MAST) INPUT: A tuple $\vec{T} = (T^1, \ldots, T^k)$ of leaf-labeled trees, with a common set of labels $A = \{a_1, \ldots, a_n\}$, such that for each tree T^i there exists one-to-one mapping between the set of leaves of that tree and the set of labels A.

OUTPUT: k-MAST($\vec{T}$) is equal to the maximum cardinality agreement set of $\vec{T}$.

Key Results

In the general setting, k-MAST problem is NP-complete for $k \geq 3$ [1]. Under the assumption

that the degree of at least one of the trees is bounded, Farach et al. proposed an algorithm leading to the following theorem:

Theorem 1 *If the degree of the trees in the tuple* $\vec{T} = (T^1, \ldots, T^k)$ *is bounded by* d*, then the* k-MAST$(\vec{T})$ *can be computed in* $O(kn^3 + n^d)$ *time.*

In what follows, the problem is restricted to finding the cardinality of the maximum agreement set rather than the set itself. The extension of this algorithm to an algorithm that finds the agreement set (and subsequently the agreement subtree) within the same time bounds is relatively straightforward.

Recall that the classical $O(n^2)$ dynamic programming algorithm for MAST of two binary trees [11] processes all pairs of internal nodes of the two trees in a bottom-up fashion. For each pair of such nodes, it computes the MAST value for the subtrees rooted at this pair. There are $O(n^2)$ pairs of nodes, and the MAST value for the subtrees rooted at a given pair of nodes can be computed in constant time from MAST values of previously processed pairs of nodes.

To set the stage for the more general case, let k-MAST$(\vec{v})$ be the solution to the k-MAST problem for the subtrees of $T^1(v_1), \ldots, T^k(v_k)$ where $T^i(v_i)$ is the subtree if T^i rooted at v_i. If, for all i, u_i is a strict ancestor of v_i in T^i, then, $\vec{v}$ is *dominated* by $\vec{u}$ (denoted $\vec{v} \prec \vec{u}$).

A naive extension of the algorithm for two trees to an algorithm for k trees would require computing k-MAST$(\vec{v})$ for all possible tuples $\vec{v}$ by processing these tuples in the order consistent with the domination relation. The basic idea that allows to avoid $\Omega(n^k)$ complexity is to replace the computation of k-MAST$(\vec{v})$ with the computation of a related value, mast$(\vec{v})$, defined to be the size of the maximum agreement set for the subtrees of $(T^1, \ldots, T^k)$ rooted at $(v_1, \ldots v_k)$ subject to the additional restriction that the agreement subtrees themselves are rooted at $v_1, \ldots v_k$, respectively. Clearly

$$k\text{-MAST}(T^1, \ldots, T^k) = \max_{\vec{v}} \text{mast}(\vec{v}).$$

The benefit of computing mast rather than k-MAST follows from the fact that most of mast values are zero and it is possible to identify (very efficiently) $\vec{v}$ with nonzero mast values.

Remark 1 If mast$(\vec{v}) > 0$ then $\vec{v} = (\text{lca}^{T^1}(a, b), \ldots \text{lca}^{T^k}(a, b))$ for some leaf labels a, b where $\text{lca}^{T_i}(a, b)$ is the lowest common ancestor of leaves labeled by a and b in the tree T^i.

A tuple $\vec{v}$ such that $\vec{v} = (\text{lca}^{T^1}(a, b), \ldots \text{lca}^{T^k}(a, b))$ for some $a, b \in A$ is called an *lca-tuple*. By Remark 1 it suffices to compute mast values for the lca-tuples only. Just like in the naive approach, mast$(\vec{v})$ is computed from mast values of other lca-tuples dominated by $\vec{v}$. Another important observation is that only some lca-tuples dominated by $\vec{v}$ are needed to compute mast$(\vec{v})$. To capture this, Farach et al. define the so-called proper domination relation (introduced formally below) and show that the mast value for any lca-tuple $\vec{v}$ can be computed from mast values of lca-tuples properly dominated by $\vec{v}$ and that the proper domination relation has size $O(n^3)$.

Proper Domination Relation

Index the children of each internal node of any tree in an arbitrary way. Given a pair $\vec{v}, \vec{w}$ of lca-tuples such that $\vec{w} \prec \vec{v}$ the corresponding domination relation has associated with it *direction* $\vec{\delta}_{\vec{w} \prec \vec{v}} = (\delta_1, \ldots, \delta_k)$ where w_i descends from the child of v_i indexed with δ_i. Let $v_i(j)$ be the child of v_i with index j. The direction domination is termed *active* is if the subtrees rooted at the $v_1(\delta_1), \ldots, v_k(\delta_k)$ have at least one leaf label in common. Note that each leaf label can witness only one active direction, and consequently each lca-tuple can have at most n active domination directions. Two directions $\vec{\delta}_{\vec{w} \prec \vec{v}}$ and $\vec{\delta}_{\vec{u} \prec \vec{v}}$ are called *compatible* if and only if the direction vectors differ in all coordinates.

Definition 1 $\vec{v}$ properly denominates $\vec{u}$ (denoted $\vec{u} < \vec{v}$) if $\vec{v}$ dominates $\vec{u}$ along an active direction $\vec{\delta}$ and there exists another tuple $\vec{w}$ which is also

dominated by $\vec{v}$ along an active direction $\vec{\delta}_\perp$ compatible with δ.

From the definition of proper domination and from the fact that each leaf label can witness only one active domination direction, the following observations can be made:

Remark 2 The strong domination relation $<$ on lca-tuples has size $O(n^3)$. Furthermore, the relation can be computed in $O(kn^3)$ time.

Remark 3 For any lca-tuple $\vec{v}$, if $\text{mast}(\vec{v}) > 0$ then either $\vec{v}$ is an lca-tuple composed of leaves with the same label or $\vec{v}$ properly dominates some lca-tuple.

It remains to show how the values $\text{mast}(\vec{v})$ are computed. For each lca-tuple $\vec{v}$, the so-called compatibility graph $G(\vec{v})$ is constructed. The nodes of $G(\vec{v})$ are active directions from $\vec{v}$ and there is an edge between two such nodes if and only if corresponding directions are compatible. The vertices of $G(\vec{v})$ are weighted and the weight of a vertex corresponding to an active direction $\vec{\delta}$ equals the maximum mast value of a lca-tuple dominated by $\vec{v}$ along the this direction. Let $\text{MWC}(G(\vec{v}))$ be the maximum weight clique in $G(\vec{v})$.

The bottom-up algorithm for computing nonzero mast values based on the following recursive dependency whose correctness follows immediately from the corresponding definitions and Remark 3:

Lemma 1 *For any lca-tuple* $\vec{v}$

$$\text{mast}(\vec{v}) = \max \begin{cases} 1 & \text{if all elements of } \vec{v} \text{ are leaves} \\ \text{MWC}(G(\vec{v})) & \text{otherwise} \end{cases}. \tag{1}$$

The final step is to demonstrate that once the lca-tuples and the strong domination relation is precomputed, the computation all nonzero mast values can be preformed in $O(n^d)$ time. This is done by generating all possible cliques for all $G(\vec{v})$. Using the fact that the degree of at least one tree is bounded by d, one can show that all the cliques can be generated in $O\left(\sum_{l \le d} \binom{n}{l}\right) = O(d^3(ne/d)^d)$ time and that there is $O(d(ne/d)^d)$ of them [6].

Applications

The k-MAST problem is motivated by the need to compare evolutionary trees. Recent advances in experimental techniques in molecular biology provide diverse data that can be used to construct evolutionary trees. This diversity of data combined with the diversity of methods used to construct evolutionary trees often leads to the situation when the evolution of the same set of species is explained by different evolutionary trees. The maximum agreement subtree problem has emerged as a measure of the agreement between such trees and as a method to identify subtree which is common for these trees. The problem was first defined by Finden and Gordon in the context of two binary trees [7]. These authors also gave a heuristic algorithm to solve the problem. The $O(n^2)$ dynamic programming algorithm for computing MAST values for two binary trees has been given in [11]. Subsequently, a number of improvements leading to fast algorithms for computing MAST value of two trees under various assumptions about rooting and tree degrees [5, 8, 10] and references therein.

The MAST problem for three or more unbounded degree trees is NP-complete [1]. Amir and Keselman report an $O(kn^{d+1} + n^{2d})$ time algorithm for the agreement of k bounded degree trees. The work described here provides a $O(kn^3 + n^d)$ for the case where the number of trees is k and the degree of at least one tree is bounded by d. For $d = 2$ the complexity of the algorithm is dominated by the first term. An $O(kn^3)$ algorithm for this case was also given by Bryant [4] and $O(n^2 \log^{k-1} n)$ implementation of this algorithm was proposed in [9].

k-MAST problem is a fixed parameter tractable in p, the smallest number of leaf labels such that the removal of the corresponding leaves

produces agreement (see [2] and references therein). The approximability of the MAST and related problem has been studied in [3] and references therein.

Cross-References

- Maximum Agreement Subtree (of 2 Binary Trees)
- Maximum Agreement Supertree
- Maximum Compatible Tree

Acknowledgments This work was supported by the Intramural Research Program of the National Institutes of Health, National Library of Medicine.

Recommended Reading

1. Amir A, Keselman D (1997) Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithms. SIAM J Comput 26(6):1656–1669
2. Berry V, Nicolas F (2006) Improved parameterized complexity of the maximum agreement subtree and maximum compatible tree problems. IEEE/ACM Trans Comput Biol Bioinform 3(3):289–302
3. Berry V, Guillemot S, Nicolas F, Paul C (2005) On the approximation of computing evolutionary trees. In: COCOON, Kunming, pp 115–125
4. Bryand D (1997) Building trees, hunting for trees, and comparing trees: theory and methods in phylogenetic analysis. Ph.D. thesis, Department of Mathematics, University of Canterbury
5. Cole R, Farach-Colton M, Hariharan R, Przytycka T, Thorup M (2001) An $o(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. SIAM J Comput 1385–1404
6. Farach M, Przytycka TM, Thorup M (1995) On the agreement of many trees. Inf Process Lett 55(6):297–301
7. Finden CR, Gordon AD (1985) Obtaining common pruned trees. J Classif 2:255–276
8. Kao M-Y, Lam T-W, Sung W-K, Ting H-F (2001) An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. J Algorithms 40(2):212–233
9. Lee C-M, Hung L-J, Chang M-S, Tang C-Y (2004) An improved algorithm for the maximum agreement subtree problem. In: BIBE, Taichung, p 533
10. Przytycka TM (1998) Transforming rooted agreement into unrooted agreement. J Comput Biol 5(2):335–349
11. Steel MA, Warnow T (1993) Kaikoura tree theorems: computing the maximum agreement subtree. Inf Process Lett 48(2):77–82

Maximum Agreement Supertree

Jesper Jansson[1] and Wing-Kin Sung[2]
[1]Laboratory of Mathematical Bioinformatics, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto, Japan
[2]Department of Computer Science, National University of Singapore, Singapore, Singapore

Keywords

Fixed-parameter tractability; Maximum agreement supertree; NP-hardness; Phylogenetic tree; Rooted triplet

Years and Authors of Summarized Original Work

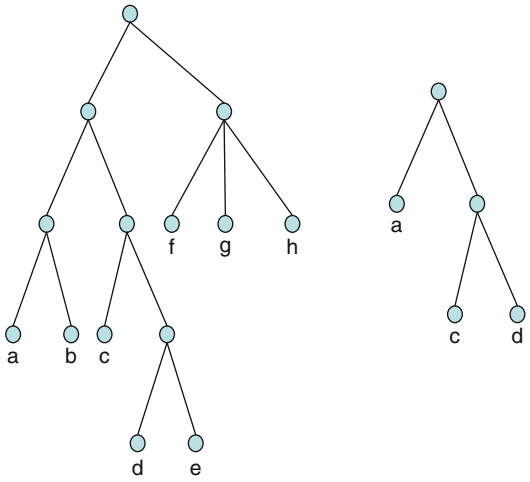
2005; Jansson, Ng, Sadakane, Sung
2007; Berry, Nicolas
2010; Guillemot, Berry
2011; Hoang, Sung

Problem Definition

A *phylogenetic tree* is a rooted, unordered tree whose leaves are distinctly labeled and whose internal nodes have degree at least two. By distinctly labeled, we mean that no two leaves in the tree have the same label. Let T be a phylogenetic tree with a leaf label set S. For any subset S' of S, *the topological restriction of* T *to* S' (denoted by $T \mid S'$) is the tree obtained from T by deleting all nodes which are not on any path from the root to a leaf in S' along with their incident edges and then contracting every edge between a node having just one child and its child. See Fig. 1 for an illustration. For any phylogenetic tree T, denote its set of leaf labels by $\Lambda(T)$.

The *maximum agreement supertree problem (MASP)* [12] is defined as follows.

Problem 1 Let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ be an input set of phylogenetic trees, where the sets $\Lambda(T_i)$ may overlap. The maximum agreement supertree problem (MASP) asks for a phylogenetic



Maximum Agreement Supertree, Fig. 1 Let T be the phylogenetic tree on the *left*. Then $T \mid \{a, c, d\}$ is the phylogenetic tree shown on the *right*

tree Q with leaf label set $\Lambda(Q) \subseteq \bigcup_{T_i \in \mathcal{T}} \Lambda(T_i)$ such that $|\Lambda(Q)|$ is maximized and for each $T_i \in \mathcal{T}$, it holds that $T_i \mid \Lambda(Q)$ is isomorphic to $Q \mid \Lambda(T_i)$.

The following notation is used below: $n = \left|\bigcup_{T_i \in \mathcal{T}} \Lambda(T_i)\right|$, $k = |\mathcal{T}|$, and $D = \max_{T_i \in \mathcal{T}}\{\deg(T_i)\}$, where $\deg(T_i)$ is the degree of T_i (i.e., the maximum number of children of any node belonging to T_i).

A problem related to MASP is the *maximum compatible supertree problem (MCSP)* [2]:

Problem 2 Let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ be an input set of phylogenetic trees, where the sets $\Lambda(T_i)$ may overlap. The maximum compatible supertree problem (MCSP) asks for a phylogenetic tree W with leaf label set $\Lambda(W) \subseteq \bigcup_{T_i \in \mathcal{T}} \Lambda(T_i)$ such that $|\Lambda(W)|$ is maximized and for each $T_i \in \mathcal{T}$, it holds that $T_i \mid \Lambda(W)$ can be obtained from $W \mid \Lambda(T_i)$ by applying a series of edge contractions.

For information about MCSP, refer to [2, 11].

Key Results

The special case of the maximum agreement supertree problem in which $\Lambda(T_1) = \Lambda(T_2) \ldots = \Lambda(T_k)$ has been well studied in the literature and is also known as the *maximum agreement subtree problem (MAST)*. By utilizing known results for MAST, several results can be obtained for various special cases of MASP. Firstly, it is known that MAST can be solved in $O(\sqrt{D}n \log(2n/D))$ time when $k = 2$ (see [13]) or in $O(kn^3 + n^D)$ time when $k \geq 3$ (see [4, 6]), which leads to the following theorems.

Theorem 1 ([12]) *When $k = 2$, MASP can be solved in $O(T_{MAST} + n)$ time, where T_{MAST} is the time required to solve MAST for two $O(n)$-leaf trees. Note that $T_{MAST} = O\left(\sqrt{D}\, n \log(2n/D)\right)$.*

Theorem 2 ([2]) *For any fixed $k \geq 3$, if every leaf appears in either 1 or k trees, MASP can be solved in $O(T'_{MAST} + kn)$ time, where T'_{MAST} is the time required to solve MAST for $\{T_1|L, T_2|L, \ldots, T_k|L\}$, where $L = \bigcap_{T_i \in \mathcal{T}} \Lambda(T_i)$. Note that $T'_{MAST} = O(k|L|^3 + |L|^D)$.*

On the negative side, the maximum agreement supertree problem is NP-hard in general, as shown by the next theorem. (A *rooted triplet* is a binary phylogenetic tree with exactly three leaves.)

Theorem 3 ([2, 12]) *For any fixed $k \geq 3$, MASP with unbounded D is NP-hard. Furthermore, MASP with unbounded k remains NP-hard even if restricted to rooted triplets, i.e., $D = 2$.*

The inapproximability results for MAST by Hein et al. [9] and Gąsieniec et al. [7] immediately carry over to MASP with unbounded D as follows.

Theorem 4 ([2, 12]) *cannot be approximated within a factor of $2^{\log^{\delta} n}$ in polynomial time for any constant $\delta < 1$, unless NP $\subseteq$ DTIME[$2^{\text{polylog}\, n}$], even when restricted to $k = 3$. Also, MASP cannot be approximated within a factor of n^{ε} for any constant ε where $0 \leq \varepsilon < \frac{1}{9}$ in polynomial time unless $P = NP$, even for instances containing only trees of height 2.*

Although MASP is difficult to approximate in polynomial time, a simple approximation

algorithm based on a technique from [1] achieves an approximation factor that is close to the bounds given in Theorem 4.

Theorem 5 ([12]) *MASP can be approximated within a factor of* $\frac{n}{\log n}$ *in* $O(n^2) \cdot \min\{O(k \cdot (\log\log n)^2), O(k + \log n \cdot \log\log n)\}$ *time. MASP restricted to rooted triplets can be approximated within a factor of* $\frac{n}{\log n}$ *in* $O(k + n^2 \log^2 n)$ *time.*

Fixed-parameter tractable algorithms for solving MASP also exist. In particular, for *binary* phylogenetic trees, Jansson et al. [12] first gave an $O(k(2n^2)^{3k^2})$-time algorithm. Later, Guillemot and Berry [8] improved the time complexity to $O((8n)^k)$. Hoang and Sung [11] further improved the time complexity to $O((6n)^k)$, as summarized in Theorem 6.

Theorem 6 ([11]) *MASP restricted to* $D = 2$ *can be solved in* $O((6n)^k)$ *time.*

For the case where each tree in $\mathcal{T}$ has degree at most D, Hoang and Sung [11] gave the following fixed-parameter polynomial-time solution.

Theorem 7 ([11]) *MASP restricted to phylogenetic trees of degree at most* D *can be solved in* $O((kD)^{kD+3}(2n)^k)$ *time.*

For unbounded n, k, and D, Guillemot and Berry [8] proposed a solution that is efficient when the input trees are similar.

Theorem 8 ([8]) *MASP can be solved in* $O((2k)^p k n^2)$ *time, where* p *is an upper bound on the number of leaves that are missing from* $\bigcup_{T_i \in \mathcal{T}} \Lambda(T_i)$ *in a MASP solution.*

Applications

One challenge in phylogenetics is to develop good methods for merging a collection of phylogenetic trees on overlapping sets of taxa into a single supertree so that no (or as little as possible) branching information is lost. Ideally, the resulting supertree can then be used to deduce evolutionary relationships between taxa which do not occur together in any one of the input trees. Supertree methods are useful because most individual studies investigate relatively few taxa [15] and because sample bias leads to certain taxa being studied much more frequently than others [3]. Also, supertree methods can combine trees constructed for different types of data or under different models of evolution. Furthermore, although computationally expensive methods for constructing reliable phylogenetic trees are infeasible for large sets of taxa, they can be applied to obtain highly accurate trees for smaller, overlapping subsets of the taxa which may then be merged using computationally less intense, supertree-based techniques (see, e.g., [5, 10, 14]).

Since the set of trees which is to be combined may in practice contain contradictory branching structure (e.g., if the trees have been constructed from data originating from different genes or if the experimental data contains errors), a supertree method needs to specify how to resolve conflicts. One intuitive idea is to identify and remove a smallest possible subset of the taxa so that the remaining taxa can be combined without conflicts. In this way, one would get an indication of which ancestral relationships can be regarded as resolved and which taxa need to be subjected to further experiments. The above biological problem can be formalized as MASP.

Open Problems

An open problem is to improve the time complexity of the currently fastest algorithms for solving MASP. Moreover, the existing fixed-parameter polynomial-time algorithms for MASP are not practical, so it could be useful to provide heuristics that work well on real data.

Cross-References

- Maximum Agreement Subtree (of 2 Binary Trees)
- Maximum Agreement Subtree (of 3 or More Trees)
- Maximum Compatible Tree

Acknowledgments JJ was funded by the Hakubi Project at Kyoto University and KAKENHI grant number 26330014.

Recommended Reading

1. Akutsu T, Halldórsson MM (2000) On the approximation of largest common subtrees and largest common point sets. Theor Comput Sci 233(1–2): 33–50
2. Berry V, Nicolas F (2007) Maximum agreement and compatible supertrees. J Discret Algorithms 5(3):564–591
3. Bininda-Emonds ORP, Gittleman JL, Steel MA (2002) The (super)tree of life: procedures, problems, and prospects. Annu Rev Ecol Syst 33: 265–289
4. Bryant D (1997) Building trees, hunting for trees, and comparing trees: theory and methods in phylogenetic analysis. PhD thesis, University of Canterbury, Christchurch
5. Chor B, Hendy M, Penny D (2007) Analytic solutions for three taxon ML trees with variable rates across sites. Discret Appl Math 155(6–7): 750–758
6. Farach M, Przytycka T, Thorup M (1995) On the agreement of many trees. Inf Process Lett 55(6):297–301
7. Gąsieniec L, Jansson J, Lingas A, Östlin A (1999) On the complexity of constructing evolutionary trees. J Comb Optim 3(2–3):183–197
8. Guillemot S, Berry V (2010) Fixed-parameter tractability of the maximum agreement supertree problem. IEEE/ACM Trans Comput Biol Bioinform 7(2):342–353
9. Hein J, Jiang T, Wang L, Zhang K (1996) On the complexity of comparing evolutionary trees. Discret Appl Math 71(1–3):153–169
10. Henzinger MR, King V, Warnow T (1999) Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. Algorithmica 24(1):1–13
11. Hoang VT, Sung W-K (2011) Improved algorithms for maximum agreement and compatible supertrees. Algorithmica 59(2):195–214
12. Jansson J, Ng JHK, Sadakane K, Sung W-K (2005) Rooted maximum agreement supertrees. Algorithmica 43(4):293–307
13. Kao M-Y, Lam T-W, Sung W-K, Ting H-F (2001) An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. J Algorithms 40(2):212–233
14. Kearney P (2002) Phylogenetics and the quartet method. In: Current topics in computational molecular biology. MIT, Cambridge pp 111–133
15. Sanderson MJ, Purvis A, Henze C (1998) Phylogenetic supertrees: assembling the trees of life. TRENDS Ecol Evol 13(3):105–109

Maximum Cardinality Stable Matchings

Eric McDermid
Cedar Park, TX, USA

Keywords

Approximation algorithm; Lower bounds; Matching; NP-hard; Preferences; Stability; Ties; UGC-hard; Upper bounds

Years and Authors of Summarized Original Work

2002; Manlove, Irving, Iwama, Miyazaki, Morita
2003; Halld'orsson, Iwama, Miyazaki, Yanagisawa
2004; Halld'orsson, Iwama, Miyazaki, Yanagisawa
2004; Iwama, Miyazaki, Okamoto
2007; Halld'orsson, Iwama, Miyazaki, Yanagisawa
2007; Iwama, Miyazaki, Yamauchi
2007; Yanagisawa
2008; Irving, Manlove
2008; Iwama, Miyazaki, Yamauchi
2009; McDermid
2011; Kir'aly
2013; Kir'aly
2014; Huang and Kavitha
2014; Iwama, Miyazaki, and Yanagisawa
2014; Paluch
2014; Radnai
2015; Dean, Jalasutram

Problem Definition

The input to an instance of the classical *stable marriage problem* consists of a set of n men and n women. Additionally, each person provides a strictly ordered preference list of the opposite set. The goal is to find a complete matching of men to women that is also *stable*, i.e., a matching having the property that there does not exist a

man and a woman who prefer each other over their matched assignment. In their seminal work, Gale and Shapley [2] showed that every instance of the stable marriage problem admits at least one stable matching and showed that one can be found in polynomial time (see entry ▸ Stable Marriage). In fact, stable marriage instances can have exponentially many stable matchings [18].

More general settings arise when relaxations of Gale and Shapley's original version are permitted. In the *stable marriage problem with incomplete lists* (SMI), men and women may deem arbitrary members of the opposite set *unacceptable*, prohibiting the pair from being matched together. In the *stable marriage problem with ties* (SMT), preference lists need not be strictly ordered but may instead contain subsets of agents all having the same rank. Instances of SMT and SMI always admit a stable matching, and, crucially, all stable matchings for a fixed instance have the same cardinality. Interestingly, when both ties and incomplete lists are allowed (denoted SMTI, see entry ▸ Stable Marriage with Ties and Incomplete Lists), stable matchings again exist but can differ in cardinality. How can we find one of maximum cardinality?

Key Results

Benchmark Results

Manlove et al. [20] established two key benchmarks for the problem of computing a maximum cardinality stable matching (MAX-SMTI). First, they showed that the problem is NP-hard under the following two simultaneous restrictions. One set of agents, say, the men, all have strictly ordered preference lists, while each woman's preference list is either strictly ordered or is a tie of length two. Second, they showed that MAX-SMTI is approximable within a factor of 2 by arbitrarily breaking the ties and finding any stable matching in the resulting SMI instance.

Since then, researchers have focused on improving the approximability bounds for MAX-SMTI. The severity of the restrictions in Manlove et al.'s hardness results has led researchers to study not only the general version but a number of special cases of MAX-SMTI as well.

Upper Bounds: The General Case

For the general case of MAX-SMTI, Iwama et al. [12] gave a $2-c\left(\frac{\log n}{n}\right)$ approximation algorithm, where c is a positive constant. This algorithm was subsequently improved to yield a performance guarantee of $2-\frac{c'}{\sqrt{n}}$, where c' is a positive constant which is at most $1/4\sqrt{6}$ [14]. The first approximation algorithm to achieve a constant performance guarantee better than two was given by Iwama et al. [13], establishing a performance ratio of $15/8$. Next, Király [16] devised a new approximation algorithm with a bound of 5/3. Finally, McDermid [21] obtained 3/2, which is currently the best known approximation ratio. Later, Paluch [22] and Király [17] also obtained approximation algorithms with the same performance guarantee of 3/2; however, their algorithms have the advantage of running in linear time. Király's has the extra benefit of requiring only "local" preference list information. The following theorem summarizes the best known upper bound for the general case.

Theorem 1 *There is a 3/2-approximation algorithm for MAX-SMTI.*

Upper Bounds: Special Cases

The special case of MAX-SMTI that has received the most attention is that in which ties may only appear in one set only. We let 1S-MAX-SMTI denote this problem. Halldórsson et al. [3] gave a $(2/(1+T^{-2}))$-approximation algorithm for 1S-MAX-SMTI, where T is the length of the longest tie. This bound was improved to 13/7 for MAX-SMTI instances in which ties are restricted to be of size at most 2 [3]. They later showed that 10/7 is achievable for 1S-MAX-SMTI [4] via a randomized approximation algorithm. Irving and Manlove [11] described a 5/3-approximation algorithm for 1S-MAX-SMTI instances in which lists may have at most one tie that may only appear at the end of the preference list. One of the most important results in this area was that of Király [16], who provided a particularly simple and elegant 3/2-approximation algorithm with an equally transparent analysis for 1S-MAX-SMTI (with no further restrictions

on the problem). Since then, further improvements have been obtained for 1S-MAX-SMTI by Iwama, Miyazaki, and Yanagisawa [15], who exploited a linear programming relaxation to obtain a 25/17-approximation. Huang and Kavitha [10] used different techniques to improve upon this, giving a linear-time algorithm with a ratio of 22/15. Radnai [23] tightened their analysis to show that 41/28 is in fact achieved. Finally, Dean and Jalasutram [1] showed that the algorithm given in [15] actually achieves 19/13 through an analysis using a factor-revealing LP. The following theorem summarizes the best known upper bounds for the special cases of MAX-SMTI.

Theorem 2 *There is a 19/13-approximation algorithm for 1S-MAX-SMTI. When all ties have length at most two, there is a (randomized) 10/7-approximation algorithm for MAX-SMTI.*

Lower Bounds

The best lower bounds on approximability are due to Yanagisawa [24] and Iwama et al. [5]. Yanagisawa [24] showed that MAX-SMTI is NP-hard to approximate within 33/29 and UGC-hard to approximate within 4/3. 1S-MAX-SMTI was shown by Iwama et al. [5] to be NP-hard to approximate within 21/19 and UGC-hard to approximate within 5/4. The next theorem summarizes these results.

Theorem 3 *It is NP-hard to approximate MAX-SMTI (1S-MAX-SMTI) within* 33/29 (21/19). *It is UGC-hard to approximate MAX-SMTI (1S-MAX-SMTI) within* 4/3 (5/4).

Applications

Stable marriage research is a fascinating subset of theoretical computer science not only for its intrinsic interest but also for its widespread application to real-world problems. Throughout the world, centralized matching schemes are used in various contexts such as the assignment of students to schools and graduating medical students to hospitals. We direct the reader to [19, *Section 1.3.7*] for a comprehensive overview (see also entry ▸ Hospitals/Residents Problem). Perhaps the most famous of these is the National Resident Matching Program (NRMP) [7] in the United States, which allocates over 35,000 graduating medical students to their first job at a hospital. Similar schemes exist in Canada [8], Scotland [9], and Japan [6]. In one way or another, all of these matching schemes require one or both of the sets involved to produce preference lists ranking the other set. Methods similar to the Gale-Shapley algorithm are then used to create the assignments.

Both economists and computer scientists alike have influenced the design and implementation of such matching schemes. In fact, the 2012 Nobel Prize for Economic Sciences was awarded to Alvin Roth and Lloyd Shapley, in part for their contribution to the widespread deployment of matching algorithms in practical settings. Researchers Irving and Manlove at the School of Computing Science at the University of Glasgow led the design and implementation of algorithms for the *Scottish Foundation Allocation Scheme* that have been used by NHS Education for Scotland to assign graduating medical students to hospital programs [19, *Section 1.3.7*]. This setting has actually yielded true instances of MAX-SMTI, as hospital programs have been allowed to have ties in their preference lists.

Recommended Reading

1. Dean B, Jalasutram R (2015, to appear) Factor revealing LPs and stable matching with ties and incomplete lists. In: Proceedings of MATCH-UP 2015: the 3rd international workshop on matching under preferences, Glasgow
2. Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69:9–15
3. Halldórsson MM, Iwama K, Miyazaki S, Yanagisawa H (2003) Improved approximation of the stable marriage problem. In: Proceedings of ESA 2003: the 11th annual European symposium on algorithms, Budapest. Lecture notes in computer science, vol 2832. Springer, pp 266–277
4. Halldórsson MM, Iwama K, Miyazaki S, Yanagisawa H (2004) Randomized approximation of the stable marriage problem. Theor Comput Sci 325(3):439–465
5. Halldórsson M, Iwama K, Miyazaki S, Yanagisawa H (2007) Improved approximation of the stable marriage problem. ACM Trans Algorithms 3(3):30-es

6. http://www.jrmp.jp (Japan Resident Matching Program website)
7. http://www.nrmp.org (National Resident Matching Program website)
8. http://www.carms.ca (Canadian Resident Matching Service website)
9. http://www.nes.scot.nhs.uk/sfas (Scottish Foundation Allocation Scheme website)
10. Huang C-C, Kavitha T (2014) An improved approximation algorithm for the stable marriage problem with one-sided ties. In: Proceedings of IPCO 2014, the 17th conference on integer programming and combinatorial optimization, Bonn. Lecture notes in computer science, vol 8494. Springer, pp 297–308
11. Irving RW, Manlove D (2008) Approximation algorithms for hard variants of the stable marriage and hospitals/residents problem. J Comb Optim 16(3):279–292
12. Iwama K, Miyazaki S, Okamoto K (2004) A $\left(2 - c\frac{\log n}{n}\right)$-approximation algorithm for the stable marriage problem. In: Proceedings of SWAT 2004: the 9th Scandinavian workshop on algorithm theory, Humlebaek. Lecture notes in computer science, vol 3111. Springer, pp 349–361
13. Iwama K, Miyazaki S, Yamauchi N (2007) A 1.875-approximation algorithm for the stable marriage problem. In: Proceedings of SODA 2007: the eighteenth ACM/SIAM symposium on discrete algorithms, New Orleans, pp 288–297
14. Iwama K, Miyazaki S, Yamauchi N (2008) A $\left(2 - c\frac{1}{\sqrt{n}}\right)$-approximation algorithm for the stable marriage problem. Algorithmica 51(3):342–356
15. Iwama K, Miyazaki S, Yanagisawa H (2014) A 25/17-approximation algorithm for the stable marriage problem with one-sided ties. Algorithmica 68:758–775
16. Király Z (2011) Better and simpler approximation algorithms for the stable marriage problem. Algorithmica 60(1):3–20
17. Király Z (2013) Linear time local approximation algorithm for maximum stable marriage. MDPI Algorithms 6(3):471–484
18. Knuth DE (1976) Mariages stables. Les Presses de L'Université de Montréal, Montréal
19. Manlove D (2013) Algorithmics of matching under preferences. World Scientific, Hackensack
20. Manlove DF, Irving RW, Iwama K, Miyazaki S, Morita Y (2002) Hard variants of stable marriage. Theor Comput Sci 276(1–2):261–279
21. McDermid EJ (2009) A 3/2-approximation algorithm for general stable marriage. In: Proceedings of ICALP 2009: the 36th international colloquium on automata, languages and programming, Rhodes. Lecture notes in computer science, vol 5555. Springer, pp 689–700
22. Paluch KE (2014) Faster and simpler approximation of stable matchings. MDPI Algorithms 7(2): 189–202
23. Radnai A (2014) Approximation algorithms for the stable matching problem. Master's thesis, Eötvös Loránd University
24. Yanagisawa H (2007) Approximation algorithms for stable marriage problems. PhD thesis, School of Informatics, Kyoto University

Maximum Compatible Tree

Vincent Berry
Institut de Biologie Computationnelle, Montpellier, France

Keywords

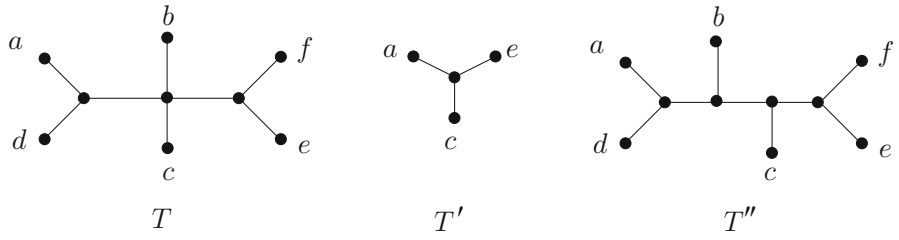
Consensus of trees; Maximum compatible tree; Pattern matching on trees; Phylogenetics; Trees

Years and Authors of Summarized Original Work

2001; Ganapathy, Warnow

Problem Definition

This problem is a pattern matching problem on leaf-labeled trees. Each input tree is considered as a branching pattern inducing specific groups of leaves. Given a set of input trees with identical leaf sets, the goal is to find the largest subset of leaves on the branching pattern of which the input trees do not disagree. A *maximum compatible tree* is a tree on such a leaf set and with a branching pattern respecting that of each input tree (see below for a formal definition). The maximum compatible tree problem (MCT) is to find such a tree or, equivalently, its leaf set. The main motivation for this problem is in phylogenetics, to measure the similarity between evolutionary trees or to represent a consensus of a set of trees. The problem was introduced in [10] and [11, under the MRST acronym]. Previous related works concern the well-known maximum agreement subtree problem (MAST).



Maximum Compatible Tree, Fig. 1 Three unrooted trees. A tree T, a tree T' such that $T' = T|\{a, c, e\}$, and a tree T'' such that $T'' \trianglerighteq T$

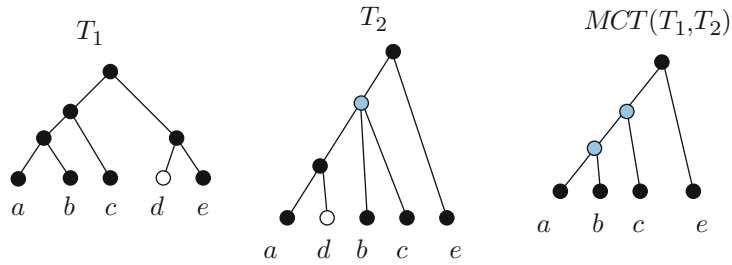
Solving MAST is finding the largest subset of leaves on which all input trees *exactly* agree. The difference between MAST and MCT is that MAST seeks a tree whose branching information is isomorphic to that of a subtree in each of the input trees, while MCT seeks a tree that contains the branching information (i.e., groups) of the corresponding subtree of each input tree. This difference allows the tree obtained for MCT to be more informative, as it can include branching information present in one input tree but not in the others, as long as this information is *compatible* (in the sense of [14]) with the latters. Both problems are equivalent when all input trees are binary. Ganapathy and Warnow [6] were the first to give an algorithm to solve MCT in its general form. Their algorithm relies on a simple dynamic programming approach similar to a work on MAST [13] and has a running time exponential in the number of input trees and in the maximum degree of a node in the input trees. Later, [1] proposed a fixed-parameter algorithm using one parameter only. Approximation results have also been obtained, [3,7] proposing low-cost polynomial-time algorithms that approximate the complement of MCT within a constant factor.

Notations Trees considered here are evolutionary trees (*phylogenies*). Such a tree T has its leaf set $L(T)$ in bijection with a label set and is either rooted, in which case all internal nodes have at least two children each, or unrooted, in which case internal nodes have a degree of at least three. Given a set L of labels and a tree T, the *restriction* of T to L, denoted $T|L$, is the tree obtained in taking the smallest induced subgraph of T that connects leaves with labels in $L \cap L(T)$ and then removing any degree-two (non-root) node to make the tree homeomorphically irreducible. Two trees T, T' are *isomorphic*, denoted $T = T'$, if and only if there is a graph isomorphism $T \mapsto T'$ preserving leaf labels (and the roots if both trees are rooted). A tree T *refines* a tree T', denoted $T \trianglerighteq T'$, whenever T can be transformed into T' by collapsing some of its internal edges (*collapsing* an edge means removing it and merging its extremities). See Fig. 1 for examples of these relations between trees. Note that a tree T properly refining another tree T' agrees with the entire evolutionary history of T' while containing additional information absent from T': at least one high-degree node of T' is replaced in T by several nodes of lesser degree; hence, T contains more information than T' on which species belong together.

Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of input trees with identical leaf sets L, a tree T with leaves in L is said to be *compatible with* $\mathcal{T}$ if and only if $\forall T_i \in \mathcal{T}$, $T \trianglerighteq T_i|L(T)$. If there is a tree T compatible with $\mathcal{T}$ such that $L(T) = L$, then the collection $\mathcal{T}$ is said to be *compatible*. Knowing whether a collection is compatible is a problem for which linear-time algorithms have been known for a long time (e.g., [9]). The MAXIMUM COMPATIBLE TREE problem is a natural optimization version of this problem to deal with incompatible collections of trees.

Problem 1 (MAXIMUM COMPATIBLE TREE – MCT)

INPUT: A collection $\mathcal{T}$ of trees with the same leaf sets.



Maximum Compatible Tree, Fig. 2 An incompatible collection of two input trees $\{T_1, T_2\}$ and their maximum compatible tree, $T = MCT(T_1, T_2)$. Removing the leaf d renders the input trees compatible, hence $L(T) = \{a, b, c, e\}$. Here, T strictly refines T_2 restricted to $L(T)$, which is expressed by the fact that a node in T_2 (the *blue* one) has its child subtrees distributed between several connected nodes of T (*blue nodes*). Note also that here $|MCT(T_1, T_2)| > |MAST(T_1, T_2)|$

OUTPUT: A tree compatible with $\mathcal{T}$ having the largest number of leaves. Such a tree is denoted $MCT(\mathcal{T})$.

See Fig. 2 for an example. Note that $\forall \mathcal{T}$, $|MCT(\mathcal{T})| \geq |MAST(\mathcal{T})|$ and that MCT is equivalent to MAST when the input trees are binary. Note also that instances of MCT and MAST can have several optimum solutions.

Key Results

Exact Algorithms

The MCT problem was shown to be NP-hard on 6 trees by [10] and then on 2 trees by [11]. The NP-hardness holds as long as one of the input trees is not of bounded degree. For two bounded-degree trees, Hein et al. [11] mention a polynomial-time algorithm based on *aligning* trees. Ganapathy and Warnow propose an exponential algorithm for solving MCT in the general case [6]. Given two trees T_1, T_2, they show how to compute a binary MCT of any pair of subtrees $(S_1 \in T_1, S_2 \in T_2)$ by dynamic programming. Subtrees whose root is of high degree are handled by considering every possible partition of the roots's children in two sets. This leads the complexity bound to have a term exponential in d, the maximum degree of a node in the input trees. When dealing with k input trees, k-tuples of subtrees are considered, and the simultaneous bipartitions of the roots's children for k subtrees are considered. Hence, the complexity bound is also exponential in k.

Theorem 1 ([6]) *Let L be a set of n leaves. The* MCT *problem for a collection of k rooted trees on L in which each tree has degree at most $d + 1$ can be solved in $O(2^{2kd} n^k)$ time.*

The result easily extends to unrooted trees by considering each of the n leaves in turn as a possible root for all trees of the collection.

Theorem 2 ([6]) *Given a collection of k unrooted trees with degree at most $d + 1$ on an n-leaf set, the* MCT *problem can be solved in $O(2^{2kd} n^{k+1})$.*

Let $\mathcal{T}$ be a collection on a set L of n leaves, [1] considered the following decision problem denoted MCT_p: given $\mathcal{T}$ and $p \in [0, n]$, does $|MCT(\mathcal{T})| \geq n - p$?

Theorem 3 ([1])

1. MCT_p *on rooted trees can be solved in $O(\min\{3^p kn, 2.27^p + kn^3\})$ time.*
2. MCT_p *on unrooted trees can be solved in $O\big((p+1) \times \min\{3^p kn, 2.27^p + kn^3\}\big)$ time.*

The $3^p kn$ term comes from an algorithm that first identifies in $O(kn)$ time a 3-leaf set S

on which the input trees conflict and then recursively obtains a maximum compatible tree T_1, resp. T_2, T_3 for each of the three collections $\mathcal{T}_1$, resp. $\mathcal{T}_2, \mathcal{T}_3$ obtained by removing from the input trees a leaf in S and lastly returning the $\mathcal{T}_i$ such that $|\text{MCT}(\mathcal{T}_i)|$ is maximum (with $i \in [1,3]$). The $2.27^p + kn^3$ term comes from an algorithm using a reduction of MCT to 3-HITTING SET. Negative results have been obtained by Guillemot and Nicolas concerning the fixed-parameter tractability of MCT with regard to the maximum degree D of the input trees:

Theorem 4 ([8])

1. MCT *is* $W[1]$*-hard with respect to* D.
2. MCT *cannot be solved in* $O(N^{o(2^{D/2})})$ *time unless SNP* $\subseteq$ *SE, where* N *denotes the input length, i.e.,* $N = O(kn)$.

The MCT problem also admits a variant that deals with *supertrees*, i.e., trees having different (but overlapping) sets of leaves. The resulting problem is $W[2]$-hard with respect to p [2].

Approximation Algorithms

The idea of locating and then eliminating successively all the conflicts between the input trees has also led to approximation algorithms for the *complement* version of the MCT problem, denoted CMCT. Let L be the leaf set of each tree in an input collection $\mathcal{T}$; CMCT aims at selecting the smallest number of leaves $S \subseteq L$ such that the collection $\{T_i|(L-S) : T_i \in \mathcal{T}\}$ is compatible.

Theorem 5 ([7]) *Given a collection* $\mathcal{T}$ *of* k *rooted trees on an* n*-leaf set* L*, there is a 3-approximation algorithm for* CMCT *that runs in* $O(k^2n^2)$ *time.*

The running time of this algorithm was later improved:

Theorem 6 ([3, 5]) *There is an* $O(kn + n^2)$ *time 3-approximation algorithm for* CMCT *on a collection of* k *rooted trees with* n *leaves.*

Note also that working on rooted or unrooted trees does not change the achievable approximation ratio for CMCT [3].

Applications

In bioinformatics, the MCT problem (and similarly MAST) is used to reach different practical goals. The first motivation is to measure the similarity of a set of trees. These trees can represent RNA secondary structures [11, 12] or estimates of a phylogeny inferred from different datasets composed of molecular sequences (e.g., genes) [14]. The gap between the size of a maximum compatible tree and the number of input leaves indicates the degree of disimilarity of the input trees. Concerning the phylogenetic applications, quite often some edges of the trees inferred from the datasets have been collapsed due to insufficient statistical support, resulting in some higher-degree nodes in the trees considered by MCT. Each such node does not indicate a multi-speciation event but rather the uncertainty with respect to the branching pattern to be chosen for its child subtrees. In such a situation, the MCT problem is to be preferred to MAST, as it correctly handles high-degree nodes, enabling them to be resolved according to branching information present in other input trees. As a result, more leaves are conserved in the output tree; hence, a larger degree of similarity is detected between the input trees. Note also that a low similarity value between the input trees can be due to horizontal gene transfers. When these events are not too numerous, identifying species subject to such effects is done by first suspecting leaves discarded from a maximum compatible tree.

The shape of a maximum compatible tree, i.e., not just its size, also has an application in systematic biology to obtain a consensus of a set of phylogenies that are optimal for some tree-building criterion. For instance, the maximum parsimony and maximum likelihood criteria can provide several dozens (sometimes hundreds) of optimal or near-optimal trees. In practice, these

trees are first grouped into islands of neighboring trees, and a consensus tree is obtained for each island by resorting to a classical consensus tree method, e.g., the majority-rule or strict consensus. The trees representing the islands form a collection of which a consensus is then sought. However, consensus methods keeping all input leaves tend to create poorly resolved trees. An alternative approach lies in proposing a representative tree that contains a largest possible subset of leaves on the position of which the trees of the collection agree. Again, MCT is more suited than MAST as the input trees can contain some high-degree nodes, with the same meaning as discussed above.

Open Problems

A direction for future work would be to examine the variant of MCT where some leaves are imposed in the output tree. This question arises when a biologist wants to ensure that the species central to his study are contained in the output tree. For MAST on two trees, this constrained variant of the problem was shown in a natural way to be of the same complexity as the regular version [4]. For MCT however, such a constraint can lead to several optimization problems that need to be sorted out. Another important work to be done is a set of experiments to measure the range of parameters for which the algorithms proposed to solve or approximate MCT are useful.

URLs to Code and Datasets

A Perl program can be asked to the author of this entry.

Cross-References

- Maximum Agreement Subtree (of 2 Binary Trees)
- Maximum Agreement Subtree (of 3 or More Trees)

Recommended Reading

1. Berry V, Nicolas F (2006) Improved parametrized complexity of the maximum agreement subtree and maximum compatible tree problems. IEEE/ACM Trans Comput Biol Bioinformatics 3(3):289–302
2. Berry V, Nicolas F (2007) Maximum agreement and compatible supertrees. J Discret Algorithms 5(3):564–591
3. Berry V, Guillemot S, Nicolas F, Paul C (2005) On the approximation of computing evolutionary trees. In: Wang L (ed) Proceedings of the 11th annual international conference on computing and combinatorics (COCOON'05), Shanghai. LNCS, vol 3595. Springer, pp 115–125
4. Berry V, Peng ZS, Ting HF (2008) From constrained to unconstrained maximum agreement subtree in linear time. Algorithmica 50(3):369–385
5. Berry V, Guillemot S, Nicolas F, Paul C (2009) Linear time 3-approximation for the mast problem. ACM Trans. Algorithms 5(2):23:1–23:18
6. Ganapathy G, Warnow TJ (2001) Finding a maximum compatible tree for a bounded number of trees with bounded degree is solvable in polynomial time. In: Gascuel O, Moret BME (eds) Proceedings of the 1st international workshop on algorithms in bioinformatics (WABI'01), Aarhus, pp 156–163
7. Ganapathy G, Warnow TJ (2002) Approximating the complement of the maximum compatible subset of leaves of k trees. In: Proceedings of the 5th international workshop on approximation algorithms for combinatorial optimization (APPROX'02), Rome, pp 122–134
8. Guillemot S, Nicolas F (2006) Solving the maximum agreement subtree and the maximum compatible tree problems on many bounded degree trees. In: Lewenshtein M, Valiente G (eds) Proceedings of the 17th combinatorial pattern matching symposium (CPM'06), Barcelona. LNCS, vol 4009. Springer, pp 165–176
9. Gusfield D (1991) Efficient algorithms for inferring evolutionary trees. Networks 21:19–28
10. Hamel AM, Steel MA (1996) Finding a maximum compatible tree is NP-hard for sequences and trees. Appl Math Lett 9(2):55–59
11. Hein J, Jiang T, Wang L, Zhang K (1996) On the complexity of comparing evolutionary trees. Discr Appl Math 71(1–3):153–169
12. Jiang T, Wang L, Zhang K (1995) Alignment of trees – an alternative to tree edit. Theor Comput Sci 143(1):137–148
13. Steel MA, Warnow TJ (1993) Kaikoura tree theorems: computing the maximum agreement subtree. Inf Process Lett 48(2):77–82
14. Swofford D, Olsen G, Wadell P, Hillis D (1996) Phylogenetic inference. In: Hillis D, Moritz D, Mable B (eds) Molecular systematics, 2nd edn. Sinauer Associates, Sunderland, pp 407–514

Maximum Lifetime Coverage

Weili Wu[1,2,3] and Ling Ding[4]
[1]College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi Province, China
[2]Department of Computer Science, California State University, Los Angeles, CA, USA
[3]Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA
[4]Institute of Technology, University of Washington Tacoma, Tacoma, WA, USA

Keywords

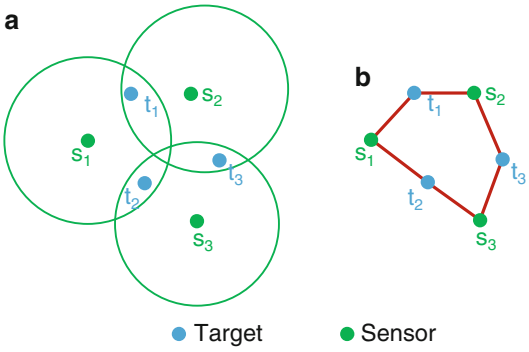
Algorithm; Approximation; Constant; Coverage; Lifetime; Wireless sensor network

Years and Authors of Summarized Original Work

2005; Cardei, Mihaela; Thai, My T.; Li, Yingshu; Wu, Weili
2012; Ding, Ling; Wu, Weili; Willson, James; Wu, Lidong; Lu, Zaixin; Lee, Wonjun

Problem Definition

Energy resources are very limited in wireless sensor networks since the wireless devices are small and battery powered. There are two ways to deploy wireless sensor networks. One is deterministic [1] deployment, and the other one is stochastic or random deployment [2]. In deterministic deployment, the goal is to minimize the number of sensors. In the latter one, the goal is to improve coverage ratio. Normally, in random or stochastic deployment, one target is covered by several sensors. It is unnecessary and a waste of energy to activate all sensors around the target to monitor it. We can prolong the coverage duration through making sleep/activate schedules in wireless sensor networks when we don't have abundant energy resources. In Fig. 1, t_1, t_2, and t_3 are three targets. s_1, s_2, and s_3 are three sensors.



Maximum Lifetime Coverage, Fig. 1 Network model and comparison of disjoint and non-disjoint coverage [3]

s_1 can cover t_1 and t_2. s_2 can cover t_1 and t_3. s_3 can cover t_2 and t_3. Assume each sensor can be active for 1 h. s_1 and s_2 can collaborate to cover all targets, and the coverage duration will be 1 h. After 1 h, there are no enough sensors to cover all targets. The coverage lifetime in this case is 1 h, and s_3 is sleep within this 1 h. There can be another coverage choice. s_1 and s_2 collaborate for 0.5 h to cover all targets while s_3 sleeps. s_2 and s_3 collaborate for 0.5 h while s_1 sleeps. s_1 and s_3 collaborate for 0.5 h while s_2 sleeps. The total coverage lifetime will become 1.5 h. The problem is how to divide sensors into groups and how long each group should work to prolong the coverage lifetime. One sensor can appear in several groups. But the total active time of one sensor should satisfy its battery capacity.

We model a wireless sensor network as a graph $G(S, T, E, W, L)$. The sensor set is denoted as S. T represents the set of targets in the network. If one target $t \in T$ can be covered by $s \in S$, then there is an edge (s, t) in G. In Fig. 1b, there is an edge between t_1 and s_1 since t_1 is in s_1's sensing range. All of these edges are stored in E. Heterogeneous sensors are considered. Different sensors may have different energy consumption to do the same tasks. In general, we have different weights of sensors. W denotes the weights of all sensors, and L denotes the energy capacity of all sensors in G. Based on the definition of G, the formal definition of MLCP is defined as follow.

Formal Definition of MLCP

Definition 1 (MLCP[3]) MLCP is that given $G = (\mathcal{S}, \mathcal{T}, \mathcal{E}, \mathcal{W}, \mathcal{L})$, find a set of sensor subsetsets and duration of each subset $(S_1, L_1), (S_2, L_2), \ldots, (S_k, L_k)$ in G to maximize $\sum_{i=1}^{k} L_i$, where S_i represents the sensor subset in G and L_i represents the time duration of S_i, satisfying:

1. $\forall i \in \{1, 2, \ldots, k\}$, S_i satisfies full coverage. $\forall\, t \in \mathcal{T}$ and $\forall S_i$, $\exists s \in S_i$ satisfying $(t, s) \in \mathcal{E}$.
2. For each sensor, the total active time should be smaller or equal to its power constraint.

It is a long-standing open problem whether Maximum Lifetime Coverage Problem (MLCP) has a polynomial-time constant-approximation algorithm [3].

Based on primal-dual method (PD method), *M*inimum *W*eight *S*ensor *C*overage *P*roblem (MWSCP) is used to help to solve MLCP. The formal definition of MWSCP is given in Definition 2.

Definition 2 (MWSCP[3]) MWSCP is to find a sensor subset SC in $G = (\mathcal{S}, \mathcal{T}, \mathcal{E}, \mathcal{W}, \mathcal{L})$ to minimize $\sum_{s \in SC} w(s)$, where $w(s)$ represents the weight of sensor s, such that $\forall\, t \in \mathcal{T}$, $\exists s \in SC$ satisfying $(t, s) \in \mathcal{E}$.

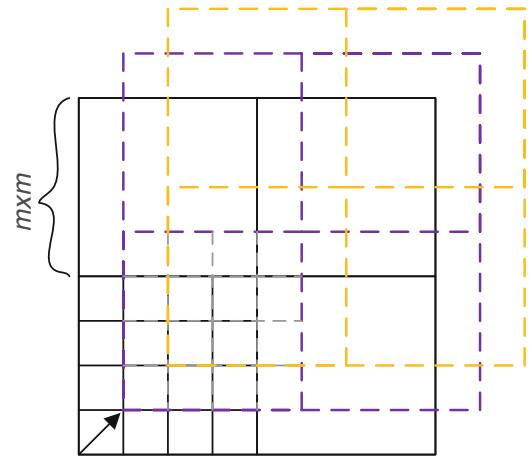
Key Results

1. Heuristic linear programming without performance guarantee [4]
2. Pure heuristic algorithm better than heuristic linear programming
3. One 4-approximation algorithms for MLCP with improvement from $1 + \ln n$, where n is the number of sensors
4. One 4-approximation algorithms for MWSCP

Integer Linear Programming and Heuristic Algorithms Proposed by Cardei et al. [4]

Cardei et al. prove that MLCP is NP-hard. Two algorithms are proposed [4]. The first algorithm is to model MLCP as an integer linear programming. To solve it, the authors first relax the integer variables to real values and get the optimal solution to the relaxed linear programming. Find the maximum time duration of each group based on the optimal solution to the relaxed linear programming. Update all sensors' remaining battery capacity. A new MLCP is formed with different remaining sensor power abilities after previous round. Finally, a maximum lifetime will be achieved in the network.

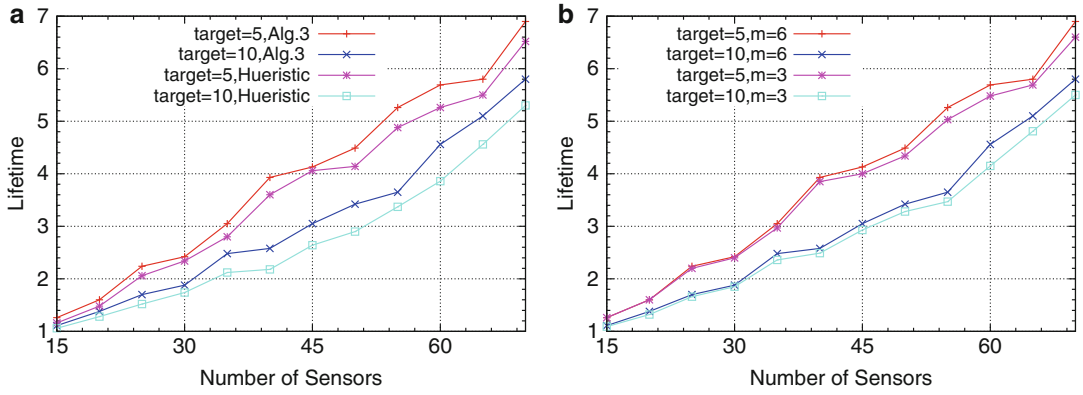
The second algorithm is a heuristic algorithm. Find a sensor group which can cover all targets. The lifetime of this group is determined by the minimum power ability of sensors in the group. Update all sensors' energy level and choose sensor group again till no such group can be found. The final lifetime is the sum of time duration of all sensor groups.



Maximum Lifetime Coverage, Fig. 2 Double partition and shifting [3]

Performance-Guaranteed Approximation Algorithm Proposed by Ling et al. [3]

Ling et al. use primal-dual method to solve MCLP. The primal problem is MWSCP [5]. To get a constant-approximation algorithm for MWSCP, double partition and shifting are used. As shown in Fig. 2, the area is divided into cells with size $\frac{m \times r}{\sqrt{2}} \times \frac{m \times r}{\sqrt{2}}$, where r represents sensing range of sensors and m is a predetermined value. $r = 1$ in Fig. 2. Each cell is further divided into



Maximum Lifetime Coverage, Fig. 3 Comparison of lifetime. (**a**) Lifetimes among different algorithms. (**b**) Lifetimes among different partitions [3]

small squares of size $\frac{r}{\sqrt{2}} \times \frac{r}{\sqrt{2}}$ which means there will be $m \times m$ small squares in each cell. After twice partition, there will be m horizontal strips and m vertical strips in each cell. Using dynamic programming, the optimal solution to MWSCP can be found in each cell. Combined all the optimal solutions to each cell, we can get an approximation algorithm to MWSCP for the whole area. To solve the sensors on the border of each cell, the small squares are shifted to purple position in Fig. 2 and then to yellow position. The shift will stop after m times. The final result will be achieved by taking average of the solutions to black partition, purple partition, yellow partition, and other shifts. It is proved in [3] that the final result of MWSCP has a constant performance ratio 4. The running time of the algorithm is determined by the predetermined value m. If m is big, the result will be more precise, but the running time is high. If m is small, the running time is small, but the performance will drop.

Based on the idea of primal-dual method, the solution to MWSCP will help to derive the solution to MCLP with the same performance ratio. The solution to MWSCP is derived iteration by iteration. In each iteration, find the MWSCP firstly and then determine the time duration of the sensor set. Update the lifetime and weight of each sensors. The algorithm will stop if there is no such a sensor set exists.

Experimental Results

Cardei et al. demonstrate that their pure heuristic algorithm outperforms their heuristic linear programming algorithm in running time and lifetime.

Ding et al. [3] conducts their experimental comparisons in an area of $6\sqrt{2} * 6\sqrt{2}$ and $m = 6$. They deploy sensors and targets randomly in that area. All sensors have the same sensing range of 2. The initial power capacity is 1 of each sensor. To show the density's effect on the performance, they increase the sensors from 15 to 70 by 5 and increase the number of targets from 5 to 10. Ling et al. compare their algorithm to Cardei's pure heuristic algorithm. If there are more sensors and the number of targets is fixed, the lifetime will be increased because there are more sensor groups (Fig. 3).

Recommended Reading

1. Bai X, Xuan D, Yun Z, Lai TH, Jia W (2008) Complete optimal deployment patterns for full-coverage and k-connectivity (k<=6) wireless sensor networks. In: Proceedings of ACM MobiHoc, Hong Kong
2. Balister P, Bollobas B, Sarkar A, Kumar S (2007) Reliable density estimates for coverage and connectivity in thin strips of finite length. In: Proceedings of ACM MobiCom, Montréal
3. Ding L, Wu W, Willson J, Wu L, Lu Z, Lee W (2012) Constant-approximation for target coverage problem in wireless sensor networks. In: INFOCOM, Orlando, pp 1584–1592

4. Cardei M, Thai MT, Li Y, Wu W (2005) Energy-efficient target coverage in wireless sensor networks. In: Proceedings of IEEE INFOCOM, Miami
5. Huang Y, Gao X, Zhang Z, Wu W (2009) A better constant-factor approximation for weighted dominating set in unit disk graph. J Comb Optim 18(2):179–194

Maximum Matching

Marcin Mucha
Faculty of Mathematics, Informatics and Mechanics, Institute of Informatics, Warsaw, Poland

Keywords

Algebraic graph algorithms; Fast matrix multiplication; Matching in graphs

Years and Authors of Summarized Original Work

2004; Mucha, Sankowski

Problem Definition

Let $G = (V, E)$ be an undirected graph, and let $n = |V|$, $m = |E|$. A *matching* in G is a subset $M \subseteq E$, such that no two edges of M have a common endpoint. A *perfect matching* is a matching of cardinality $n/2$. The most basic matching related problems are finding a *maximum matching* (i.e., a matching of maximum size) and, as a special case, finding a *perfect matching* if one exists. One can also consider the case where a weight function $w : E \to R$ is given and the problem is to find a *maximum weight matching*.

The maximum matching and maximum weight matching are two of the most fundamental algorithmic graph problems. They have also played a major role in the development of combinatorial optimization and algorithmics. An excellent account of this can be found in a classic monograph [11] by Lovász and Plummer devoted entirely to matching problems. A more up-to-date but also more technical discussion of the subject can be found in [19].

Classical Approach

Solving the maximum matching problem in time polynomial in n is a highly nontrivial task. The first such solution was given by Edmonds [3] in 1965 and has time complexity $O(n^3)$. Edmond's ingenious algorithm uses a combinatorial approach based on augmenting paths and blossoms. Several improvements followed, culminating in the algorithm with complexity $O(m\sqrt{n})$ given by Micali and Vazirani [12] in 1980 (a complete proof of the correctness of this algorithm was given much later by Vazirani [21], a nice exposition of the algorithm and its generalization to the weighted case can be found in a work of Gabow and Tarjan [4]). Beating this bound proved very difficult, several authors managed to achieve only a logarithmic speed-up for certain values of m and n. All these algorithms essentially follow the combinatorial approach introduced by Edmonds.

The maximum matching problem is much simpler for bipartite graphs. The complexity of $O(m\sqrt{n})$ was achieved for this case already in 1971 by Hopcroft and Karp [7], while the key ideas of the first polynomial algorithms date back to the 1920s and the works of König and Egerváry (see [11] and [19]).

Algebraic Approach

Around the time Micali and Vazirani introduced their matching algorithm, Lovász gave a randomized (Monte Carlo) reduction of the problem of testing whether a given n-vertex graph has a perfect matching to the problem of computing a certain determinant of a $n \times n$ matrix. Using the Hopcroft-Bunch fast Gaussian elimination algorithm [1], this determinant can be computed in time $MM(n) = O(n^\omega)$ – time required to multiply two $n \times n$ matrices. Since $\omega < 2.38$ (see [2, 20]), for dense graphs, this algorithm is asymptotically faster than the matching algorithm of Micali and Vazirani.

However, Lovász's algorithm only tests for perfect matching, it does not find it. Using it to find perfect/maximum matchings in a straightforward fashion yields algorithm with complexity $O(mn^{\omega}) = O(n^{4.38})$. A major open problem in the field was thus: can maximum matchings be actually found in $O(n^{\omega})$ time?

The first step in this direction was taken in 1989 by Rabin and Vazirani [16]. They showed that maximum matchings can be found in time $O(n^{\omega+1}) = O(n^{3.38})$.

Key Results

The following theorems state the key results of [13].

Theorem 1 *Maximum matching in a n-vertex graph G can be found in $O(n^3)$ time (Las Vegas) by performing Gaussian elimination on a certain matrix related to G.*

Theorem 2 *Maximum matching in an n-vertex bipartite graph can be found in $\tilde{O}(n^{\omega})$ time (Las Vegas) by performing a Hopcroft-Bunch fast Gaussian elimination on a certain matrix related to G.*

Theorem 3 *Maximum matching in an n-vertex graph can be found in $\tilde{O}(n^{\omega})$ time (Las Vegas).*

Note: $\tilde{O}$ notation suppresses polylogarithmic factors, so $\tilde{O}(f(n))$ means $O(f(n)\log^k(n))$ for some k.

Let us briefly discuss these results. Theorem 1 shows that effective matching algorithms can be simple. This is in large contrast to augmenting paths-/blossoms-based algorithms which are generally regarded as quite complicated.

The other two theorems show that, for dense graphs, the algebraic approach is asymptotically faster than the combinatorial one.

The algorithm for the bipartite case is very simple. It's only nonelementary part is the fast matrix multiplication algorithm used as black box by the Hopcroft-Bunch algorithm. The general algorithm, however, is complicated and uses strong structural results from matching theory. A natural question is whether or not it is possible to give a simpler and/or purely algebraic algorithm. This has been positively answered by Harvey [5].

Several other related results followed. Mucha and Sankowski [14] showed that maximum matchings in planar graphs can be found in time $\tilde{O}(n^{\omega/2}) = O(n^{1.19})$ which is currently fastest known. Yuster and Zwick [22] extended this to any excluded minor class of graphs. Harvey [6] described a significantly simpler and purely algebraic version of the algorithm for general graphs. Sankowski [17] gave an RNC work-efficient matching algorithm (see also Mulmuley et al. [15] and Karp et al. [9] for earlier, less efficient RNC matching algorithms, and Karloff [8] for a description of a general technique for making such algorithm Las Vegas). He also generalized Theorem 2 to the case of weighted bipartite graphs with integer weights from $[0, \ldots, W]$, showing that in this case maximum weight matchings can be found in time $\tilde{O}(Wn^{\omega})$ (see [18]).

Applications

The maximum matching problem has numerous applications, both in practice and as a subroutine in other algorithms. A nice discussion of practical applications can be found in the monograph [11] by Lovász and Plummer. It should be noted, however, that algorithms based on fast matrix multiplication are completely impractical, so the results discussed here are not really useful in these applications.

On the theoretical side, faster maximum (weight) matching algorithms yield faster algorithms for related problems: disjoint $s - t$ paths problem, the minimum (weight) edge cover problem, the (maximum weight) b-matching problem, the (maximum weight) b-factor problem, the maximum (weight) T-join, or the Chinese postman problem. For detailed discussion of all these applications, see [11] and [19].

The algebraic algorithm of Theorem 1 also has a significant educational value. The combinato-

rial algorithms for the general maximum matching problem are generally regarded too complicated for an undergraduate course. That is definitely not the case with the algebraic $O(n^3)$ algorithm.

Open Problems

One of the most important open problems in the area is generalizing the results discussed above to weighted graphs. Sankowski [18] gives a $\tilde{O}(Wn^{\omega})$ algorithm for bipartite graphs with integer weights from the interval $[0 \ldots W]$. The complexity of this algorithm is really bad in terms of W. No effective algebraic algorithm is known for general weighted graphs.

Another interesting but most likely very hard problem is the derandomization of the algorithms discussed.

Cross-References

- ▸ All Pairs Shortest Paths via Matrix Multiplication
- ▸ Assignment Problem

Recommended Reading

1. Bunch J, Hopcroft J (1974) Triangular factorization and inversion by fast matrix multiplication. Math Comput 125:231–236
2. Coppersmith D, Winograd S (1987) Matrix multiplication via arithmetic progressions. In: Proceedings of the 19th annual ACM conference on theory of computing (STOC), New York, pp 1–6
3. Edmonds J (1965) Paths, trees, and flowers. Can J Math 17:449–467
4. Gabow HN, Tarjan RE (1991) Faster scaling algorithms for general graph matching problems. J ACM 38(4):815–853
5. Harvey N (2006) Algebraic structures and algorithms for matching and matroid problems. In: Proceedings of the 47th annual IEEE symposium on foundations of computer science (FOCS), Berkeley
6. Harvey JAN (2009) Algebraic algorithms for matching and matroid problems. SIAM J Comput 39(2):679–702
7. Hopcroft JE, Karp RM (1973) An $O(n^{5/2})$ Algorithm for maximum matchings in bipartite graphs. SIAM J Comput 2:225–231
8. Karloff H (1986) A Las Vegas RNC algorithm for maximum matching. Combinatorica 6:387–391
9. Karp R, Upfal E, Widgerson A (1986) Constructing a perfect matching is in random NC. Combinatorica 6:35–48
10. Lovász L (1979) On determinants, matchings and random algorithms. In: Budach L (ed) Fundamentals of computation theory (FCT'79), Wendisch-Rietz, pp 565–574. Akademie-Verlag, Berlin
11. Lovász L, Plummer MD (1986) Matching theory. Akadémiai Kiadó – North Holland, Budapest
12. Micali S, Vazirani VV (1980) An $O(\sqrt{VE})$ algorithm for finding maximum matching in general graphs. In: Proceedings of the 21st annual IEEE symposium on foundations of computer science (FOCS), Syracuse, pp 17–27
13. Mucha M, Sankowski P (2004) Maximum matchings via Gaussian elimination. In: Proceedings of the 45th annual IEEE symposium on foundations of computer science (FOCS), Rome, pp 248–255
14. Mucha M, Sankowski P (2006) Maximum matchings in planar graphs via Gaussian elimination. Algorithmica 45:3–20
15. Mulmuley K, Vazirani UV, Vazirani VV (1987) Matching is as easy as matrix inversion. In: Proceedings of the 19th annual ACM conference on theory of computing, New York, pp 345–354. ACM
16. Rabin MO, Vazirani VV (1989) Maximum matchings in general graphs through randomization. J Algorithms 10:557–567
17. Sankowski P (2005) Processor efficient parallel matching. In: Proceeding of the 17th ACM symposium on parallelism in algorithms and architectures (SPAA), Las Vegas, pp 165–170
18. Sankowski P (2006) Weighted bipartite matching in matrix multiplication time. In: Proceedings of the 33rd international colloquium on automata, languages and programming, Venice, pp 274–285
19. Schrijver A (2003) Combinatorial optimization: polyhedra and efficiency. Springer, Berlin/Heidelberg
20. Vassilevska Williams V (2012) Multiplying matrices faster than Coppersmith-Winograd. In: Proceedings of the 44th symposium on theory of computing conference (STOC), New York, pp 887–898
21. Vazirani VV (1994) A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{VE})$ maximum matching algorithm. Combinatorica 14(1):71–109
22. Yuster R, Zwick U (2007) Maximum matching in graphs with an excluded minor. In: Proceedings of the ACM-SIAM symposium on discrete algorithms (SODA), New Orleans

Maximum-Average Segments

Kun-Mao Chao
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

Keywords

Maximum-average segment

Years and Authors of Summarized Original Work

1994; Huang

Problem Definition

Given a sequence of numbers, $A=\langle a_1,a_2,\ldots,a_n\rangle$, and two positive integers L, U, where $1\leq L \leq U \leq n$, the maximum-density segment problem is to find a consecutive subsequence, i.e., a segment or substring, of A with length at least L and at most U such that the average value of the numbers in the subsequence is maximized.

Key Results

If there is no length constraint, then obviously the maximum-density segment is the maximum number in the sequence. Let's first consider the problem where only the length lower bound L is imposed. By observing that the length of the shortest maximum-density segment with length at least L is at most $2L-1$, Huang [9] gave an $O(nL)$-time algorithm. Lin et al. [13] proposed a new technique, called the *right-skew decomposition*, to partition each suffix of A into *right-skew* segments of strictly decreasing averages. The right-skew decomposition can be done in $O(n)$ time, and it can answer, for each position i, a consecutive subsequence of A starting at that position such that the average value of the numbers in the subsequence is maximized. On the basis of the right-skew decomposition, Lin et al. [13] devised an $O(n \log L)$-time algorithm for the maximum-density segment problem with a lower bound L, which was improved to $O(n)$ time by Goldwasser et al. [8]. Kim [11] gave another $O(n)$-time algorithm by reducing the problem to the maximum-slope problem in computation geometry. As for the problem which takes both L and U into consideration, Chung and Lu [6] bypassed the construction of the right-skew decomposition and gave an $O(n)$-time algorithm.

It should be noted that a closely related problem in data mining, which basically deals with a binary sequence, was independently formulated and studied by Fukuda et al. [7].

An Extension to Multiple Segments

Given a sequence of numbers, $A = \langle a_1, a_2, \ldots, a_n\rangle$, and two positive integers L and k, where $k \leq \frac{n}{L}$, let $d(A[i, j])$ denote the *density* of segment $A[i, j]$, defined as $(a_i + a_{i+1} + \cdots + a_j)/(j - i + 1)$. The problem is to find k disjoint segments $\{s_1, s_2, \ldots, s_k\}$ of A, each has a length of at least L, such that $\sum_{1\leq i\leq k} d(s_i)$ is maximized. Chen et al. [5] proposed an $O(nkL)$-time algorithm and an improved $O(nL + k^2L^2)$-time algorithm was given by Bergkvist and Damaschke [2]. Liu and Chao [14] gave an $O(n+k^2L \log L)$-time algorithm.

Applications

In all organisms, the GC base composition of DNA varies between 25–75 %, with the greatest variation in bacteria. Mammalian genomes typically have a GC content of 45–50 %. Nekrutenko and Li [15] showed that the extent of the compositional heterogeneity in a genomic sequence strongly correlates with its GC content. Genes are found predominantly in the GC-richest isochore classes. Hence, finding GC-rich regions is an important problem in gene recognition and comparative genomics.

Given a DNA sequence, one would attempt to find segments of length at least L with the highest C+G ratio. Specifically, each of nucleotides C and G is assigned a score of 1, and each of nucleotides A and T is assigned a score of 0.
DNA sequence: *ATGACTCGAGCTCGTCA*
Binary sequence: *0010101101101010* The maximum-average segments of the binary sequence correspond to those segments with the highest GC ratio in the DNA sequence. Readers can refer to [1, 3, 4, 11–13, 16–18] for more variants and applications.

Open Problems

The best asymptotic time bound of the algorithms for the multiple maximum-density segments problem is $O(n + k^2 L \log L)$. Can this problem be solved in $O(n)$ time?

Cross-References

▸ Maximum-Sum Segments

Recommended Reading

1. Arslan A, Eğecioğlu Ö, Pevzner P (2001) A new approach to sequence comparison: normalized sequence alignment. Bioinformatics 17:327–337
2. Bergkvist A, Damaschke P (2005) Fast algorithms for finding disjoint subsequences with extremal densities. In: Proceedings of the 16th annual international symposium on algorithms and computation, Sanya. Lecture notes in computer science, vol 3827, pp 714–723
3. Burton BA (2011) Searching a bitstream in linear time for the longest substring of any given density. Algorithmica 61:555–579
4. Burton BA, Hiron M (2013) Locating regions in a sequence under density constraints. SIAM J Comput 42:1201–1215
5. Chen YH, Lu HI, Tang CY (2005) Disjoint segments with maximum density. In: Proceedings of the 5th annual international conference on computational science, Atlanta, pp 845–850
6. Chung K-M, Lu H-I (2004) An optimal algorithm for the maximum-density segment problem. SIAM J Comput 34:373–387
7. Fukuda T, Morimoto Y,Morishita S, Tokuyama T (1999) Mining optimized association rules for numeric attributes. J Comput Syst Sci 58:1–12
8. Goldwasser MH, Kao M-Y, Lu H-I (2005) Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications. J Comput Syst Sci 70:128–144
9. Hsieh Y-H, Yu C-C, Wang B-F (2008) Optimal algorithms for the interval location problem with range constraints on length and average. IEEE/ACM Trans Comput Biol Bioinform 5:281–290
10. Huang X (1994) An algorithm for identifying regions of a DNA sequence that satisfy a content requirement. Comput Appl Biosci 10:219–225
11. Kim SK (2003) Linear-time algorithm for finding a maximum-density segment of a sequence. Inf Process Lett 86:339–342
12. Lin Y-L, Huang X, Jiang T, Chao K-M (2003) MAVG: locating non-overlapping maximum average segments in a given sequence. Bioinformatics 19:151–152
13. Lin Y-L, Jiang T, Chao K-M (2002) Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. J Comput Syst Sci 65:570–586
14. Liu H-F, Chao K-M (2006) On locating disjoint segments with maximum sum of densities. In: Proceedings of the 17th annual international symposium on algorithms and computation, Kolkata. Lecture notes in computer science, vol 4288, pp 300–307
15. Nekrutenko A, Li WH (2000) Assessment of compositional heterogeneity within and between eukaryotic genomes. Genome Res 10:1986–1995
16. Stojanovic N, Dewar K (2005) Identifying multiple alignment regions satisfying simple formulas and patterns. Bioinformatics 20:2140–2142
17. Stojanovic N, Florea L, Riemer C, Gumucio D, Slightom J, Goodman M, Miller W, Hardison R (1999) Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions. Nucl Acid Res 19:3899–3910
18. Zhang Z, Berman P, Wiehe T, Miller W (1999) Post-processing long pairwise alignments. Bioinformatics 15:1012–1019

Maximum-Sum Segments

Kun-Mao Chao
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

Keywords

Shortest path; Longest path

Years and Authors of Summarized Original Work

2002; Lin, Jiang, Chao

Problem Definition

Given a sequence of numbers, $A = \langle a_1, a_2, \ldots, a_n \rangle$, and two positive integers L, U, where $1 \leq L \leq U \leq n$, the maximum-sum segment problem is to find a consecutive subsequence, i.e., a segment or substring, of A with length at least L and at most U such that the sum of the numbers in the subsequence is maximized.

Key Results

The maximum-sum segment problem without length constraints is linear-time solvable by using Kadane's algorithm [2]. Huang extended the recurrence relation used in [2] for solving the maximum-sum segment problem and derived a linear-time algorithm for computing the maximum-sum segment with length at least L. Lin et al. [13] proposed an $O(n)$-time algorithm for the maximum-sum segment problem with both L and U constraints, and an online version was given by Fan et al. [10].

An Extension to Multiple Segments

Computing the k largest sums over all possible segments is a natural extension of the maximum-sum segment problem. This extension has been considered from two perspectives, one of which allows the segments to overlap, while the other disallows.

Linear-time algorithms for finding all the nonoverlapping maximal segments were given in [5, 15]. On the other hand, one may focus on finding the k maximum-sum segments whose overlapping is allowed. A naïve approach is to choose the k largest from the sums of all possible contiguous subsequences which requires $O(n^2)$ time. Bae and Takaoka [1] presented an $O(kn)$-time algorithm for the k maximum segment problem. Liu and Chao [14] noted that the k maximum-sum segment problem can be solved in $O(n + k)$ time [9] and gave an $O(n + k)$-time algorithm for the length-constrained k maximum-sum segment problem.

Applications

The algorithms for the maximum-sum segment problem have applications in finding GC-rich regions in a genomic DNA sequence, postprocessing sequence alignments, and annotating multiple sequence alignments. Readers can refer to [3–8, 11, 13, 15–18] for more variants and applications.

Open Problems

It would be interesting to consider the higher dimensional cases.

Cross-References

▸ Maximum-Average Segments

Recommended Reading

1. Bae SE, Takaoka T (2004) Algorithms for the problem of k maximum sums and a VLSI algorithm for the k maximum subarrays problem. In: Proceedings of the 7th international symposium on parallel architectures, algorithms and networks, Hong Kong, pp 247–253
2. Bentley J (1986) Programming pearls. Addison-Wesley, Reading
3. Burton BA (2011) Searching a bitstream in linear time for the longest substring of any given density. Algorithmica 61:555–579
4. Burton BA, Hiron M (2013) Locating regions in a sequence under density constraints. SIAM J Comput 42:1201–1215
5. Chen K-Y, Chao K-M (2004) On the range maximum-sum segment query problem. In: Proceedings of the 15th international symposium on algorithms and computation, Hong Kong. LNCS, vol 3341, pp 294–305
6. Chen K-Y, Chao K-M (2005) Optimal algorithms for locating the longest and shortest segments satisfying a sum or an average constraint. Inf Process Lett 96:197–201

7. Cheng C-H, Chen K-Y, Tien W-C, Chao K-M (2006) Improved algorithms for the k maximum-sum problems. Theor Comput Sci 362:162–170
8. Csűrös M (2004) Maximum-scoring segment sets. IEEE/ACM Trans Comput Biol Bioinform 1:139–150
9. Eppstein D (1998) Finding the k shortest paths. SIAM J Comput 28:652–673
10. Fan T-H, Lee S, Lu H-I, Tsou T-S, Wang T-C, Yao A (2003) An optimal algorithm for maximum-sum segment and its application in bioinformatics. In: Proceedings of the eighth international conference on implementation and application of automata, Santa Barbara. LNCS, vol 2759, pp 251–257
11. Hsieh Y-H, Yu C-C, Wang B-F (2008) Optimal algorithms for the interval location problem with range constraints on length and average. IEEE/ACM Trans Comput Biol Bioinform 5:281–290
12. Huang X (1994) An algorithm for identifying regions of a DNA sequence that satisfy a content requirement. Comput Appl Biosci 10:219–225
13. Lin Y-L, Jiang T, Chao K-M (2002) Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. J Comput Syst Sci 65:570–586
14. Liu H-F, Chao K-M (2008) Algorithms for finding the weight-constrained k longest paths in a tree and the length-constrained k maximum-sum segments of a sequence. Theor Comput Sci 407:349–358
15. Ruzzo WL, Tompa M (1999) A linear time algorithm for finding all maximal scoring subsequences. In: Proceedings of the 7th international conference on intelligent systems for molecular biology, Heidelberg, pp 234–241
16. Stojanovic N, Dewar K (2005) Identifying multiple alignment regions satisfying simple formulas and patterns. Bioinformatics 20:2140–2142
17. Stojanovic N, Florea L, Riemer C, Gumucio D, Slightom J, Goodman M, Miller W, Hardison R (1999) Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions. Nucleic Acids Res 19:3899–3910
18. Zhang Z, Berman P, Wiehe T, Miller W (1999) Post-processing long pairwise alignments. Bioinformatics 15:1012–1019

Max-Min Allocation

Deeparnab Chakrabarty
Microsoft Research, Bangalore, Karnataka, India

Keywords

Approximation algorithms; Linear programs; Scheduling and resource allocation

Years and Authors of Summarized Original Work

2006; Bansal, Sviridenko
2007; Asadpour, Saberi
2008; Feige
2009; Chakrabarty, Chuzhoy, Khanna

Problem Definition

The max-min allocation problem has the following setting. There is a set A of m agents and a set I of n items. Each agent $i \in A$ has utility $u_{ij} \in \mathbb{R}_{\geq 0}$ for item $j \in I$. Given a subset of items $S \subseteq I$, the utility of this set to agent i is denoted as $u_i(S) := \sum_{j \in S} u_{ij}$. The max-min allocation problem is to find an allocation of items to agents such that the minimum utility among the agents is maximized. That is, $\min_{i \in A} u_i(S_i)$ is maximized, where $S_i \subseteq I$ is the set of items allocated to agent i and $S_i \cap S_{i'} = \emptyset$.

The problem naturally arises as an approach to maximize fairness. Fairness is an important concept arising in numerous settings ranging from border disputes in political science to frequency allocations in spectrum auctions. Max-min fairness is one of the standard notions of fairness and has been an object of study for decades [6]. Most of the older works, however, have focussed on divisible settings, that is, situations where the resource can be infinitely divided and allocated. Furthermore, the computational perspective, that is, how *efficiently* can one find a fair allocation, has not been a primary viewpoint. The max-min allocation problem is a combinatorial allocation problem where the items cannot be divided, and the interest is in designing polynomial time algorithms to obtain fair, or near-fair, allocations.

Key Results

The max-min allocation problem is NP-hard and the focus is on designing approximation algorithms. Let OPT be the optimum value of a certain instance. A ρ-approximate solution, for $\rho > 1$, is an allocation where each agent gets

utility at least OPT/ρ. A ρ-approximation algorithm returns a ρ-approximate solution given any instance. An algorithm is a polynomial time approximation scheme (PTAS) if for any constant $\varepsilon > 0$, it returns an $(1+\varepsilon)$-approximate solution in polynomial time.

Woeginger [11] obtained a PTAS for the max-min allocation problem when the utility of an item is the *same* for all agents. Bezakova and Dani [5] gave the first nontrivial $(n-m+1)$-approximation algorithm for the general max-min allocation problem and also showed that it is NP-hard to obtain a better than 2-approximation algorithm for the problem. The latter result remains the best hardness known till date.

Bansal and Sviridenko [3] introduced a restricted version of the max-min allocation problem which they called the *Santa Claus* problem. In this version, each item has an inherent utility u_j, however, it can only be allocated to an agent in a certain subset $A_j \subseteq A$. Equivalently, for each item j, $u_{ij} \in \{u_j, 0\}$. Bansal and Sviridenko [3] described an $O(\log\log m/\log\log\log m)$-approximation algorithm for the Santa Claus problem. Soon after, Feige [8] described an algorithm which *estimates* the value of the optimum of the Santa Claus problem up to $O(1)$-factor in polynomial time, although at the time no efficient algorithm was known to construct the allocation. Following constructive versions of the Lovasz local lemma due to Moser and Tardos [10] and Haeupler et al. [9], there now exists a polynomial time $O(1)$-approximation algorithm for the Santa Claus problem. The constant, however, is necessarily quite large and to our knowledge has not been explicitly specified in any published work. In contrast, Asadpour et al. [2] described a local search algorithm which returns a 4-approximate solution to the Santa Claus problem; however, it is not known whether the procedure terminates in polynomial time or not.

Asadpour and Saberi [1] described a polynomial time $O(\sqrt{m}\log^3 m)$ approximation algorithm for the general max-min allocation problem. Bateni et al. [4] obtained an $O(m^\varepsilon)$-approximation algorithm running in $m^{O(1/\varepsilon)}$ time for certain special cases of the max-min allocation problem; in their special cases, utilities u_{ij} lay in the set $\{0, 1, \infty\}$, and furthermore, for each item j there exists at most one agent i with $u_{ij} = \infty$. Chakrabarty et al. [7] designed an $O(n^\varepsilon)$-approximation algorithm for the general max-min allocation problem which runs in $n^{O(1/\varepsilon)}$-time, for any $\varepsilon > \frac{9\log\log n}{\log n}$. This implies *quasi-polynomial* time $O(\log^{10} n)$-approximation algorithm and $O(m^\varepsilon)$-approximation algorithm, for any constant $\varepsilon > 0$, for the max-min allocation problem. An algorithm runs in quasi-polynomial time, if the logarithm of its running time is upper bounded by a polynomial in the bit length of the data.

Sketch of the Techniques

Almost all algorithms for the max-min allocation problem follow by rounding linear programming (LP) relaxations of the problem. One starts with a guess T of the optimum OPT. Using this, one writes an LP which has a feasible solution if $\mathsf{OPT} \geq T$. The nontrivial part is to round this LP solution to obtain an allocation with every agent getting utility $\geq T/\rho$. Since, by doing a binary search over the guesses, one can get T very close to OPT, the rounding step implies a ρ-approximation algorithm. Henceforth, we assume that T has been guessed to be OPT, and furthermore, by scaling all utility values appropriately, we assume $\mathsf{OPT} = 1$.

The first LP relaxation one may think of is the following. First one clips each utility value at 1; $u_{ij} = \min(1, u_{ij})$. If $\mathsf{OPT} = 1$, then the following LP is feasible.

$$\sum\nolimits_{j\in I} u_{ij}x_{ij} \geq 1, \qquad \forall i \in A \qquad (1)$$

$$\sum\nolimits_{i\in A} x_{ij} = 1, \qquad \forall j \in I \qquad (2)$$

The first inequality states that every agent gets utility at least $\mathsf{OPT} = 1$, and the second states that each item is allocated. It is not hard to find instances, and in fact Santa Claus instances, where the LP is feasible for $\mathsf{OPT} = 1$, but in any allocation, some agent will obtain util-

ity at most $1/m$. In other words, the *integrality gap* of this LP relaxation is (at least) m.

There is a considerably stronger LP relaxation which is called the *configuration* LP relaxation. The variables in this LP are of the form $y_{i,C}$ where $i \in A$ and $C \subseteq I$ where $\sum_{j \in C} u_{ij} \geq 1$. (i, C) is called a feasible configuration in this case. $\mathcal{C}$ denotes the collection of all feasible configurations.

$$\sum_{C} y_{i,C} = 1, \quad \forall i \in A \tag{3}$$

$$\sum_{i \in A} \sum_{C:(i,C) \in \mathcal{C}, j \in C} y_{i,C} = 1, \quad \forall j \in I \tag{4}$$

The first inequality states that each agent precisely gets one subset of items, and the second states that each item is in precisely one feasible configuration. Although the LP has possibly exponentially variables, the dual has only polynomially many variables and can be solved to arbitrary accuracy via the ellipsoid method. We refer the reader to [3] for details. Bansal and Sviridenko [3] show that in the Santa Claus problem, a solution above LP can be rounded to give an allocation where every agent obtains utility $\rho = \Omega\left(\frac{\log\log\log n}{\log\log n}\right)$. To do this, the authors partition the items into *big*, if $u_j \geq \rho$ and *small* otherwise. A solution is ρ-approximate if any agent gets either one big item or $\geq 1/\rho$ small items. The big items are taken care of via a "matching like" procedure, while the small items are allocated by randomized rounding. Bansal and Sviridenko [3] use the Lovasz local lemma (LLL) to analyze the randomized algorithm. Feige [8] uses a more sophisticated randomized rounding in phases along with an LLL analysis to obtain a constant factor approximation to the value of the optimum. At the time, no algorithmic proofs of LLL were known; however, following works of [9, 10], a polynomial time $O(1)$-approximation algorithm is now known for the Santa Claus problem.

In the general max-min allocation, the main problem in generalizing the above technique is that the same item could be big for one agent and small for another agent. Asadpour and Saberi [1] give a two-phase rounding algorithm. In the first phase, a random matching is obtained between agents and *their* big items with roughly the property that each item is allocated with the same probability that the configuration LP prescribes. In the second phase, each agent i randomly selects a set C of items with probability $y_{i,C}$. Since there are m agents, at most m items are allocated in the first phase. This allows [1] to argue that, with high probability, there is enough (roughly $1/\sqrt{m}$) utility remaining among the unmatched items in C. Finally, they also show that the same item is not "claimed" by not more than $O(\log m)$ agents.

The integrality gap of the configuration LP is $\Omega(\sqrt{m})$. Therefore a new LP relaxation is required to go beyond the Asadpour-Saberi result. We now briefly sketch the technique of Chakrabarty et al. [7]. First, they show that any instance can be "reduced" to a canonical instance where *agents* are either heavy or light. Heavy agents have utility 1 for a subset of big items and 0 for the rest. Light agents have a unique *private* item which give them utility 1, and the rest of the items either are small and give utility $1/K$ or give utility 0. Here $K \approx n^{\varepsilon}$ is a large integer. The LP of [7] is parametrized by a maximum matching M between heavy agents and their big items. If all heavy agents are matched, then there is nothing to be done since light agents can allocate their private item. Otherwise, there is a *reassignment* strategy: where a light agent is allocated K small items upon which he "frees" his private item, which is then again allocated to another agent and so on, till an unmatched heavy agent gets a big item. This reassignment can be seen as a directed in-arborescence whose depth can be argued is at most $1/\varepsilon$ since at each level we encounter roughly K new light agents. The LP encodes this reassignment as a flow with a variable for each flow path of length at most $1/\varepsilon$; this implies the number of variables is at most $n^{O(1/\varepsilon)}$ and a similar number of constraints. Therefore, the LP can be solved in $n^{O(1/\epsilon)}$ time which dominates the running time. If the instance has $\mathsf{OPT} = 1$, then the LP has a feasible solution. One would then expect that given such a feasible

solution, one can obtain an allocation with every heavy agent getting a big item and each light agent getting either his private item or n^{ε} small items. Unfortunately, this may not be true. What Chakrabarty et al. [7] show is that if the LP has a feasible solution, then a "partial" allocation can be found where some light agents obtain sufficiently many small items, and their private items can be used to obtain a larger matching M' among heavy agents and big items. The process is then repeated iteratively, with a new LP at each step guiding the partial allocation, till one matches every heavy agent.

Summary

In summary, the best polynomial time algorithms for the general max-min allocation problem, as of the date this article is written, have approximation factors which are a polynomial in the input data. On the other hand, even a 2-approximation for the problem has not been ruled out. Closing this gap is a confounding problem in the area of approximation algorithms. A constant factor approximation algorithm is known for the special case called the Santa Claus problem; even here, getting a polynomial time algorithm achieving a "small" constant factor is an interesting problem.

Recommended Reading

1. Asadpour A, Saberi A (2010) An approximation algorithm for max-min fair allocation of indivisible goods. SIAM J Comput 39(7):2970–2989
2. Asaspour A, Feige U, Saberi A (2012) Santa Claus meets hypergraph matching. ACM Trans Algorithms 8(3):24
3. Bansal N, Sviridenko M (2006) The Santa Claus problem. In: ACM symposium on theory of computing (STOC), Seattle
4. Bateni M, Charikar M, Guruswami V (2009) Maxmin allocation via degree lower-bounded arborescences. In: ACM symposium on theory of computing (STOC), Bethesda
5. Bezakova I, Dani V (2005) Allocating indivisible goods. SIGecom Exch 5(3):11–18
6. Brams S, Taylor A (1996) Fair division: from cake-cutting to dispute resolution. Cambridge University Press, Cambridge/New York
7. Chakrabarty D, Chuzhoy J, Khanna S (2009) On allocations that maximize fairness. In: Proceedings, IEEE symposium on foundations of computer science (FOCS), Atlanta
8. Feige U (2008) On allocations that maximize fairness. In: Proceedings, ACM-SIAM symposium on discrete algorithms (SODA), San Francisco
9. Haeupler B, Saha B, Srinivasan A (2011) New constructive aspects of the lovász local lemma. J ACM 58(6)
10. Moser R, Tardos G (2010) A constructive proof of the general lovász local lemma. J ACM 57(2)
11. Woeginger G (1997) A polynomial-time approximation scheme for maximizing the minimum machine completion time. Oper Res Lett 20(4):149–154

Mechanism Design and Differential Privacy

Kobbi Nisim[1] and David Xiao[2]
[1]Department of Computer Science, Ben-Gurion University, Beer Sheva, Israel
[2]CNRS, Université Paris 7, Paris, France

Keywords

Differential privacy; Mechanism design; Privacy-aware mechanism design; Purchasing privacy

Years and Authors of Summarized Original Work

2007; McSherry, Talwar
2011; Ghosh, Roth
2012; Nissim, Orlandi, Smorodinsky
2012; Fleischer, Lyu
2012; Ligett, Roth
2013; Chen, Chong, Kash, Moran, Vadhan
2014; Nissim, Vadhan, Xiao

Problem Definition

Mechanism design and private data analysis both study the question of performing computations over data collected from individual agents while satisfying additional restrictions. The focus in mechanism design is on performing computations that are compatible with the incentives of

the individual agents, and the additional restrictions are toward motivating agents to participate in the computation (individual rationality) and toward having them report their true data (incentive compatibility). The focus in private data analysis is on performing computations that limit the information leaked by the output on each individual agent's sensitive data, and the additional restriction is on the influence each agent may have on the outcome distribution (differential privacy). We refer the reader to the sections on *algorithmic game theory* and on *differential privacy* for further details and motivation.

Incentives and privacy. In real-world settings, incentives influence how willing individuals are to part with their private data. For example, an agent may be willing to share her medical data with her doctor, because the utility from sharing is greater than the loss of utility from privacy concerns, while she would probably not be willing to share the same information with her accountant.

Furthermore, privacy concerns can also cause individuals to misbehave in otherwise incentive-compatible, individually rational mechanisms. Consider for example a second-price auction: the optimal strategy in terms of payoff is to truthfully report valuations, but an agent may consider misreporting (or abstaining) because the outcome reveals the valuation of the second-price agent, and the agent does not want to risk their valuation being revealed. In studies based on sensitive information, e.g., a medical study asking individuals to reveal whether they have syphilis, a typical individual with syphilis may be less likely to participate than a typical individual without the disease, thereby skewing the overall sample. The bias may be reduced by offering appropriate compensation to participating agents.

The framework. Consider a setting with n individual agents, and let $x_i \in X$ be the private data of agent i for some type set X. Let $f : X^n \to Y$ be a function of the joint inputs of the agents $\mathbf{x} = (x_1, \ldots, x_n)$. Our goal is to build a mechanism M that computes $f(\mathbf{x})$ accurately and is compatible with incentives and privacy as we will now describe.

We first fix a function v that models the gain in utility that an agent derives from the *outcome* of the mechanism. We restrict our attention to a setting where this value can only depend on the agent's data and the outcome y of the mechanism:

$$v_i = v(x_i, y).$$

We also fix a function λ that models the loss in utility that an agent incurs because information about her private data is leaked by the outcome of the mechanism. Importantly, λ depends on the mechanism M, as the computation M performs determines the leakage. The loss can also depend on how much the agent values privacy, described by a parameter p_i (a real number in our modeling), on the actual data of all the individuals, on the outcome, as well as other parameters such as the strategy of the agent:

$$\lambda_i = \lambda(M, p_i, \mathbf{x}_{-i}, x_i, y, \ldots).$$

The overall utility that agent i derives from participating in the computation of M is

$$u_i = v_i - \lambda_i. \tag{1}$$

With this utility function in mind, our goal will be to construct truthful mechanisms M that compute f accurately. We note that in Eq. 1 we typically think about both v_i and λ_i as positive quantities, but we do not exclude either of them being negative, so either quantity may result in a gain or a loss in utility.

We can now define the mechanism $M : X^n \times \mathbb{R}^n \to Y$ to be a randomized function taking as inputs the private inputs of the agents $\mathbf{x}$ and their privacy valuations $\mathbf{p}$ and returns a value in the set Y.

Modeling the privacy loss. In order to analyze specific mechanisms, we will need to be able to control the privacy loss λ. Toward this end, we will need to assume that λ has some structure, and so we now discuss the assumptions we make and their justifications.

One view of privacy loss is to consider a framework of *sequential games*: an individual

is not only participating in mechanism M, but she will also participate in other mechanisms $M', M'', \ldots$ in the future, and each participation will cause her to gain or lose in utility. Because her inputs to these functions may be correlated, revealing her private inputs in M may cause her to obtain less utility in the future. For example, an individual may hesitate to participate in a medical study because doing so might reveal she has a genetic predisposition to a certain disease, which may increase her insurance premiums in the future. This view is general and can formalize many of the concerns we typically associate with privacy: discrimination because of medical conditions, social ostracism, demographic profiling, etc.

The main drawback of this view is that it is difficult to know what the future mechanisms $M', M'', \ldots$ may be. However, if M is differentially private, then participating in M entails a guarantee that remains meaningful *even without knowing the future mechanisms*. To see this, we will use the following definition that is equivalent to the definition of ϵ-differential privacy [3]:

Definition 1 (Differential privacy) A (randomized) mechanism $M : X^n \to Y$ is ϵ-differentially private if for all $\mathbf{x}, \mathbf{x}' \in X^n$ that differ on one entry, and for all $g : Y \to [0, \infty)$, it holds that

$$\text{Exp}[g(M(\mathbf{x}))] \leq e^{\epsilon} \cdot \text{Exp}[g(M(\mathbf{x}'))],$$

where the expectation is over the randomness introduced by the mechanism M.

Note that $e^{\epsilon} \approx 1 + \epsilon$ for small ϵ; thus, if $g(y)$ models the expected utility of an individual tomorrow given that the result of $M(x) = y$ today, then by participating in a differentially private mechanism, the individual's utility will change by at most ϵ.

Fact 1. Let $g : Y \to [-1, 1]$. If M is ϵ-differential private, then $\text{Exp}[g(M(\mathbf{x}'))] - \text{Exp}[g(M(\mathbf{x}))] \leq 2(e^{\epsilon} - 1) \approx 2\epsilon$ for all $\mathbf{x}, \mathbf{x}' \in X^n$ that differ on one entry.

To see why this is true, let $g_-(y) = \mathbf{max}(0, -g(y))$ and $g_+(y) = \mathbf{max}(0, g(y))$. From Definition 1 and the bound on the outcome of g, we get that $\text{Exp}[g_+(M(\mathbf{x}'))] - \text{Exp}[g_+(M(\mathbf{x}))] \leq (e^{\epsilon} - 1) \cdot \text{Exp}[g_+(M(\mathbf{x}))] \leq e^{\epsilon} - 1$ and, similarly, $\text{Exp}[g_-(M(\mathbf{x}))] - \text{Exp}[g_-(M(\mathbf{x}'))] \leq e^{\epsilon} - 1$. As $g(y) = g_+(y) - g_-(y)$, we conclude that $\text{Exp}[g(M(\mathbf{x}'))] - \text{Exp}[g(M(\mathbf{x}))] \leq 2(e^{\epsilon} - 1)$.

With this in mind, we typically view λ as being "bounded by differential privacy" in the sense that if M is ϵ-differentially private, then $|\lambda_i| \leq p_i \cdot \epsilon$, where p_i (a positive real number) is an upper bound on the maximum value of $2|g(y)|$. In certain settings we make even more specific assumptions about λ_i, and these are discussed in the sequel.

Generic Problems

We will discuss two generic problems for which key results will be given in the next section:

Privacy-aware mechanism design. Given an optimization problem $q : X^n \times Y \to \mathbb{R}$, construct a privacy-aware mechanism whose output $\hat{y}$ approximately maximizes $q(\mathbf{x}, \cdot)$. Using the terminology above, this corresponds to setting $f(\mathbf{x}) = \mathbf{argmax}_y q(\mathbf{x}, y)$, and the mechanism is said to compute $f()$ with accuracy α if (with high probability) $q(\mathbf{x}, f(\mathbf{x})) - q(\mathbf{x}, \hat{y}) \leq \alpha$. We mention two interesting instantiations of $q()$. When $q(\mathbf{x}, y) = \sum_i v(x_i, y)$, the problem is of maximizing social welfare. When x_i corresponds to how agent i values a digital good and $Y = \mathbb{R}^+$ is interpreted as a price for the good, setting $q(\mathbf{x}, y) = y \cdot |i : x_i \geq y|$ corresponds to maximizing the revenue from the good.

Purchasing privacy. Given a function $f : X^n \to Y$, construct a mechanism computing payments to agents for eliciting permission to use (some of) the entries of $\mathbf{x}$ in an approximation for $f(\mathbf{x})$. Here it is assumed that the agents cannot lie about their private values (but can misreport their privacy valuations). We will consider two variants of the problem. In the *insensitive value model*, agents only care about the privacy of their private values $\mathbf{x}$. In the *sensitive value model*, agents also care about the privacy of their

privacy valuations $\mathbf{p}$, e.g., because there may be a correlation between x_i and p_i.

Basic Differentially Private Mechanisms

We conclude this section with two differentially private mechanisms that are used in the constructions presented in the next section.

The Laplace mechanism [3]. The Laplace distribution with parameter $1/\epsilon$, denoted $\text{Lap}(1/\epsilon)$, is a continuous probability distribution with zero mean and variance $2/\epsilon$. The probability density function of $\text{Lap}(1/\epsilon)$ is $h(z) = \frac{\epsilon}{2} e^{-\epsilon|z|}$. For $\Delta \geq 0$ we get $\Pr_{Z\sim\text{Lap}(1/\epsilon)}[|Z| > \Delta] = e^{-\epsilon\Delta}$.

Fact 2. The mechanism M_{Lap} that on input $\mathbf{x} \in \{0,1\}^n$ outputs $y = \#\{i : x_i = 1\} + Z$ where $Z \sim \text{Lap}(1/\epsilon)$ is ϵ-differentially private. From the properties of the Laplace distribution, we get that

$$\Pr_{y\sim M_{\text{Lap}}(\mathbf{x})}[|y - \#\{i : x_i = 1\}| > \Delta] \leq e^{-\epsilon\Delta}.$$

The exponential mechanism [8]. Consider the optimization problem defined by $q : X^n \times Y \to \mathbb{R}$, where q satisfies $|q(\mathbf{x}, y) - q(\mathbf{x}', y)| \leq 1$ for all $y \in Y$ and all $\mathbf{x}, \mathbf{x}'$ that differ on one entry.

Fact 3. The mechanism M_{Exp} that on input $\mathbf{x} \in X^n$ outputs $y \in Y$ chosen according to

$$\Pr[y = t] = \frac{\exp\left(\frac{\epsilon}{2} q(\mathbf{x}, t)\right)}{\sum_{\ell\in Y} \exp\left(\frac{\epsilon}{2} q(\mathbf{x}, \ell)\right)} \qquad (2)$$

is ϵ-differentially private. Moreover,

$$\Pr_{y\sim M_{\text{Exp}}(\mathbf{x})}[q(\mathbf{x}, y) \geq \text{opt}(\mathbf{x}) - \Delta] \geq 1 - |Y| \cdot \exp(-\epsilon \cdot \Delta/2), \qquad (3)$$

where $\text{opt}(\mathbf{x}) = \mathbf{max} y \in Y(q(\mathbf{x}, y))$.

Notation. For two n-entry vectors $\mathbf{x}, \mathbf{x}'$, we write $\mathbf{x} \sim_i \mathbf{x}'$ to denote that they agree on all but the i-th entry. We write $\mathbf{x} \sim \mathbf{x}'$ if $\mathbf{x} \sim_i \mathbf{x}'$ for some i.

Key Results

The work of McSherry and Talwar [8] was first to realize a connection between differential privacy and mechanism design. They observed that (with bounded utility from the outcome) a mechanism that preserves ϵ-differential privacy is also ϵ-truthful, yielding ϵ-truthful mechanisms for approximately maximizing social welfare or revenue. Other works in this vein – using differential privacy but without incorporating the effect of privacy loss directly into the agent's utility function – include [6, 10, 12].

Privacy-Aware Mechanism Design

The mechanisms of this section share the following setup assumptions:

Optimization problem. $q : X^n \times Y \to [0, n]$ and a utility function $U : X \times Y \to [0, 1]$.

Input. n players each having an input $x_i \in X$ and a privacy valuation p_i. The players may misreport x_i.

Output. The mechanism outputs an element $y \in Y$ approximately maximizing $q(\mathbf{x}, y)$.

Utility. Each player obtains utility $U(x_i, y) - \lambda_i$ where the assumptions on how the privacy loss λ_i behaves vary for the different mechanisms below and are detailed in their respective sections.

Accuracy. Let $\text{opt}(\mathbf{x}) = \mathbf{max}_{y\in Y}(q(\mathbf{x}, y))$. A mechanism is (Δ, δ)-accurate for all $\mathbf{x}$ if it chooses $y \in Y$ such that $\Pr[\text{opt}(\mathbf{x}) - q(\mathbf{x}, y) \leq \Delta] \geq 1 - \delta$ where the probability is taken over the random coins of the mechanism. (One can also define accuracy in terms of $\text{opt}(\mathbf{x}) - \text{Exp}[q(\mathbf{x}, y)]$.)

Worst-Case Privacy Model

In the worst-case privacy model, the privacy loss of mechanism M is only assumed to be upper bounded by the mechanisms' privacy parameter, as in the discussion following Fact 1 [9]:

$$0 \leq \lambda_i \leq p_i \cdot \epsilon \quad \text{where} \quad \epsilon = \mathbf{max}_{x'\sim x, y\in Y} \ln \frac{\Pr[M(x) = y]}{\Pr[M(x') = y]}. \qquad (4)$$

Nissim, Orlandi, and Smorodinsky [9] give a generic construction of privacy-aware mechanisms assuming an upper bound on the privacy loss as in Eq. 4. The fact λ_i is only upper bounded and excludes the possibility of punishing misreporting via privacy loss (compare with Algorithms 3 and 4 below), and hence, the generic construction resorts to a somewhat nonstandard modeling from [10]. To illustrate the main components of the construction, we present a specific instantiation in the context of pricing a digital good, where such a nonstandard modeling is not needed.

Pricing a digital good. An auctioneer selling digital good wishes to design a single price mechanism that would (approximately) optimize her revenue. Every agent i has a valuation $x_i \in X = \{0, 0.01, 0.02, \ldots, 1\}$ for the good and privacy preference p_i. Agents are asked to declare x_i to the mechanism, which chooses a price $y \in Y = \{0.01, 0.02, \ldots, 1\}$. Let x'_i be the report of agent i. If $x'_i < y$, then agent i does not pay nor receives the good and hence gains zero utility, i.e., $v_i = 0$. If $x'_i \geq y$, then agent i gets the good and pays y and hence gains in utility. We let this gain be $v_i = x'_i - y + 0.005$, where the additional 0.005 can be viewed as modeling a preference to receive the good (technically, this breaks the tie between the cases $x'_i = y$ and $x'_i = y - 1$). To summarize,

$$v(x_i, x'_i, y) = \begin{cases} x_i - y + 0.005 & \text{if } y < x'_i \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 1 (ApxOptRev)

Auxiliary input: privacy parameter ϵ, probability $0 < \eta < 1$.
Input: $\mathbf{x}' = (x'_1, \ldots, x'_n) \in X^n$.

ApxOptRev executes $\boldsymbol{M}_1$ with probability $1 - \eta$ and $\boldsymbol{M}_2$ otherwise, where $\boldsymbol{M}_1, \boldsymbol{M}_2$ are:

$\boldsymbol{M}_1$: Choose $y \in Y$ using the exponential mechanism, $\boldsymbol{M}_{\text{Exp}}$ (Fact 3), i.e.,

$$\Pr[y = t] = \frac{\exp\left(\frac{\epsilon}{2} \cdot t \cdot \left|\{i : x'_i \geq t\}\right|\right)}{\sum_{\ell \in Y} \exp\left(\frac{\epsilon}{2} \cdot \ell \cdot \left|\{i : x'_i \geq \ell\}\right|\right)}.$$

$\boldsymbol{M}_2$: Choose $y \in Y$ uniformly at random.

The privacy loss for agent i is from the information that may be potentially leaked on x'_i via the chosen price y. The auctioneer's optimal revenue is $\text{opt}(\mathbf{x}) = \mathbf{max}_{t \in Y}(t \cdot |\{i : x_i \geq t\}|)$, and the revenue she obtains when the mechanism chooses price y is $y \cdot |\{i : x'_i \geq y\}|$. The mechanism is presented in Algorithm 1.

Agent utility. To analyze agent behavior, compare the utility of a misreporting agent to a truthful agent. (i) As Algorithm 1 is ϵ-differentially private, by our assumption on λ_i, by misreporting agent i may reduce her disutility due to information leakage by *at most* $p_i \cdot \epsilon$. (ii) Note that $v(x_i, x'_i, y) \leq v(x_i, x_i, y)$. Using this and Fact 1, we can bound the expected gain due to misreporting in M_1 as follows:

$$\begin{aligned} &\text{Exp}_{y \sim \boldsymbol{M}_1(\mathbf{x}'_{-i}, x'_i)}[v(x_i, x'_i, y)] - \text{Exp}_{y \sim \boldsymbol{M}_1(\mathbf{x}'_{-i}, x_i)}[v(x_i, x_i, y)] \leq \\ &\text{Exp}_{y \sim \boldsymbol{M}_1(\mathbf{x}'_{-i}, x'_i)}[v(x_i, x'_i, y)] - \text{Exp}_{y \sim \boldsymbol{M}_1(\mathbf{x}'_{-i}, x_i)}[v(x_i, x'_i, y)] \leq 2 \cdot \epsilon. \end{aligned}$$

(iii) On the other hand, in M_2, agent i loses at least $g = 0.01 \cdot 0.005$ in utility whenever $x'_i \neq x'_i$; this is because y falls in the set $\{x_i + 0.01, \ldots, x'_i\}$ with probability $x'_i - x_i \geq 0.01$ when $x_i < x'_i$, in which case she loses at least 0.005 in utility and, similarly, y falls in the set $\{x'_i, \ldots, x_i - 0.01\}$ with probability $x_i - x'_i \geq 0.01$ when $x_i < x'_i$, in which case she loses at least 0.005 in utility.

We hence get that agent i strictly prefers to report truthfully when

$$2 \cdot \epsilon - \eta \cdot g + p_i \cdot \epsilon < 0. \tag{5}$$

Designer utility. Let m be the number of agents for which Eq. 5 does not hold. We have $\text{opt}(\mathbf{x}') \geq \text{opt}(\mathbf{x}) - m$, and hence, using Fact 3, we get that

$$\Pr_{y\sim \text{ApxOptRev}(\mathbf{x}')}\left[y\cdot\left|\{i:x_i'\geq y\}\right|<\text{opt}(\mathbf{x})-m-\Delta\right]\leq |Y|\cdot\exp(-\epsilon\Delta/2)+\eta$$
$$=100\cdot\exp(-\epsilon\Delta/2)+\eta.$$

We omit from this short summary the discussion of how to choose the parameters ϵ and η (this choice directly affects m). One possibility is to assume the p_i has nice properties [9].

Per-Outcome Privacy Model

In the output specific model, the privacy loss of mechanism M is evaluated on a per-output basis [2]. Specifically, on output $y \in Y$ is assumed that

$$|\lambda_i(\mathbf{x},y)|\leq p_i\cdot F_i(\mathbf{x},y) \text{ where } F_i(\mathbf{x},y)$$
$$=\mathbf{max}_{\mathbf{x}',\mathbf{x}''\sim_i\mathbf{x}}\ln\frac{\Pr[M(\mathbf{x}')=y]}{\Pr[M(\mathbf{x}'')=y]}. \quad (6)$$

To interpret Eq. 6, consider an Bayesian adversary that has a prior belief μ on x_i and fix $\mathbf{x}_{-i}$. After seeing $y = M(\mathbf{x}_{-i}, x_i)$, the Bayesian adversary updates her belief to μ'. For every event E defined over x_i, we get that

$$\mu'(E)=\mu(E|M(\mathbf{x}_{-i},x_i)=y)$$
$$=\mu(E)\cdot\frac{\Pr[M(\mathbf{x}_{-i},x_i)=y|E]}{\Pr[M(\mathbf{x}_{-i},x_i)=y]}$$
$$\in\mu'(E)\cdot e^{\pm F_i(\mathbf{x},y)}.$$

This suggests that λ_i models harm that is "continuous" in the change in adversarial belief about i, in the sense that a small adversarial change in belief entails small harm. (Note, however, that this argument is restricted to adversarial beliefs on x_i *given* $\mathbf{x}_{-i}$.)

Comparison with Worst-Case Privacy

Note that if M is ϵ-differentially private, then $F_i(\mathbf{x}, y) \leq \epsilon$ for all $\mathbf{x}, y$. Equation 6 can hence be seen as a variant of Eq. 4 where the fixed value ϵ is replaced with the output specific $F_i(\mathbf{x}, y)$. One advantage of such a per-outcome model is that the typical gain from misreporting is significantly smaller than ϵ. In fact, for all $\mathbf{x} \in X^n$ and $x_i' \in X$,

$$\left|\text{Exp}_{y\sim\mathbf{M}(\mathbf{x})}[F_i(\mathbf{x},y)]-\text{Exp}_{y\sim\mathbf{M}(\mathbf{x}_{-i},x_i')}[F_i(\mathbf{x},y)]\right|=O(\epsilon^2).$$

On the other hand, the modeled harm is somewhat weaker, as (by Fact 1) Eq. 4 also captures harm that is not continuous in beliefs (such as decisions based on the belief crossing a certain threshold).

Assuming privacy loss is bounded as in Eq. 6, Chen, Chong, Kash, Moran, and Vadhan [2] construct truthful mechanisms for an election between two candidates, facility location, and a VCG mechanism for public projects (the latter uses payments). Central to the constructions is the observation that F_i is large exactly when agent i has influence on the outcome of $M()$. To illustrate the main ideas in the construction, we present here the two-candidate election mechanism.

Two-candidate election. Consider the setting of an election between two candidates. Every agent i has a preference $x_i \in X = \{A, B\}$ and privacy preference p_i. Agents are asked to declare x_i to the mechanism, which chooses an outcome $y \in Y = \{A, B\}$. The utility of agent i is then

$$v(x_i,y)=\begin{cases}1 \text{ if } x=y\\ 0 \text{ otherwise}\end{cases}$$

The privacy loss for agent i is from the information that may be potentially leaked on her reported x_i' via the outcome y. The designer's goal is to (approximately) maximize the agents' social

Algorithm 2 ApxMaj

Auxiliary input: privacy parameter ϵ.
Input: $\mathbf{x}' = (x'_1, \ldots, x'_n) \in X^n$.

ApxMaj performs the following:

1. Sample a value Z from $\text{Lap}(1/\epsilon)$.
2. Choose $y = A$ if $|\{j : x'_j = A\}| > |\{j : x'_j = B\}| + Z$ and $y = B$ otherwise.

welfare (i.e., total utility from the outcome). The mechanism is presented in Algorithm 2.

Agent utility. To analyze agent behavior, we compare the utility of a misreporting agent to a truthful agent. Notice that once the noise Z is fixed if agent i affects the outcome, then her disutility from information leakage is at most $p_i \cdot \epsilon$ and her utility from the outcome decreases by 1. If agent i cannot affect the outcome, then misreporting does not change either. We hence get that agent i strictly prefers to report truthfully when

$$p_i \cdot \epsilon < 1. \tag{7}$$

Note that by our analysis, Eq. 7 implies *universal truthfulness* – agent i prefers to report truthfully for every choice of the noise Z. In contrast, Eq. 5 only implies truthfulness in expectation.

Social welfare. Letting m be the number of agents for which Eq. 7 does not hold, and using Fact 2, we get that Algorithm ApxMaj maximizes social welfare up to error $m + \frac{\log 1/\delta}{\epsilon}$ with probability $1 - \delta$. As in the previous section, we omit from this short summary the discussion of how to choose ϵ (this choice affects m and hence the accuracy of the mechanism).

Purchasing Privacy

The mechanisms of this section share the following setup assumptions, unless noted otherwise:

Input. n players each having a data bit $x_i \in \{0, 1\}$ and a privacy valuation $p_i > 0$. The players may misreport p_i but cannot misreport x_i. We will assume for convenience of notation that $p_1 \leq p_2 \leq \ldots \leq p_n$.

Intermediate outputs. The mechanism selects a subset of participating players $S \subset [n]$ and a scaling factor t and a privacy parameter ϵ.

Output. The mechanism uses the Laplace mechanism to output an estimate $s = \frac{1}{t}\left(\sum_{i \in [S]} x_i + Z\right)$ where $Z \sim \text{Lap}(1/\epsilon)$ and payments v_i for $i \in [n]$.

Utility. Each player obtains utility $v_i - \lambda_i$ where the assumptions on how the privacy loss λ_i behaves vary for the different mechanisms below and are detailed in their respective sections.

Accuracy. A mechanism is α-accurate if $\Pr[|s - f(x)| \leq \alpha n] \geq 2/3$ where the probability is taken over the random coins of the mechanism.

We focus on designing mechanisms that approximate the sum function $f(x) = \sum_{i=1}^{n} x_i$ where each $x_i \in \{0, 1\}$, which has been the most widely studied function in this area. As one can see from the above setup assumptions, the crux of the mechanism design problem is in selecting the set S, choosing a privacy parameter ϵ, and computing payments for the players. We note that several of the works we describe below generalize beyond the setting we describe here (i.e., computing different f, fewer assumptions, etc.). The following presentation was designed to give a unified overview (sacrificing some generality), but to preserve the essence both of the challenges posed by the problem of purchasing private data and each mechanism's idea in addressing the challenges.

Insensitive Valuation Model

In the insensitive valuation model, the privacy loss λ_i of a mechanism M is assumed to be [5]

$$\lambda_i = p_i \cdot \epsilon_i \text{ where } \epsilon_i = \mathbf{max}_{x, x' \sim_i x, p, s} \ln \frac{\Pr[M(x, p) = s]}{\Pr[M(x', p) = s]}. \tag{8}$$

It is named the insensitive valuation model because ϵ_i only measures the effect on privacy of changing player i's data bit, but not the effect of changing that player's privacy valuation.

Algorithm 3 (FairQuery)

Auxiliary input: budget constraint $B > 0$.

1. Let $k \in [n]$ be the largest integer such that $p_k(n-k) \leq B/k$.
2. Select $S = \{1, \ldots, k\}$ and set $\epsilon = \frac{1}{n-k}$. Set the scaling factor $t = 1$.
3. Set payments $v_i = 0$ for all $i > k$ and $v_i = \min\{\frac{B}{k}, p_{k+1}\epsilon\}$ for all $i \leq k$.

Algorithm 4 MinCostAuction

Auxiliary input: accuracy parameter $\alpha \in (0, 1)$.

1. Set $\alpha' = \frac{\alpha}{1/2+\ln 3}$ and $k = (1-\alpha')n$.
2. Select $S = \{1, \ldots, k\}$ and $\epsilon = \frac{1}{n-k}$. Set the scaling factor $t = 1$.
3. Set payments $v_i = 0$ for $i > k$ and $v_i = p_{k+1}\epsilon$ for all $i \leq k$.

Mechanisms. Two mechanisms are presented in the insensitive value model in [5], listed in Algorithms 3 and 4. Algorithm 3 (FairQuery) is given a hard budget constraint and seeks to optimize accuracy under this constraint; Algorithm 4 (MinCostAuction) is given a target accuracy requirement and seeks to minimize payouts under these constraints.

Guarantees. Algorithms 3 and 4 are individually rational and truthful. Furthermore, Algorithm 3 achieves the best possible accuracy (up to constant factors) for the class of *envy-free* and individually rational mechanisms, where the sum of payments to players does not exceed B. Algorithm 4 achieves the minimal payout (up to constant factors) for the class of *envy-free* and individually rational mechanisms that achieve α-accuracy.

Sensitive Value Model

Ghosh and Roth [5] also defined the sensitive value model where λ_1 is as in Eq. 8, except that ϵ_i is defined to equal

$$\mathbf{max}_{x,p,(x',p')\sim_i(x,p),s} \ln \frac{\Pr[M(x,p)=s]}{\Pr[M(x',p')=s]}. \quad (9)$$

Namely, we also measure the effect on the outcome distribution of the change in a single player's privacy valuation. It was shown in [5] and subsequent generalizations [11] that in this model and various generalizations where the privacy valuation itself is sensitive, it is impossible to build truthful, individually rational, and accurate mechanisms with worst-case guarantees and making finite payments. To bypass these impossibility results, several relaxations were introduced.

Bayesian relaxation [4]. Fleischer and Lyu use the sensitive notion of privacy loss given in Eq. 9. In order to bypass the impossibility results about sensitive values, they assume that the mechanism designer has knowledge of prior distributions P^0, P^1 for the privacy valuations. They assume that all players with data bit b have privacy valuation sampled independently according to P^b, namely, that $p_i \overset{R}{\leftarrow} P^{x_i}$, independently for all i. Their mechanism is given in Algorithm 5.

Algorithm 5 Bayesian mechanism from [4]

Auxiliary input: privacy parameter ϵ.

1. Compute $c = 1 - \frac{2}{\epsilon^2 n}$. Compute α_b for $b \in \{0, 1\}$ such that $\Pr_{p \overset{R}{\leftarrow} P^b}[p \leq \alpha_b] = c$.
2. Set S be the set of players i such that $p_i \leq \alpha_{x_i}$. Set the scaling factor $t = c$.
3. For each player $i \in S$, pay $\epsilon\alpha_{x_i}$. Pay the other players 0.

Algorithm 5 is truthful and individually rational. Assuming that the prior beliefs are correct, the mechanism is $O(\frac{1}{\epsilon n})$-accurate. The key use of knowledge of the priors is in accuracy: the probability of a player participating is c independent of its data bit.

Take-it-or-leave-it mechanisms [7]. Ligett and Roth put forward a setting where the privacy loss is decomposed into two parts

$$\lambda_i = \lambda_i^p + \lambda_i^x,$$

where λ_i^p is the privacy loss incurred by leaking information of whether or not an individual is selected to participate (i.e., whether individual i is in the set S), and where λ_i^x is the privacy loss incurred by leaking information about the actual data bit.

The interpretation is that a surveyor approaches an individual and offers them v_i to participate. The individual cannot avoid responding to this question and so unavoidably incurs a privacy loss λ_i^p without compensation. If he chooses to participate, then he loses an additional λ_i^x, but in this case he receives v_i in payment. While this is the framework we have been working in all along, up until now we have not distinguished between these two sources of privacy loss, rather considering only the overall loss. By explicitly separating them, [7] can make more precise statements about how incentives relate to each source of privacy loss.

In this model the participation decision of an individual is a function (only) of its privacy valuation, and so we define

$$\begin{aligned}\lambda_i^p &= p_i \epsilon_i^p \text{ where } \epsilon_i^p \\ &= \mathbf{max}_{x,p,p' \sim_i p,s} \ln \frac{\Pr[M(x,p)=s]}{\Pr[M(x,p')=s]}.\end{aligned} \tag{10}$$

We define $\lambda_i^x = p_i \epsilon_i^x$ where ϵ_i^x is as in the insensitive model, Eq. 8. The mechanism is given in Algorithm 6.

Algorithm 6 is α-accurate. It is not individually rational since players cannot avoid the take-it-or-leave-it offer, which leaks information about their privacy valuation that is not compensated. However, it is "one-sided truthful" in the sense that rational players will accept any offer v_i satisfying $v_i \geq \lambda_i^p - \lambda_i^x$. [7] also proves that for appropriately chosen η, the total payments made by Algorithm 6 are not much more than that of the optimal envy-free mechanism making the same take-it-or-leave-it offers to every player.

Monotonic valuations [11]. Nissim, Vadhan, and Xiao [11] study a relaxation of sensitive values that they call *monotonic valuations*, where it is assumed that

$$\begin{aligned}\lambda_i(x,p) &\leq p_i \cdot \epsilon_i^{\mathsf{mon}}(x,p) \text{ where } \epsilon_i^{\mathsf{mon}}(x,p) \\ &= \mathbf{max}_{(x',p') \sim_i^{\mathsf{mon}}(x,p),s} \\ &\quad \ln \frac{\Pr[M(x,p)=s]}{\Pr[M(x',p')=s]}.\end{aligned} \tag{11}$$

Here, $(x',p') \sim_i^{\mathsf{mon}} (x,p)$ denotes that $(x',p'),(x,p)$ are identical in all entries except the i'th entry, and in the i'th entry, it holds that either $x_i > x_i'$ and $p_i \geq p_i'$ both hold or $x_i < x_i'$ and $p_i \leq p_i'$ both hold.

The intuition behind the definition is that for many natural settings, $x_i = 1$ is more sensitive than $x_i = 0$ (e.g., if x_i represents whether an individual tested positive for syphilis), and it is therefore reasonable to restrict attention to the case where the privacy valuation when $x_i = 1$ is at least the privacy valuation when $x_i = 0$.

There are two other aspects in which this notion is unlike those used in the earlier works on purchasing privacy: (i) the definition may depend on the input, so the privacy loss may be smaller on some inputs than others, and (ii) we assume

Algorithm 6 (Take-it-or-leave-it mechanism [7])

Auxiliary input: accuracy parameter $\alpha \in (0,1)$, payment increment $\eta > 0$.

1. Set $j = 1$ and $\epsilon = \alpha$.
2. Repeat the following:
   (a) Set $E_j = 100(\log j + 1)/\alpha^2$ and $S_j = \emptyset$.
   (b) For $i = 1$ until E_j:
      i. Sample without replacement $i \xleftarrow{R} [n]$.
      ii. Offer player i a payment of $(1+\eta)^j$.
      iii. If player i accepts, set $S_j = S_j \cup \{i\}$.
   (c) Sample $\nu \xleftarrow{R} \Lambda(1/\epsilon)$. If $|S_j| + \nu \geq (1 - \alpha/8)E_j$, then break and output selected set $S = S_j$, privacy parameter ϵ, and normalizing factor $t = E_j$. For every $j' \leq j$, pay $(1+\eta)^{j'}$ to each player that accepted in round j' and pay 0 to all other players.
   (d) Otherwise, increment j and continue.

only an upper bound on the privacy loss, since ϵ_i^{mon} does not say *which* information is leaked about player i, and so it may be that the harm done to player i is not as severe as ϵ_i^{mon} would suggest. The mechanism is given in Algorithm 7.

Algorithm 7 (Mechanism for monotonic valuations [11])

Auxiliary inputs: budget constraint $B > 0$, privacy parameter $\epsilon > 0$.

1. Set $\tau = \frac{B}{2\epsilon n}$.
2. Output selected set $S = \{i \mid p_i \leq \tau\}$, output privacy parameter ϵ, and scaling factor $t = 1$.
3. Pay B/n to players in S, pay 0 to others.

Algorithm 7 is individually rational for all players and truthful for all players satisfying $p_i \leq \tau$. Assuming all players are rational, on inputs where there are h players having $p_i > \tau$, the mechanism is $(O(\frac{1}{\epsilon n}) + h)$-accurate. The accuracy guarantee holds regardless of how the players with $p_i > \tau$ report their privacy valuations.

Recommended Reading

1. Alpert CJ, Chan T, Kahng AB, Markov IL, Mulet P (1998) Faster minimization of linear wirelength for global placement. IEEE Trans CAD 17(1):3–13
2. Chen Y, Chong S, Kash IA, Moran T, Vadhan SP (2013) Truthful mechanisms for agents that value privacy. In: EC 2013, Philadelphia, pp 215–232
3. Dwork C, McSherry F, Nissim K, Smith A (2006) Calibrating noise to sensitivity in private data analysis. In: TCC 2006, New York, pp 265–284
4. Fleischer L, Lyu Y-H (2012) Approximately optimal auctions for selling privacy when costs are correlated with data. In: EC 2012, Valencia, pp 568–585
5. Ghosh A, Roth A (2011) Selling privacy at auction. In: EC 2011, San Jose, pp 199–208
6. Huang Z, Kannan S (2012) The exponential mechanism for social welfare: private, truthful, and nearly optimal. In: FOCS 2012, New Brunswick, pp 140–149
7. Ligett K, Roth A (2012) Take it or leave it: running a survey when privacy comes at a cost. In: WINE 2012, Liverpool, pp 378–391
8. McSherry F, Talwar K (2007) Mechanism design via differential privacy. In: FOCS 2007, Providence, pp 94–103
9. Nissim K, Orlandi C, Smorodinsky R (2012) Privacy-aware mechanism design. In: EC 2012, Valencia, pp 774–789
10. Nissim K, Smorodinsky R, Tennenholtz M (2012) Approximately optimal mechanism design via differential privacy. In: ITCS 2012, Boston, pp 203–213
11. Nissim K, Vadhan SP, Xiao D (2014) Redrawing the boundaries on purchasing data from privacy-sensitive individuals. In: ITCS 2014, Princeton, pp 411–422
12. Xiao D (2013) Is privacy compatible with truthfulness? In: ITCS 2013, Berkeley, pp 67–86

Mellor-Crummey and Scott Mutual Exclusion Algorithm

Danny Hendler
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Keywords

Critical section; Local spinning; MCS lock; Queue lock; Remote memory references (RMRs)

Years and Authors of Summarized Original Work

1991; Mellor-Crummey, Scott

Problem Definition

Mutual exclusion is a fundamental concurrent programming problem (see ▸ Concurrent Programming, Mutual Exclusion entry), in which a set of processes must coordinate their access to a *critical section* so that, at any point in time, at most a single process is in the critical section.

To a large extent, shared-memory mutual exclusion research focused on *busy-waiting* mutual exclusion, in which, while waiting for the critical section to be freed, processes repeatedly test the values of shared-memory variables. A significant portion of this research over the last two decades was devoted to *local-spin algorithms* [2], in which all busy-waiting is done by means

of read-only loops that repeatedly test locally accessible variables.

Local-Spin Algorithms and the RMRs Metric

A natural way to measure the time complexity of algorithms in shared-memory multiprocessors is to count the number of memory accesses they require. This measure is problematic for busy-waiting algorithms because, in this case, a process may perform an unbounded number of memory accesses while busy-waiting for another process holding the lock. Moreover, Alur and Taubenfeld [1] have shown that even the first process to enter the critical section can be made to perform an unbounded number of accesses.

As observed by Anderson, Kim, and Herman [6], "most early shared-memory algorithms employ... busy-waiting loops in which many shared variables are read and written... Under contention, such busy-waiting loops generate excessive traffic on the processors-to-memory interconnection network, resulting in poor performance."

Contemporary shared-memory mutual exclusion research focuses on *local-spin* algorithms, which avoid this problem as they busy-wait by means of performing read-only loops that repeatedly test locally accessible variables (see, e.g., [4, 5, 7, 9, 11, 13, 14]). The performance of these algorithms is measured using the remote memory references (RMRs) metric.

The classification of memory accesses into local and remote depends on the type of multiprocessor. In the *distributed shared-memory* (DSM) model, each shared variable is local to exactly one processor and remote to all others. In the *cache-coherent* (CC) model, each processor maintains local copies of shared variables inside a cache; the consistency of copies in different caches is ensured by a coherence protocol. At any given time, a variable is local to a processor if the coherence protocol guarantees that the corresponding cache contains an up-to-date copy of the variable and is remote otherwise.

Anderson was the first to present a local-spin mutual exclusion algorithm using only reads and writes with bounded RMR complexity [3]. In his algorithm, a process incurs $O(n)$ RMRs to enter and exit its critical section, where n is the maximum number of processes participating in the algorithm. Yang and Anderson improved on that and presented an $O(\log n)$ RMRs mutual exclusion algorithm based on reads and writes [20]. This is asymptotically optimal under both the CC and DSM models [7].

Read-Modify-Write Operations

The system's hardware or operating system provides *primitive operations* (or simply *operations*) that can be applied to shared variables. The simplest operations, which are always assumed, are the familiar *read* and *write* operations. Modern architectures provide stronger *read-modify-write* operations (a.k.a. *fetch-and-*Φ operations). The most notable of these is *compare and swap* (abbreviated CAS), which takes three arguments: an address of a shared variable, an expected value, and a new value. The CAS operation atomically does the following: if the variable stores the expected value, it is replaced with the new value; otherwise, it is unchanged. The success or failure of the CAS operation is then reported back to the program. It is crucial that this operation is executed atomically; thus, an algorithm can read a datum from memory, modify it, and write it back only if no other process modified it in the meantime.

Another widely implemented RMW operation is the *swap* operation, which takes two arguments: an address of a shared variable and a new value. When applied, it atomically stores the new value to the shared variable and returns the previous value. The CAS operation may be viewed as a conditional version of swap, since it performs a swap operation only if the value of the variable to which it is applied is the expected value.

Architectures supporting strong RMW operations admit implementations of mutual exclusion that are more efficient in terms of their RMR complexity as compared with architectures that support only read and write operations. In work that preceded the introduction of the MCS lock, Anderson [2] and Graunke and Thakkar [12] presented lock algorithms, using strong RMW

operations such as CAS and swap, that incur only a constant number of RMRs on CC multiprocessors. However, these algorithms are not local spin on DSM multiprocessors and the amount of pre-allocated memory per lock is linear in the number of processes that may use it.

Key Results

Mellor-Crummey and Scott's algorithm [18] is the first local-spin mutual exclusion algorithm in which processes incur only a constant number of RMRs to enter and exit the critical section, in both CC and DSM multiprocessors. The amount of memory that needs to be pre-allocated by locks using this algorithm (often called *MCS locks*) is constant rather than a function of the maximum number of processes that may use the lock. Moreover, MCS locks guarantee a strong notion of fairness called *first-in, first-out* (FIFO, a.k.a. *first-come-first-served*). Informally, FIFO ensures that processes succeed in capturing the lock in the order in which they start waiting for it (see [15] for a more formal definition of the FIFO property).

Algorithm 1 Mellor-Crummey and Scott algorithm

```
1  Qnode: structure {bit locked, Qnode* next};
2  shared Qnode nodes[0 . . . n − 1], Qnode* tail
   ch605:initially null;
3  local Qnode* myNode initially &nodes[i],
   successor, pred;
   Entry code for process i
5  myNode.next ← null;
6  pred ← swap(tail, myNode);
7  if pred ≠ null then
8  |  myNode.locked ← true;
9  |  pred.next ← myNode;
10 |  repeat while myNode.locked = true;
   end
   Critical Section;
   Exit code for process i;
14 if myNode.next = null then
15 |  if compare-and-swap(tail, myNode, null) =
   |  false then
16 |  |  repeat while myNode.next = null ;
17 |  |  successor ← myNode.next;
18 |  |  succcessor.locked ← false ;
   |  end
   else
21 |  successor ← myNode.next;
22 |  succcessor.locked ← false;
   end
```

The Algorithm

Pseudocode of the algorithm is presented in Algorithm 1. The key data structure used by the algorithm is the *nodes* array (statement 2), where entry i is owned by process i, for $i \in \{0, \ldots, n-1\}$. This array represents a queue of processes waiting to enter the critical section. Each array entry is a Qnode structure (statement 1), comprising a *next* pointer to the structure of the next process in the queue and a *locked* flag on which a process waits until it is signaled by its predecessor. Shared variable *tail* points to the end of the queue and either stores a pointer to the structure of the last process in the queue or is null if the queue is empty.

Before entering the critical section, a process p first initializes the *next* pointer of its Qnode structure to null (statement 5), indicating that it is about to become the last process in the queue. It then becomes the last queue process by atomically swapping the values of *tail* and its local Qnode structure (statement 6); the previous value of *tail* is stored to local variable *pred*. If the queue was previously empty (statement 7), p enters the critical section immediately. Otherwise, p initializes its Qnode structure (statement 8), writes a pointer to its Qnode structure to the *next* field of its predecessor's Qnode structure (statement 9), and then busy-waits until it is signaled by its predecessor (statement 10).

To exit the critical section, process p first checks whether its queue successor (if any) has set the *next* pointer of its Qnode structure (statement 14), in which case p signals its successor to enter the critical section (statements 21–22). Even if no process has set p's *next* pointer yet, it is still possible that p does have a successor q that executed the swap of statement 6 but did not yet update p's *next* pointer in statement 9. Also in this case, p must signal q before it is allowed to exit the critical section. To determine whether or not this is the case, p attempts to perform a

CAS operation that will swap the value of *tail* back to null if p is the single queue process. If the CAS fails, then p does have a successor and must therefore wait until its *next* pointer is updated by the successor. Once this happens, p signals its successor and exits (statements **16**–**18**).

Mellor-Crummey and Scott's paper won the 2006 Edsger W. Dijkstra Prize in Distributed Computing. Quoting from the prize announcement, the MCS lock is "...probably the most influential practical mutual exclusion algorithm of all time."

Cross-References

- Concurrent Programming, Mutual Exclusion
- Transactional Memory
- Wait-Free Synchronization

Further Reading

For a comprehensive discussion of local-spin mutual exclusion algorithms, the reader is referred to the excellent survey by Anderson, Kim, and Herman [6]. Craig, Landin, and Hagersten [8, 17] presented another queue lock – the CLH lock. The algorithm underlying CLH locks is simpler than the MCS algorithm and, unlike MCS, only requires the swap strong synchronization operation. On the downside, CLH locks are not local spin on DSM multiprocessors.

In many contemporary multiprocessor architectures, processors are organized in clusters and intercluster communication is much slower than intra-cluster communication. *Hierarchical locks* take into account architecture-dependent considerations such as inter- and intra-cluster latencies and may thus improve lock performance on nonuniform memory architectures (NUMA). The idea underlying hierarchical locks is that intercluster lock transfers should be favored. A key challenge faced by hierarchical lock implementations is that of ensuring fairness.

Radovic and Hagersten presented the first hierarchical lock algorithms [19], based on the idea that a process busy-waiting for a lock should back off for a short duration if the lock is held by a process from its own cluster and for a much longer duration otherwise; this is a simple way of ensuring that intra-cluster lock transfers become more likely. Several works pursued this line of research by presenting alternative NUMA-aware lock algorithms (e.g., [10, 16]). For alternatives to lock-based concurrent programming, the reader is referred to [Wait-free Synchronization, Transactional Memory].

Recommended Reading

1. Alur R, Taubenfeld G (1992) Results about fast mutual exclusion. In: Proceedings of the 13th IEEE real-time systems symposium, Phoenix, pp 12–21
2. Anderson TE (1990) The performance of spin lock alternatives for shared-memory multiprocessors. IEEE Trans Parallel Distrib Syst 1(1):6–16
3. Anderson JH (1993) A fine-grained solution to the mutual exclusion problem. Acta Inf 30(3):249–265
4. Anderson J, Kim YJ (2002) An improved lower bound for the time complexity of mutual exclusion. Distrib Comput 15(4):221–253
5. Kim Y-J, Anderson JH (2006) Nonatomic mutual exclusion with local spinning. Distrib Comput 19(1):19–61
6. Anderson J, Kim YJ, Herman T (2003) Shared-memory mutual exclusion: major research trends since 1986. Distrib Comput 16:75–110
7. Attiya H, Hendler D, Woelfel P (2008) Tight RMR lower bounds for mutual exclusion and other problems. In: STOC, Victoria, pp 217–226
8. Craig T (1993) Building FIFO and priority-queuing spin locks from atomic swap. Technical report
9. Danek R, Golab WM (2010) Closing the complexity gap between FCFS mutual exclusion and mutual exclusion. Distrib Comput 23(2):87–111
10. Dice D, Marathe VJ, Shavit N (2012) Lock cohorting: a general technique for designing NUMA locks. In: PPOPP, New Orleans, pp 247–256
11. Golab WM, Hendler D, Woelfel P (2010) An $O(1)$ RMRS leader election algorithm. SIAM J Comput 39(7):2726–2760
12. Graunke G, Thakkar SS (1990) Synchronization algorithms for shared-memory multiprocessors. IEEE Comput 23(6):60–69
13. Kim YJ, Anderson JH (2007) Adaptive mutual exclusion with local spinning. Distrib Comput 19(3):197–236
14. Kim YJ, Anderson JH (2012) A time complexity lower bound for adaptive mutual exclusion. Distrib Comput 24(6):271–297
15. Lamport L (1974) A new solution of Dijkstra's concurrent programming problem. Commun ACM 17(8):453–455

16. Luchangco V, Nussbaum D, Shavit N (2006) A hierarchical CLH queue lock. In: Euro-Par, Dresden, pp 801–810
17. Magnusson P, Landin A, Hagersten E (1994) Queue locks on cache coherent multiprocessors. In: IPPS, Cancún. IEEE Computer Society, pp 165–171
18. Mellor-Crummey JM, Scott ML (1991) Algorithms for scalable synchronization on shared-memory multiprocessors. ACM Trans Comput Syst 9(1):21–65. doi:10.1145/103727.103729, http://doi.acm.org/10.1145/103727.103729
19. Radovic Z, Hagersten E (2003) Hierarchical backoff locks for nonuniform communication architectures. In: HPCA, Anaheim, pp 241–252
20. Yang JH, Anderson J (1995) A fast, scalable mutual exclusion algorithm. Distrib Comput 9(1):51–60

Memory-Constrained Algorithms

Matias Korman
Graduate School of Information Sciences, Tohoku University, Miyagi, Japan

Keywords

Connectivity; Constant workspace; Logspace; Multi-pass algorithms; One-pass algorithms; Selection; Sorting; Stack algorithms; Streaming; Time-space trade-off; Undirected graphs

Years and Authors of Summarized Original Work

1980; Munro, Patterson
2000; Reingold
2013; Barba, Korman, Langerman, Sadakane, Silveira

Problem Definition

This field of research evolves around the design of algorithms in the presence of memory constraints. Research on this topic has been going on for over 40 years [16]. Initially, this was motivated by the high cost of memory space. Afterward, the topic received a renewed interest with appearance of smartphones and other types of handheld devices for which large amounts of memory are either expensive or not desirable.

Although many variations of this principle exist, the general idea is the same: the input is in some kind of read-only data structure, the output must be given in a write-only structure, and in addition to these two structures, we can only use a fixed amount of memory to compute the solution. This memory should be enough to cover all space requirements of the algorithm (including the variables directly used by the algorithm, space needed to make recursion, invoking procedures, *etc.*). In the following we list the most commonly considered limitations for both the input and the workspace.

Considerations on the Input

One of the most restrictive models that has been considered is the *one-pass* (or *streaming*) model. In this setting the elements of the input can only be scanned once in a sequential fashion. Given the limitations, the usual aim is to approximate the solution and ideally obtain some kind of worst-case approximation ratio with respect to the optimal solution.

The natural extension of the above constraint is the *multi-pass* model, in which the input can be scanned sequentially a constant number of times. In here we look for trade-off between the number of passes and either the size of the workspace or the quality of the approximation.

The next natural step is to allow input to be scanned any number of times and even allowing *random access* to the input values. Research for this model focuses on either computability (i.e., determining whether or not a particular problem is solvable with a workspace of fixed size) or the design of efficient algorithms whose running time is not much worse (when compared to the case in which no space constraints exist).

A more generous model considered is the *in-place* one. In this model, the values of the input can be rearranged (or sometimes even overwritten). Note that the input need not be recoverable after an execution of the algorithm. By making an appropriate permutation of the input, we can usually encode different data structures. Thus,

algorithms under this model often achieve the running times comparable to those in unconstrained settings.

Considerations on the Workspace

The most natural way to measure the space required by the algorithms is simply the number of bits used. On many cases it is simpler to count the number of *words* (i.e., the minimum amount of space needed to store a variable, a pointer to some position in the input, or simply a counter) used by the algorithm. It is widely accepted that a word needs $\Theta(\log n)$ bits; thus it is easy to alternate between both approaches.

Most of the literature focuses in the case in which the workspace can only fit $O(\log n)$ bits. This workspace (combined with random access to the input) defines the heavily studied *log-space* complexity class within computational complexity. Within this field the main focus of research is to determine whether or not a problem can be solved (without considering other properties such as the running time of the algorithm). Due to the logarithmic bit-word equivalence, an algorithm that uses $O(\log n)$ bits is also referred to as a *constant workspace* algorithm.

There has also been an interest in the design of algorithms whose workspace depends on some parameter determined by the user. In this case the aim is to obtain an algorithm whose running time decreases as the space increases (this is often referred to as a *time-space trade-off*).

Key Results

Selection and Sorting in Multi-pass Models

One of the most studied problems under the multi-pass model is *sorting*. That is, given a list of n distinct numbers, how fast can we sort them? How many passes of the input are needed when our total amount of memory is limited? Whenever workspace is not large enough to hold the sorted list, the aim is to simply report the values of the input in ascending order.

The first time-space trade-off algorithm for sorting under the multi-pass model was given by Munro and Paterson [13], where several upper and lower space bounds were given (as a function on the number of times we can scan the input). The bounds were afterward improved and extended for the random access model: it is known that the running time of an algorithm that uses $O(b)$ bits must be at least $\Omega(n^2/b)$ [8]. Matching upper bounds for the case in which $b \in \Omega(\log n) \cap O(n/\log n)$ were shown by Pagter and Rauhe [15] (where b denotes the size of the workspace, in bits).

Selection

Another closely related topic is *selection*. In addition to the list of n numbers, we are given an integer $k \leq n$. The aim is to compute the number whose rank is k (i.e., the kth smallest value). It is well-known that this problem can be solved in linear time when no space constraints exist [7]. Munro and Patterson [13] presented a time-space trade-off algorithm for the multi-pass model. For any $w \in \Omega(\log^2 n) \cap O(n/\log n)$, the algorithm runs in $O(n \log_w n + n \log w)$ time and uses $O(w)$ words of space.

The algorithm stores two values – called *filters* – for which we know that the element to select lies in between (initially, the filters are simply set to $\pm\infty$, respectively). Thus, the aim is to iteratively scan the input shrinking the distance between the filters. At each iteration we look for an element whose rank is as close as possible to k (ignoring elements that do not lie within the filters). Once we have a candidate, we can compute its exact rank in linear time by comparing its value with the other elements of the input and update either the upper or lower filter accordingly. The process is repeated until $O(w)$ elements remain between the filters.

The key of this algorithm lies in a good choice of an approximation so that a large amount of values are filtered. The method of Munro and Patterson [13] first constructs a sample of the input as follows: for a block of up to $\frac{w}{\log n}$ elements, its sample simply consists of these elements sorted in increasing value. For larger blocks B (say, $2^i \frac{w}{\log n}$ elements for some $i \in \{1, \ldots, \lceil \log(n/w) \rceil\}$), partition the block into two equally sized sub-blocks and construct their samples inductively. Then, the sample of B is created

by selecting one every other element of each of the samples of the two sub-blocks and sorting the obtained list. The sample of the whole input can be constructed in a bottom-up fashion that uses at most $O(\frac{w}{\log n})$ words in each level (thus, $O(w)$ in total). Once we have computed the sample of the input, we can extract its approximation by selecting the corresponding value within the sample.

Randomized Algorithms

The previous approach can be drastically simplified under randomized environments. Simply select an element of the input uniformly at random, compute its rank, and update one of the filters accordingly. With high probability after a constant number of iterations, a constant fraction of the input will be discarded. Thus, overall $O(\log n)$ iterations (and a constant number of words) will be needed to find the solution. Chan [9] improved this idea, obtaining an algorithm that runs in $O(n \log \log_w n)$ time and uses $O(w)$ words (for any $w \leq n$).

Improvements

For most values of w, the algorithm of Munro and Patterson is asymptotically tight, even if we allow random access to the input. Thus, further research focused in extending the range space for which optimality is known. Frederickson [11] increased the optimality range for the selection problem to any $w \in \Omega(\log n^2) \cap O(2^{\log n / \log^* n})$. Recently, Elmasry et al. [10] gave a linear time algorithm that only uses $O(n)$ bits (i.e., they preserve the linear running time of [7] and reduce the size of the workspace by a logarithmic factor). Raman and Ramnath [17] used a similar approximate median approach for the case in which $o(\log n)$ words fit in the workspace.

Undirected Graph Connectivity in the Random Access Model

Given an undirected graph $G = (V, E)$ and two vertices $s, t \in V$, the *undirected s–t connectivity problem* is to decide whether or not there exists a path from s to t in G. This problem can be easily solved in linear time using linear space (with either breadth-first search or width-first search schemes) in unrestricted models. However, determining the existence of a deterministic algorithm that only uses $O(\log n)$ bits space was a long-standing open problem in the field of complexity theory.

Problem Background

Aleliunas et al. [1] showed that the problem can be easily solved with a randomized algorithm. Essentially they show that a sufficiently long random walk will traverse all vertices of a connected graph. Thus, if we start at s and do not reach t after a polynomially bounded number of steps, we can conclude that with high probability, s and t are not connected.

The connectivity problem can also be solved with a nondeterministic logspace algorithm (where the certificate is simply the path connecting s and t). Thus, Savitch's theorem [19] can transform it to a deterministic algorithm that uses $O(\log^2 n)$ bits (and superpolynomial time). The space requirements were afterward reduced to $O(\log^{3/2} n)$ [14] and $O(\log^{4/3} n)$ [2]. Recently, Reingold [18] positively answered the question by giving a deterministic logspace algorithm. Although no discussion on the running time is explicitly mentioned, it is well known that Reingold's algorithm runs in polynomial time. This is due to the fact that a Turing machine with a logarithmic space constraint can have at most $2^{O(\log n)} = O(n^{O(1)})$ different configurations.

Reingold's Algorithm

Conceptually, the algorithm aims to transform G into a graph in which all vertices have degree three. This is done by virtually replacing each vertex of degree $k > 3$ by a cycle of k vertices each of which is adjacent to one of the neighbors of the previous graph (and adding self-loops to vertices of low degree).

The algorithm then combines the *squaring* and the *zig-zag product* operations. The squaring operation connects vertices whose distance in the original graph is at most two, while the zig-zag product between two graphs G and H essentially replaces every vertex of G with a copy of H (and connects vertices of two copies of H if and only if the original vertices were adjacent in G).

Intuitively speaking, the squaring operation will reduce the diameter while the zig-zag product keeps the degree of the vertices bounded (for this algorithm, H consists of a sparse graph of constant degree and small diameter). After repeating this process a logarithmic number of times, the resulting graph has bounded degree, logarithmic diameter and preserves the connectivity between the corresponding vertices. In particular, we can determine the connectivity between u and v by exhaustively looking through all paths of logarithmic length starting from u. Since each vertex has bounded degree, the paths can be encoded without exceeding the space bounds. The algorithm will stop as soon as v is found in any of these paths or after all paths have been explored.

Even though we cannot store the transformation of G explicitly, the only operation that is needed during the exhaustive search is to determine the ith clockwise neighbor of a vertex after j transformation steps have been done on G (for some i and $j \in O(\log n)$). Reingold provided a method to answer such queries using constant number of bits on the graph resulting after doing $j-1$ transformation steps on G. Thus, by repeating the process inductively, we can find the solution to our query without exceeding the space bounds.

Other Models of Note

The study of memory constrained algorithms has received a lot of interest by the computational geometry community. Most of them use random access to the input, use a constant number of words, and aim to reporting fundamental geometric data structures. For example, Jarvis march [12] (also known as the *gift-wrapping algorithm*) computes the convex hull of a set of n points in $O(nh)$ time (where h is the number of vertices on the convex hull). Asano and Rote [3] showed how to compute the Voronoi diagram and minimum spanning tree of a given set of points in $O(n^2)$ and $O(n^3)$ time, respectively. Similarly, several time-space trade-off algorithms have been designed for classic problems within a simple polygon, such as triangulation [4], shortest path computation [4], or visibility [5]. These algorithms use properties of the problem considered so as to somehow compute the solution using local information whenever possible. In most cases, this ends up in an algorithm that is completely different from those used when no memory constraints exist.

Compressed Stack

A different approach was taken by Barba et al. [6], where the class of *stack algorithms* is considered. This class consists of deterministic algorithms that have a one-pass access to the input and can use a constant number of words. In addition, they allow the usage of a stack so as to store elements of the input. At any instant of time, only the top element of the stack is available to the algorithm. Note that with this additional stack, we can store up to $\Theta(n)$ values of the input. Hence, the model is not strictly speaking memory constrained.

Although this model may seem a bit artificial, Barba et al. give several examples of well-known programs that fit into this class of algorithms (such as computing the convex hull of a simple polygon or computing the visibility of a point inside a simple polygon). More interestingly, they show how to remove the auxiliary stack, effectively transforming any stack algorithm into a memory constrained algorithm that uses $O(w)$ words (for any $w \in \{1, \ldots, n\}$).

Block Reconstruction

The key to this approach lies in the fact that portions of the stack can be *reconstructed* efficiently: let $a_i, a_{i+1}, \ldots, a_j$ be a set of consecutive elements of the input (for some $i < j$) such that we know that a_i and a_j are in the stack after a_j has been processed. Then, we can determine which values in between a_i and a_j are also in the stack by re-executing the algorithm restricting the input to the $a_i, \ldots, a_j$ interval (thus, taking time proportional to $O(j-i)$).

Using this idea, we can avoid explicitly storing the whole stack by using a *compressed stack* data structure that never exceeds the size of the workspace. For the particular case in which $w = \Theta(\sqrt{n})$, the algorithm virtually partitions the input into blocks of size $\sqrt{n}$. The invariant of the data structure is that the portions of the

stack corresponding to the last two blocks that have pushed elements into the stack are stored, whereas for any other block, we only store the first and last elements that are in the stack, if any. By using a charging scheme, they show that each block triggers at most one reconstruction, and each reconstruction takes time proportional to the size of the destroyed block. In all, the compressed stack data structure reduces the size of the workspace to $\Theta(\sqrt{n})$ without asymptotically increasing the running time.

In the general case, the input is split into p equally sized blocks (where $p = \max\{2, w/\log n\}$), and each block is further subpartitioned into blocks until the blocks of the lowermost level can be explicitly stored (or the recursion exceeds size of the workspace). The smaller the size of the workspace, the higher the number of levels it will have and thus more time will be spent in reconstructing blocks. This creates a time-space trade-off for any stack algorithm whose running time is $O(\frac{n^2 \log n}{2^w})$ time for any workspace of $w \in o(\log n)$ words. For larger workspaces, the algorithm runs in $O(n^{1+1/\log p})$ time and uses $O(p \log_p n)$ words (for $2 \leq p \leq n$).

Cross-References

- ▸ Exact Algorithms and Time/Space Tradeoffs
- ▸ Memoryless Gathering of Mobile Robotic Sensors
- ▸ Rank and Select Operations on Bit Strings

Recommended Reading

1. Aleliunas R, Karp RM, Lipton R, Lovasz L, Rackoff C (1979) Random walks, universal traversal sequences, and the complexity of maze problems. In: FOCS, San Juan, pp 218–223
2. Armoni R, Ta-Shma A, Widgerson A, Zhou S (2000) An $o(\log(n)^{4/3})$ space algorithm for (s,t) connectivity in undirected graphs. J ACM 47(2):294–311
3. Asano T, Rote G (2009) Constant-working-space algorithms for geometric problems. In: CCCG, Vancouver, pp 87–90
4. Asano T, Buchin K, Buchin M, Korman M, Mulzer W, Rote G, Schulz A (2012) Memory-constrained algorithms for simple polygons. Comput Geom Theory Appl 46(8):959–969
5. Barba L, Korman M, Langerman S, Silveira R (2014) Computing the visibility polygon using few variables. Comput Geom Theory Appl 47(9):918–926
6. Barba L, Korman M, Langerman S, Sadakane K, Silveira R (2015) Space-time trade-offs for stack-based algorithms. Algorithmica 72(4):1097–1129
7. Blum M, Floyd RW, Pratt VR, Rivest RL, Tarjan RE (1973) Time bounds for selection. J Comput Syst Sci 7(4):448–461
8. Borodin A, Cook S (1982) A time-space tradeoff for sorting on a general sequential model of computation. SIAM J Comput 11:287–297
9. Chan TM (2010) Comparison-based time-space lower bounds for selection. ACM Trans Algorithms 6:1–16
10. Elmasry A, Juhl DD, Katajainen J, Satti SR (2013) Selection from read-only memory with limited workspace. In: COCOON, Hangzhou, pp 147–157
11. Frederickson GN (1987) Upper bounds for time-space trade-offs in sorting and selection. J Comput Syst Sci 34(1):19–26
12. Jarvis R (1973) On the identification of the convex hull of a finite set of points in the plane. Inf Process Lett 2(1):18–21
13. Munro JI, Paterson M (1980) Selection and sorting with limited storage. Theor Comput Sci 12:315–323
14. Nisan N, Szemeredi E, Wigderson A (1992) Undirected connectivity in $o(log^{1.5}n)$ space. In: FOCS, Pittsburg, pp 24–29
15. Pagter J, Rauhe T (1998) Optimal time-space trade-offs for sorting. In: FOCS, Palo Alto, pp 264–268
16. Pohl I (1969) A minimum storage algorithm for computing the median. IBM technical report RC2701
17. Raman V, Ramnath S (1998) Improved upper bounds for time-space tradeoffs for selection with limited storage. In: SWAT '98, Stockholm, pp 131–142
18. Reingold O (2008) Undirected connectivity in log-space. J ACM 55(4):1–24
19. Savitch WJ (1970) Relationships between nondeterministic and deterministic tape complexities. J Comput Syst Sci 4(2):177–192

Memoryless Gathering of Mobile Robotic Sensors

Paola Flocchini
School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON, Canada

Keywords

Gathering; Rendezvous; Sensors aggregation

Years and Authors of Summarized Original Work

1999; Ando, Oasa, Suzuki, Yamashita
2005; Flocchini, Prencipe, Santoro, Widmayer

Problem Definition

The Model: A *mobile robotic sensor* (or simply *sensor*) is modeled as a computational unit with sensorial capabilities: it can perceive the spatial environment within a fixed distance $V > 0$, called *visibility range*, it has its own local working memory, and it is capable of performing local computations [3,4].

Each sensor is a point in its own local Cartesian coordinate system (not necessarily consistent with the others), of which it perceives itself as the center. A sensor can move in any direction, but it may be stopped before reaching its destination, e.g. because of limits to its motion energy; however, it is assumed that the distance traveled in a move by a sensor is not infinitesimally small (unless it brings the sensor to its destination).

The sensors have no means of direct communication: any communication occurs in an implicit manner, by observing the other sensors' positions. Moreover, they are *autonomous* (i.e., without a central control) *identical* (i.e., they execute the same protocol), and *anonymous* (i.e., without identifiers that can be used during the computation).

The sensors can be *active* or *inactive*. When *active*, a sensor performs a *Look-Compute-Move* cycle of operations: it first observes the portion of the space within its visibility range obtaining a snapshot of the positions of the sensors in its range at that time (*Look*); using the snapshot as an input, the sensor then executes the algorithm to determine a destination point (*Compute*); finally, it moves toward the computed destination, if different from the current location (*Move*). After that, it becomes *inactive* and stays idle until the next activation. Sensors are *oblivious*: when a sensor becomes active, it does not remember any information from previous cycles. Note that several sensors could actually occupy the same point; we call *multiplicity detection* the ability of a sensor to see whether a point is occupied by a single sensor or by more than one.

Depending on the degree of synchronization among the cycles of different sensors, three submodels are traditionally identified: *synchronous*, *semi-synchronous*, and *asynchronous*. In the *synchronous* (FSYNC) and in the *semi-synchronous* (SSYNC) models, there is a global clock tick reaching all sensors simultaneously, and a sensor's cycle is an instantaneous event that starts at a clock tick and ends by the next. In FSYNC, at each clock tick all sensors become active, while in SSYNC only a subset of the sensors might be active in each cycle. In the *asynchronous* model (ASYNC), there is no global clock and the sensors do not have a common notion of time. Furthermore, the duration of each activity (or inactivity) is finite but unpredictable. As a result, sensors can be seen while moving, and computations can be made based on obsolete observations.

Let $S(t) = \{s_1(t), \ldots, s_n(t)\}$ denote the set of the n sensors' at time t. When no ambiguity arises, we shall omit the temporal index t. Moreover, with an abuse of notation we indicate by s_i both a sensor and its position. Let $S_i(t)$ denote the set of sensors that are within distance V from s_i at time t, that is, the set of sensors that are visible from s_i. At any point in time t, the sensors induce a *visibility graph* $G(t) = (N, E(t))$ defined as follows: $N = S$ and $\forall r, s \in N$, $(r, s) \in E(t)$ iff r and s are at distance no more than the visibility range V.

The Problem: In this setting, one of the most basic coordination and synchronization task is *Gathering*: the sensors, initially placed in arbitrary distinct positions in a 2-dimensional space, must congregate at a single location (the choice of the location is not predetermined) within finite time. In the following, we assume $n > 2$. A problem closely related to *Gathering* is *Convergence*, where the sensors need to be arbitrarily close to a common location, without the requirement of ever reaching it. A special type of convergence (also called *Near-Gathering* or *collisionless con-*

vergence) requires the sensors to converge without ever colliding with each other.

Key Results

Basic Impossibility Results

First of all, neither *Convergence* nor *Gathering* can be achieved from arbitrary initial placements if the initial visibility graph $G(0)$ is not connected. So, in all the literature, $G(0)$ is always assumed to be connected. Furthermore, if the sensors have *neither* agreement on the coordinate system *nor* multiplicity detection, then *Gathering* is not solvable in SSYNC (and thus in ASYNC), regardless of the range of visibility and the amount of memory that they may have.

Theorem 1 ([8]) *In absence of multiplicity detection and of any agreement on the coordinate systems, Gathering is deterministically unsolvable in* SSYNC.

Given this impossibility result, the natural question is whether the problem can be solved with common coordinate systems.

Gathering with Common Coordinate Systems

Assuming that the sensors agree on a common coordinate system, *Gathering* has been shown to be solvable under the weakest of the three schedulers (ASYNC) [2].

Let $\mathcal{R}$ be the rightmost vertical axis where some sensor initially lie. The idea of the algorithm is to make the sensors move toward $\mathcal{R}$, in such a way that, after a finite number of steps, they will reach it and gather at the bottommost position occupied by a sensor at that time. Let the *Look* operation of sensor s_i at time t return $S_i(t)$. The computed destination point of s_i depends on the positions of the visible sensors. Once the computation is completed, s_i moves toward its destination (but it may stop before the destination is reached). Informally,l

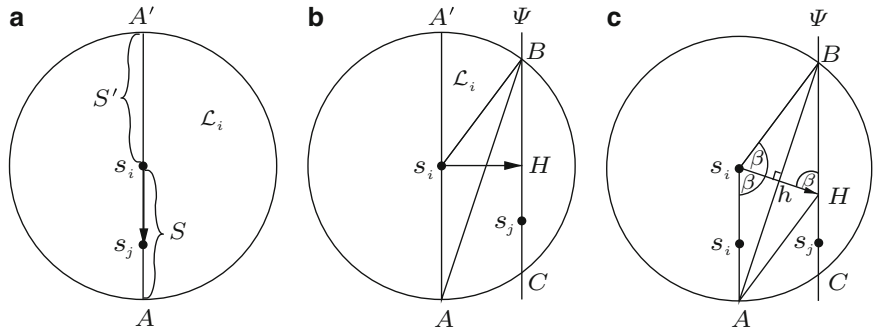
- If s_i sees sensors to its left or above on the vertical axis passing through its position (this axis will be referred to as *its vertical axis*), it does not move.
- If s_i sees sensors only below on its vertical axis, it moves down toward the nearest sensor.
- If s_i sees sensors only to its right, it moves horizontally toward the vertical axis of the nearest sensor.
- If s_i sees sensors both below on its vertical axis and on its right, it computes a destination point and performs a diagonal move to the right and down, as explained below.

To describe the diagonal movement we introduce some notation (refer to Fig. 1). Let $\overline{AA'}$ be the vertical diameter of $S_i(t)$ with A' the top and A the bottom end point; let $\mathcal{L}_i$ denote the topologically open region (with respect to $\overline{AA'}$) inside $S_i(t)$ and to the right of s_i and let $S = \overline{s_i A}$ and $S' = \overline{s_i A'}$, where neither S' and S include s_i. Let Ψ be the vertical axis of the horizontally closest sensor (if any) in $\mathcal{L}_i$.

```
Diagonal_Movement(Ψ)
  B := upper intersection between S_i(t) and Ψ;
  C := lower intersection between S_i(t) and Ψ;
  2β = A ŝ_i B;
  If β < 60° then (B, Ψ) := Rotate(s_i, B);
  H := Diagonal_Destination(Ψ, A, B);
  Move towards H.
```

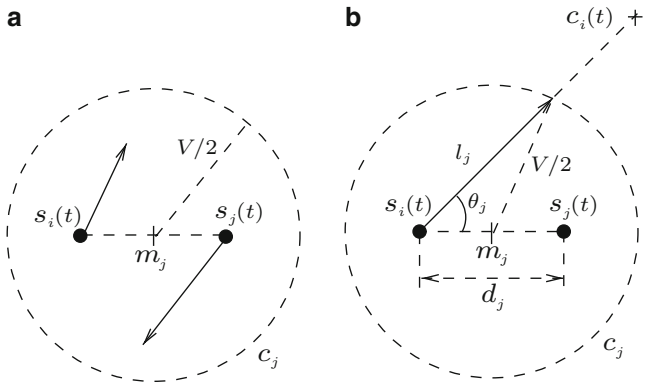
where `Rotate()` and `Diagonal_Destination()` are as follows:

- `Rotate(`s_i, B`)` rotates the segment $\overline{s_i B}$ in such a way that $\beta = 60°$ and returns the new position of B and Ψ. This angle choice ensures that the destination point is not outside the circle.
- `Diagonal_Destination(`Ψ, A, B`)` computes the destination of s_i as follows: the direction of s_i's movement is given by the perpendicular to the segment $\overline{AB}$; the destination of s_i is the point H on the intersection of the direction of its movement and of the axis Ψ.



Memoryless Gathering of Mobile Robotic Sensors, Fig. 1 From [4]: (**a**) Notation. (**b**) Horizontal move. (**c**) Diagonal move

Memoryless Gathering of Mobile Robotic Sensors, Fig. 2 From [4]: Notation for algorithm CONVERGENCE [1]



Theorem 2 ([2]) *With common coordinate systems, Gathering is possible in* ASYNC.

Gathering has been shown to be possible in SSYNC also when compasses are unstable for some arbitrary long periods, provided they have a common clockwise notion, and that they eventually stabilize, and assuming the total number of sensors is known [9].

Convergence and Near-Gathering

Convergence in SSYNC

The impossibility result does not apply to the case of *Convergence*. In fact, it is possible to solve it in SSYNC in the basic model (i.e., without common coordinate systems) [1].

Let $SC_i(t)$ denote the smallest enclosing circle of the set $\{s_j(t) | s_j \in S(t)\}$ of positions of sensors in $S(t)$; let $c_i(t)$ be the center of $SC_i(t)$.

CONVERGENCE
Assumptions: SSYNC.

If $S_i(t) = \{s_i\}$, **then** gathering is completed.
$\forall s_j \in S_i(t) \setminus \{s_i\}$:

$d_j := dist(s_i(t), s_j(t))$,
$\theta_j := c_i(t)\widehat{s_i(t)}s_j(t)$,
$l_j := (d_j/2)\cos\theta_j + \sqrt{(V/2)^2 - ((d_j/2)\sin\theta_j)^2}$,

$limit := \min_{s_j \in S_i(t)\setminus\{s_i\}}\{l_j\}$,
$goal := dist(s_i(t), c_i(t))$,
$D := \min\{goal, limit\}$,
$p :=$ point on $\overline{s_i(t)c_i(t)}$ at distance D from $s_i(t)$.
Move towards p.

Everytime s_i becomes active, it moves toward $c_i(t)$, but only up to a certain distance. Specifically, if s_i does not see any sensor other than itself, then s_i does not move. Otherwise, its destination is the point p on the segment

$\overline{s_i(t)c_i(t)}$ that is closest to $c_i(t)$ and that satisfies the following condition: For every sensor $s_j \in S(t)$, p lies in the disk $\mathcal{C}_j$ whose center is the midpoint m_j of $s_i(t)$ and $s_j(t)$ and whose range is $V/2$ (see Fig. 2). This condition ensures that s_i and s_j will still be visible after the movement of s_i, and possibly of s_j.

Theorem 3 ([1]) *Convergence is possible in* SSYNC.

Convergence in ASYNC

Convergence has been shown to be possible also in ASYNC, but under special schedulers: *partial* ASYNC [6] and *1-fair* ASYNC [5]. In *partial* ASYNC the time spent by a sensor performing the *Look*, *Compute*, and *Sleep* operations is bounded by a globally predefined amount and the time spent in the *Move* operation by a locally predefined amount; in 1-fair ASYNC between two successive activations of each sensor, all the other sensors are activated at most once. Finally, *Convergence* has been studied also in presence of perception inaccuracies (radial errors in locating a sensor) and it has been shown how to reach convergence in FSYNC for small inaccuracies.

Near-Gathering

Slight modifications can make the algorithm of [1] described above collisionless, thus solving also the *collisionless Convergence* problem (i.e., *Near-Gathering*) in SSYNC. *Near-Gathering* can be achieved also in ASYNC, with two additional assumptions [7]: 1) the sensors must partially agree on a common coordinate system (one axis is sufficient) and 2) the initial visibility graph must be *well connected*, that is, the subgraph of the visibility graph that contains only the edges corresponding to sensors at distance *strictly smaller* than V must be connected.

Open Problems

The existing results for *Gathering* and *Convergence* leave several problems open. For example, *Gathering* in SSYNC (and thus ASYNC) has been proven impossible when neither multiplicity detection nor an orientation are available. While common orientation suffices, it is not known whether the presence of multiplicity detection alone is sufficient to solve the problem. Also, the impossibility result does not apply to FSYNC; however no algorithm is known in such a setting that does not make use of orientation. Finally, it is not known whether *Convergence* (collisionless or not) is solvable in ASYNC without additional assumptions: so far no algorithm has been provided nor an impossibility proof.

Recommended Reading

1. Ando H, Oasa Y, Suzuki I, Yamashita M (1999) A distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Trans Robot Autom 15(5):818–828
2. Flocchini P, Prencipe G, Santoro N, Widmayer P (2005) Gathering of asynchronous mobile robots with limited visibility. Theor Comput Sci 337: 147–168
3. Flocchini P, Prencipe G, Santoro N (2011) Computing by mobile robotic sensors. In: Nikoletseas S, Rolim J (eds) Theoretical aspects of distributed computing in sensor networks, chap 21. Springer, Heidelberg. ISBN:978-3-642-14849-1
4. Flocchini P, Prencipe G, Santoro N (2012) Distributed computing by oblivious mobile robots. Morgan & Claypool, San Rafael
5. Katreniak B (2011) Convergence with limited visibility by asynchronous mobile robots. In: 18th international colloquium on structural information and communication complexity (SIROCCO), Gdańsk, Poland, pp 125–137
6. Lin J, Morse A, Anderson B (2007) The multi-agent rendezvous problem. Part 2: the asynchronous case. SIAM J Control Optim 46(6): 2120–2147
7. Pagli L, Prencipe G, Viglietta G (2012) Getting close without touching. In: 19th international colloquium on structural information and communication complexity (SIROCCO), Reykjavík, Iceland, pp 315–326
8. Prencipe G (2007) Impossibility of gathering by a set of autonomous mobile robots. Theor Comput Sci 384(2–3):222–231
9. Souissi S, Défago X, Yamashita M (2009) Using eventually consistent compasses to gather memoryless mobile robots with limited visibility. ACM Trans Auton Adapt Syst 4(1):1–27

Meshing Piecewise Smooth Complexes

Tamal Krishna Dey
Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA

Keywords

Delaunay mesh; Delaunay refinement; Piecewise smooth complex; Topology; Volume mesh

Years and Authors of Summarized Original Work

2006; Boissonnat, Oudot
2007; Cheng, Dey, Ramos
2007: Cheng, Dey, Levine
2012; Cheng, Dey, Shewchuk

Problem Definition

The class of piecewise smooth complex (PSC) includes geometries that go beyond smooth surfaces. They contain polyhedra, smooth and non-smooth surfaces with or without boundaries, and more importantly non-manifolds. Thus, provable mesh generation algorithms for this domain extend the scope of mesh generation to a wide variety of domains. Just as in surface mesh generation, we are required to compute a set of points on the input complex and then connect them with a simplicial complex which is *geometrically* close and is *topologically* equivalent to the input. One challenge that makes this task harder is that the PSCs allow arbitrary small input angles, a notorious well-known hurdle for mesh generation.

A PSC is a set of *cells*, each being a smooth, connected manifold, possibly with boundary. The 0-cells, 1-cells, and 2-cells are called corners, ridges, and patches, respectively. A PSC could also contain 3-cells that designate regions to be meshed with tetrahedra, if we are interested in a volume mesh.

Definition 1 (ridge; patch) A *ridge* is a closed, connected subset of a smooth 1-manifold without boundary in $\mathbb{R}^3$. A *patch* is a 2-manifold that is a closed, connected subset of a smooth 2-manifold without boundary in $\mathbb{R}^3$.

Definition 2 A *piecewise smooth complex* $\mathcal{S}$ is a finite set of vertices, ridges, and patches that satisfy the following conditions.

1. The boundary of each cell in $\mathcal{S}$ is a union of cells in $\mathcal{S}$.
2. If two distinct cells c_1 and c_2 in $\mathcal{S}$ intersect, their intersection is a union of cells in $\mathcal{S}$ included in the boundary of c_1 or c_2.

Our goal is to generate a triangulation of a PSC. Element quality is not a primary issue here though a good radius-edge ratio can be obtained by additional refinement except near the small input angles. The definition below makes our notion of triangulation of a PSC precise. Recall that $|T|$ denotes the underlying space of a complex T (Fig. 1).

Definition 3 (triangulation of a piecewise smooth complex) A simplicial complex T is a *triangulation* of a PSC $\mathcal{S}$ if there is a homeomorphism h from $|\mathcal{S}|$ to $|T|$ such that $h(v) = v$ for each vertex $v \in \mathcal{S}$ and for each cell $\xi \in \mathcal{S}$, there is a subcomplex $T_\xi \subseteq T$ such that h is a homeomorphism from ξ to $|T_\xi|$.

Key Results

To generate a mesh for a PSC with theoretical guarantees, we use Delaunay refinement as in the smooth surface meshing. For a point set $P \subset \mathbb{R}^3$, let Vor P and Del P denote the Voronoi diagram and Delaunay triangulation of P, respectively. The restricted Delaunay complex as in the smooth surface meshing plays an important role in sampling the PSCs.

Meshing Piecewise Smooth Complexes, Fig. 1 Example meshes of PSC: (left) a piecewise smooth surface, a non-manifold, a surface with non-trivial topology

Let V_ξ denote the dual Voronoi face of a Delaunay simplex ξ in Del P. The restricted Voronoi face of V_ξ with respect to $\mathbb{X} \subset \mathbb{R}^3$ is the intersection $V_\xi|_\mathbb{X} = V_\xi \cap \mathbb{X}$. The *restricted Voronoi diagram* and *restricted Delaunay triangulation* of P with respect to $\mathbb{X}$ are

$$\text{Vor } P|_\mathbb{X} = \{V_\xi|_\mathbb{X} \mid V_\xi|_\mathbb{X} \neq \emptyset\} \text{ and Del } P|_\mathbb{X} = \{\xi \mid V_\xi|_\mathbb{X} \neq \emptyset\} \text{ respectively.}$$

In words, Del $P|_\mathbb{X}$ consists of those Delaunay simplices in Del P whose dual Voronoi face intersects $\mathbb{X}$. We call these simplices *restricted*. For a restricted triangle, its dual Voronoi edge intersects the domain in a single or multiple points. These are the centers of *surface Delaunay balls* that circumscribe the vertices of the triangle.

In smooth surface meshing, the restricted triangles violating certain desirable properties are refined by the addition of the surface Delaunay ball centers. It turns out that this process cannot continue forever because the inserted points maintain a fixed lower bound on their distances from the existing points. This argument breaks down if non-smoothness is allowed. In particular, ridges and corners where several patches meet cause the local feature size to be zero in which case inserted points with a lower bound on local feature size may come arbitrarily close to each other. Nevertheless, Boissonnat and Oudot [1] showed that the Delaunay refinement that they proposed for smooth surfaces can be extended to a special class of piecewise smooth surfaces called *Lipschitz surfaces*. Their algorithm may break down for domains with small angles. The analysis requires that the input angles subtended by the tangent planes of the patches meeting at the ridges or a corner are close to 180°.

The first guaranteed algorithm for PSCs with small angles is due to Cheng, Dey, and Ramos [3]. They introduced the idea of using weighted vertices as protecting balls in a weighted Delaunay triangulation. In this triangulation each point p is equipped with a weight w_p which can also be viewed as a ball $\hat{p} = B(p, w_p)$ centered at p with radius w_p. The squared weighted distance between two points (p, w_p) and (q, w_q) is measured as $\|p-q\|^2 - w_p^2 - w_q^2$. Notice that the weight can be zero in which case the weighted point is a regular point. One can define a Voronoi diagram and its dual Delaunay triangulation for a weighted point set just like their Euclidean counterparts by replacing Euclidean distances with weighted distances. To emphasize the weighted case, let us denote a weighted point set P with $P[w]$ and its weighted Delaunay triangulation with Del $P[w]$.

The algorithm of Cheng, Dey, and Ramos [3] has two phases, the *protection* phase and the *refinement* phase. In the protection phase, it computes a set of protecting balls centered at the ridges and corners of the input PSC. The union of the protecting balls cover the ridges and corners completely. Let $P[w]$ be the weighted points representing the protecting balls. The algorithm computes Del $P[w]$ and the restricted Delaunay

triangles in Del $P[w]|_{\mathcal{S}}$. In the refinement phase, it refines the restricted triangles if either they have a large surface Delaunay ball (albeit in the weighted sense), or they fail to form a topological disk around each vertex on each patch adjoining the vertex. The algorithm guarantees that the final mesh is a triangulation of the input PSC and the two are related by a homeomorphism. The proof of the homeomorphism uses an extension of the topological ball property of Edelsbrunner and Shah [6] to accommodate PSCs.

The algorithm of Cheng et al. [3] requires difficult geometric computations such as feature size estimates. Cheng, Dey, and Levine [2] simplified some of these computations at the expense of weakening the topological guarantees. Like the smooth surface meshing algorithm in [4], they guarantee that each input patch and ridge is approximated with output simplices that form a manifold of appropriate dimension. The algorithm is supplied with a user-specified size parameter. If this size parameter is small enough, the output is a triangulation of the PSC in the sense of Definition 3. In a subsequent paper, Dey and Levine [5] proposed a strategy to combine the protection phase with refinement phase which allowed to adjust the ball sizes on the fly rather than computing them beforehand by estimating feature sizes. Cheng, Dey, and Shewchuk [4] refined this strategy even further to have an improved algorithm with detailed analysis.

Theorem 1 ([4]) *There is a Delaunay refinement algorithm that runs with a parameter $\lambda > 0$ on an input PSC $\mathcal{S}$ and outputs a mesh $T = \text{Del}\, P[w]|_{\mathcal{S}}$ with the following guarantees:*

1. *For each patch $\sigma \in \mathcal{S}$, $|\text{Del}\, P[w]|_{\sigma}|$ is a 2-manifold with boundary, and every vertex in $\text{Del}\, P[w]|_{\sigma}$ lies on σ. The boundary of $|\text{Del}\, P[w]|_{\sigma}|$ is homeomorphic to $\text{B}d\,\sigma$, the boundary of σ, and every vertex in $\text{Del}\, P[w]|_{\text{B}d\,\sigma}$ lies on $\text{B}d\,\sigma$.*
2. *If λ is sufficiently small, then T is a triangulation of $\mathcal{S}$ (recall Definition 3). Furthermore, there is a homeomorphism h from $|\mathcal{S}|$ to $|T|$ such that for every i-dimensional cell $\xi \in \mathcal{S}_i$ with $i \in [0, 2]$, h is a homeomorphism from ξ to $\left|\text{Del}\, P[w]|_{\xi}\right|$, every vertex in $\text{Del}\, P[w]|_{\xi}$ lies on ξ, and the boundary of $\left|\text{Del}\, P[w]|_{\xi}\right|$ is $\left|\text{Del}\, P[w]|_{\text{B}d\,\xi}\right|$*

Notice that the above guarantee specifies that the homeomorphism between the input and the output respects the stratification of corners, ridges, and patches and thus preserves these features. Once a mesh for the surface patches is completed, Delaunay refinement algorithms can further refine the mesh to improve the quality of the surface triangles or the tetrahedra they enclose. The algorithm can only attack triangles and tetrahedra with large orthoradius-edge ratios; some simplices with large circumradius-edge ratios may survive. The tetrahedral refinement algorithm should be careful in that if inserting a vertex at the circumcenter of a poor-quality tetrahedron destroys some surface triangle, the algorithm simply should opt not to insert the new vertex. This approach has the flaw that tetrahedra with large radius-edge ratios sometimes survive near the boundary.

URLs to Code and Data Sets

CGAL(http://cgal.org), a library of geometric algorithms, contains software for mesh generation of piecewise smooth surfaces. The DelPSC software that implements the PSC meshing algorithm as described in [4] is available from http://web.cse.ohio-state.edu/~tamaldey/delpsc.html.

Cross-References

- 3D Conforming Delaunay Triangulation
- Smooth Surface and Volume Meshing

Recommended Reading

1. Boissonnat J-D, Oudot S (2006) Provably good sampling and meshing of Lipschitz surfaces. In: Proceedings of the 22nd annual symposium on computational geometry, Sedona, pp 337–346
2. Cheng S-W, Dey TK, Levine J (2007) A practical Delaunay meshing algorithm for a large class of domains.

In: Proceedings of the 16th international meshing roundtable, Seattle, pp 477–494
3. Cheng S-W, Dey TK, Ramos EA (2007) Delaunay refinement for piecewise smooth complexes. In: Proceedings of the 18th annual ACM-SIAM symposium on discrete algorithms, New Orleans, pp 1096–1105
4. Cheng S-W, Dey TK, Shewchuk JR (2012) Delaunay mesh generation. CRC, Boca Raton
5. Dey TK, Levine JA (2009) Delaunay meshing of piecewise smooth complexes without expensive predicates. Algorithms 2(4):1327–1349
6. Edelsbrunner H, Shah N (1997) Triangulating topological spaces. Int J Comput Geom Appl 7:365–378

Message Adversaries

Michel Raynal
Institut Universitaire de France and IRISA, Université de Rennes, Rennes, France

Keywords

Dynamic transmission failure; Mobile failure; Synchronous dynamic network; Ubiquitous failures

Years and Authors of Summarized Original Work

1989; Santoro, Widmayer

Problem Definition: The Notion of a Message Adversary

Message adversaries have been introduced by N. Santoro and P. Widmayer in a paper titled *Time is not a healer* [15] to model and understand what they called *dynamic* transmission failures in the context of synchronous systems. Then, they extended their approach in [16] where they used the term *ubiquitous* failures. The terms *heard-of communication* [5], *transient link failure* [17], and *mobile failure* [12] have later been used by other authors to capture similar network behaviors in synchronous or asynchronous systems.

The aim of this approach is to consider message losses as a normal link behavior (as long as messages are correctly transmitted). The notion of a message adversary is of a different nature than the notion of the fair link assumption. A fair link assumption is an assumption on each link taken separately, while the message adversary notion considers the network as a whole; its aim is not to build a reliable network but to allow the statement of connectivity requirements that must be met for problems to be solved. Message adversaries allow us to consider topology changes not as anomalous network behaviors, but as an essential part of the deep nature of the system.

Reliable Synchronous Systems

A fully connected synchronous system is made up of n computing entities (processes) denoted $p_1, \ldots, p_n$, where each pair of processes is connected by a bidirectional link. The progress of the processes is ruled by a global clock which generates a sequence of rounds. During a round, each process (a) first sends a message to all the other processes (broadcast), (b) then receives a message from each other process, and (c) finally executes a local computation. The fundamental property of a synchronous system is that the messages sent at a round r are received during the very same round r. As we can see, this type of synchrony is an abstraction that encapsulates (and hides) specific timing assumptions (there are bounds on message transfer delays and processing times, and those are known and used by the underlying system level to generate the sequence of synchronized rounds [13]).

In the case of a reliable synchronous system, both processes and links are reliable, i.e., no process deviates from its specification, and all the messages that are sent, and only them, are received (exactly once) by each process.

Message Adversary

A *message adversary* is a daemon which, at each round, can suppress messages (hence, these messages are never received). The adversary is not prevented from having a read access to the

local states of the processes at the beginning of each round.

It is possible to associate a directed graph G^r with each round r. Its vertices are the processes, and there is an edge from p_i to p_j if the message sent at round r by p_i to p_j is not suppressed by the adversary. There is a priori no relation on the consecutive graphs G^r, G^{r+1}, etc. As an example, the daemon can define G^{r+1} from the local states of the processes at the end of round r.

Let $\mathcal{SMP}_n[\text{adv} : \text{AD}]$ denote the synchronous system whose communications are under the control of an adversary-denoted AD. $\mathcal{SMP}_n[\text{adv} : \emptyset]$ denotes the synchronous system in which the adversary has no power (it can suppress no message), while $\mathcal{SMP}_n[\text{adv} : \infty]$ denotes the synchronous system in which the adversary can suppress all the messages at every round. It is easy to see that, from a message adversary and computability point of view, $\mathcal{SMP}_n[\text{adv} : \emptyset]$ is the most powerful synchronous system model, while $\mathcal{SMP}_n[\text{adv} : \infty]$ is the weakest. More generally, the more constrained the message adversary AD, the more powerful the synchronous system.

Key Results

Key Results in Synchronous Systems

The Spanning Tree Adversary

Let TREE be the message adversary defined by the following constraint: at every round r, the graph G^r is an undirected spanning tree, i.e., the adversary cannot suppress the two messages – one in each direction – sent on the edges of G^r. Let $\mathcal{SMP}_n[\text{adv} : \text{TREE}]$ denote the corresponding synchronous system. As already indicated, for any r and $r' \neq r$, G^r and $G^{r'}$ are not required to be related; they can be composed of totally different sets of links.

Let us assume that each process p_i has an initial input v_i. It is shown in [11] that $\mathcal{SMP}_n[\text{adv} : \text{TREE}]$ allows the processes to compute any computable function on their inputs, i.e., functions on the vector $[v_1, \ldots, v_n]$.

Solving this problem amounts to ensure that each input v_i attains each process p_j despite the fact that the spanning tree can change arbitrarily from a round to the next one. This follows from the following observation. At any round r, the set of processes can be partitioned into two subsets: the set yes_i which contains the processes that have received v_i, and the set no_i which contains the processes that have not yet received v_i. As G^r is an undirected spanning tree (the tree is undirected because no message is suppressed on each of its edges), it follows that there is an edge of G^r that connects a process of the set yes_i to a process that belongs to the set no_i. So during round r, there at least one process of the set no_i which receives a copy of v_i and will consequently belong to the set yes_i of the next round. It follows that at most $(n-1)$ rounds are necessary for v_i to attain all the processes.

Consensus in the Presence of Message Adversaries

In the consensus problem, each process proposes a value and has to decide a value v such that v was proposed by a process, and no two processes decide different values. This problem is addressed in [2, 6, 8, 12, 17] in the message adversary context.

Impossibility Agreement-Related Results

As presented in [15], the k-process agreement problem (which must not be confused with the k-set agreement problem) is defined as follows. Each process p_i proposes an input value $v_i \in \{0, 1\}$, and at least k processes have to decide the same proposed value v. Let us observe that this problem can be trivially solved without any communication when $k \leq \lceil \frac{n}{2} \rceil$ (namely, each process decides its input value). Let us also notice that if $k > \lceil \frac{n}{2} \rceil$, there is at most one decision value.

It is shown in [15] that the k-process agreement problem cannot be solved for $k > \lceil \frac{n}{2} \rceil$, if the message adversary is allowed to suppress up to $(n-1)$ messages at every round. Other impossibility results are presented in [16]. More results can be found in [3].

d-Solo Executions

A process runs *solo* when it computes its local output without receiving any information from other processes, either because they crashed or they are too slow. This corresponds to a message adversary that suppresses all the messages sent to some subset of processes (which consequently run solo). The computability power of models in which several processes may run solo is addressed in [9]. This paper introduces the concept of d*-solo* model, $1 \leq d \leq n$ (synchronous round-based wait-free models where up to d processes may run solo in each round), and characterizes the colorless tasks that can be solved in a d*-solo* model. Among other results, this paper shows that the (d, ϵ)*-solo approximate agreement* task (which generalizes ϵ-approximate agreement) can be solved in the d-solo model, but cannot be solved in the $(d+1)$-solo model. Hence, the d*-solo* models define a strict hierarchy.

Key Results in Asynchronous Systems

In a very interesting way, message adversaries allows the establishment of equivalences between synchronous systems and asynchronous systems.

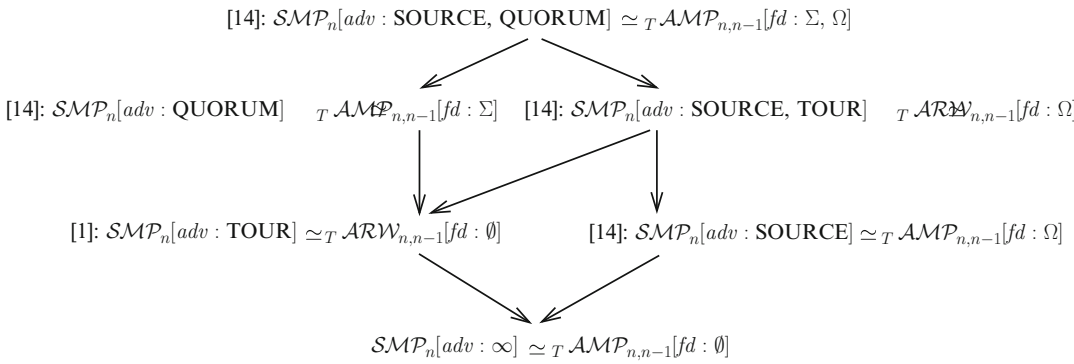
These equivalences, which are depicted in Fig. 1 (from [14]), concern tasks. A task is the distributed analogous of a mathematical function [10], in the sense that each process has an input and must compute an output, and the processes need to cooperate to compute their individual outputs (this cooperation is inescapable, which makes the task *distributed*).

$A \simeq_T B$ means that any task that can be computed in the model A can be computed in the model B and vice versa. An arrow from A to B means that, from a task solvability point of view, the model A is strictly stronger than the model B (there are tasks solvable in A and not in B, and all the tasks solvable in B are solvable in A).

Let $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$ (resp. $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$) denote the basic asynchronous read/write (resp. message-passing) system model in which up to $(n-1)$ processes may crash (premature halting). These models are also called wait-free models. The notation $[fd : \emptyset]$ means that these systems are not enriched with additional computational power. Differently, $\mathcal{ARW}_{n,n-1}[fd : \text{FD}]$ (resp. $\mathcal{AMP}_{n,n-1}[fd : \text{FD}]$) denotes $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$ (resp. $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$) enriched with a failure detector FD (see below).

- The message adversary-denoted TOUR (four tournament) has been introduced By Afek and Gafni in [1]. At any round, this adversary can suppress one message on each link but not both: for any pair of processes (p_i, p_j), either the message from p_i to p_j or the message from p_j to p_i or none of them can be suppressed.

  The authors have shown the following model equivalence: $\mathcal{SMP}_n[\text{adv} : \text{TOUR}] \simeq_T \mathcal{ARW}_{n,n-1}[fd : \emptyset]$. This is an important result as it established for the first time a



Message Adversaries, Fig. 1 A message adversary-hierarchy based on task equivalence and failure detectors

very strong relation linking message-passing synchronous systems where no process crashes, but messages can be lost according to the adversary TOUR, with the basic asynchronous wait-free read/write model.

- The other model equivalences, from a task solvability point of view, are due to Raynal and Stainer [14], who considered two failure detectors and introduced two associated message adversaries.

  A failure detector is an oracle that provides processes with information on failures. The failure detector Ω, called "eventual leader," was introduced in [4]. It is the failure detector that provides the minimal information on failures that allow consensus to be solved in $\mathcal{ARW}_{n,n-1}[fd : \emptyset]$ and in $\mathcal{AMP}_{n,n-1}[fd : \emptyset]$ where a majority of processes do not crash. The failure detector Σ, which is called "quorum", was introduced in [7]. It is the failure detector that provides the minimal information on failures that allow a read/write register to be built on top of an asynchronous message-passing system prone to up to $(n-1)$ process crashes.

  The message adversary SOURCE is constrained by the following property: there is a process p_x and a round r such that, at any round $r' > r$, the adversary does not suppress the message sent by p_x to the other processes. The message adversary QUORUM captures the message patterns that allow to obtain the quorums defined by Σ.

  As indicated, the corresponding model equivalences are depicted on Fig. 1. As an example, when considering distributed tasks, the synchronous message-passing model where no process crashes and where the message adversary is constrained by SOURCE and TOUR ($\mathcal{SMP}_n$[adv : SOURCE, TOUR]) and the basic wait-free asynchronous read/write model enriched with Ω ($\mathcal{ARW}_{n,n-1}[fd : \Omega]$) have the same computability power.

  When looking at the figure, it is easy to see that the suppression of the constraint TOUR from the model $\mathcal{SMP}_n$[adv : SOURCE, TOUR] gives the model $\mathcal{SMP}_n$[adv : SOURCE], which is equivalent to $\mathcal{AMP}_{n,n-1}[fd : \Omega]$. Hence, suppressing TOUR weakens the model from $\mathcal{ARW}_{n,n-1}[fd : \Omega]$ to $\mathcal{AMP}_{n,n-1}[fd : \Omega]$ (i.e., from asynchronous read/write with Ω to asynchronous message-passing with Ω).

Applications

Message adversaries are important because they allow network changes in synchronous systems to be easily captured. Message losses are no longer considered as link or process failures, but as a normal behavior generated by process mobility and wireless links. Message adversaries provide us with a simple way to state assumptions (and sometimes minimal assumptions) on link connectivity which allow distributed computing problems to be solved in synchronous systems. They also allow the statement of equivalences relating (a) synchronous systems weakened by dynamically changing topology and (b) asynchronous read/write (or message-passing) systems enriched with distinct types of failure detectors.

Cross-References

- ▸ Distributed Computing for Enumeration
- ▸ Failure Detectors
- ▸ Locality in Distributed Graph Algorithms

Recommended Reading

1. Afek Y, Gafni E (2013) Asynchrony from synchrony. In: Proceedings of the international conference on distributed computing and networking (ICDCN'13), Mumbai. LNCS, vol 7730. Springer, pp 225–239
2. Biely M, Robinson P, Schmid U (2012) Agreement in directed dynamic networks. In: Proceedings of the 19th international colloquium on structural information and communication complexity (SIROCCO'12), Reykjavik. LNCS, vol 7355. Springer, pp 73–84
3. Casteigts P, Flocchini P, Godard E, Santoro N, Yamashita M (2013) Expressivity of time-varying

graphs. In: Proceedings of the 19th international symposium on fundamentals of computation theory (FST'13), Liverpool. LNCS, vol 8070. Springer, pp 95–106
4. Chandra T, Hadzilacos V, Toueg S (1996) The weakest failure detector for solving consensus. J ACM 43(4):685–722
5. Charron-Bost B, Schiper A (2009) The *heard-of* model: computing in distributed systems with benign faults. Distrib Comput 22(1):49–71
6. Coulouma E, Godard E (2013) A characterization of dynamic networks where consensus is solvable. In: Proceedings of the 19th international colloquium on structural information and communication complexity (SIROCCO'13), Ischia. LNCS, vol 8179. Springer, pp 24–35
7. Delporte-Gallet C, Fauconnier H, Guerraoui R (2010) Tight failure detection bounds on atomic object implementations. J ACM 57(4):Article 22
8. Godard E, Peters GP (2011) Consensus vs broadcast in communication networks with arbitrary mobile omission faults. In: Proceedings of the 17th international colloquium on structural information and communication complexity (SIROCCO'11), Gdansk. LNCS, vol 6796. Springer, pp 29–41
9. Herlihy M, Rajsbaum S, Raynal M, Stainer J (2014) Computing in the presence of concurrent solo executions. In: Proceedings of the 11th Latin-American theoretical informatics symposium (LATIN'2014), Montevideo. LNCS, vol 8392. Springer, pp 214–225
10. Herlihy MP, Shavit N (1999) The topological structure of asynchronous computability. J ACM 46(6):858–923
11. Kuhn F, Lynch NA, Oshman R (2010) Distributed computation in dynamic networks. In: Proceedings of the 42nd ACM symposium on theory of computing (STOC'10), Cambridge. ACM, pp 513–522
12. Moses Y, Rajsbaum S (2002) A layered analysis of consensus. SIAM J Comput 31:989–1021
13. Raynal M (2010) Fault-tolerant agreement in synchronous message-passing systems. Morgan & Claypool Publishers, 165p. ISBN:978-1-60845-525-6
14. Raynal M, Stainer J (2013) Synchrony weakened by message adversaries vs asynchrony restricted by failure detectors. In: Proceedings of the 32nd ACM symposium on principles of distributed computing (PODC '13), Montréal. ACM, pp 166–175
15. Santoro N, Widmayer P (1989) Time is not a healer. In: Proceedings of the 6th annual symposium on theoretical aspects of computer science (STACS'89), Paderborn. LNCS, vol 349. Springer, pp 304–316
16. Santoro N, Widmayer P (2007) Agreement in synchronous networks with ubiquitous faults. Theor Comput Sci 384(2–3):232–249
17. Schmid U, Weiss B, Keidar I (2009) Impossibility results and lower bounds for consensus under link failures. SIAM J Comput 38(5):1912–1951

Metric TSP

Markus Bläser
Department of Computer Science, Saarland University, Saarbrücken, Germany

Keywords

Metric traveling salesman problem; Metric traveling salesperson problem

Years and Authors of Summarized Original Work

1976; Christofides

Problem Definition

The *Traveling Salesman Problem (TSP)* is the following optimization problem:

Input: A complete loopless undirected graph $G=(V, E, w)$ with a weight function w: $E \to \mathbb{Q}_{\geq 0}$ that assigns to each edge a nonnegative weight.

Feasible solutions: All Hamiltonian tours, i.e., the subgraphs H of G that are connected, and each node in them that has degree two.

Objective function: The weight function $w(H) = \sum_{e \in H} w(e)$ of the tour.

Goal: Minimization.

The TSP is an NP-hard optimization problem. This means that a polynomial time algorithm for the TSP does not exist unless $\mathsf{P} = \mathsf{NP}$. One way out of this dilemma is provided by *approximation algorithms*. A polynomial time algorithm for the TSP is called an α-approximation algorithm if the tour H produced by the algorithm fulfills $w(H) \leq \alpha \cdot \mathrm{OPT}(G)$. Here $\mathrm{OPT}(G)$ is the weight of a minimum weight tour of G. If G is clear from the context, one just writes OPT. An α-approximation algorithm always produces a feasible solution whose objective value is at most a factor of α away from the optimum value. α is also called the approximation factor or performance guarantee. α does not need to be

a constant; it can be a function that depends on the size of the instance or the number of nodes n.

If there exists a polynomial time approximation algorithm for the TSP that achieves an exponential approximation factor in n, then $\mathsf{P} = \mathsf{NP}$ [6]. Therefore, one has to look at restricted instances. The most natural restriction is the *triangle inequality*, that means,

$$w(u, v) \leq w(u, x) + w(x, v) \quad \text{for all } u, v, x \in V.$$

The corresponding problem is called the *Metric TSP*. For the Metric TSP, approximation algorithms that achieve a constant approximation factor exist. Note that for the Metric TSP, it is sufficient to find a tour that visits each vertex *at least* once: Given such a tour, we can find a Hamiltonian tour of no larger weight by skipping every vertex that we already visited. By the triangle inequality, the new tour cannot get heavier.

Key Results

A simple 2-approximation algorithm for the Metric TSP is the *tree doubling algorithm*. It uses minimum spanning trees to compute Hamiltonian tours. A *spanning tree* T of a graph $G = (V, E, w)$ is a connected acyclic subgraph of G that contains each node of V. The weight $w(T)$ of such a spanning tree is the sum of the weights of the edges in it, i.e., $w(T) = \sum_{e \in T} w(e)$. A spanning tree is called a minimum spanning tree if its weight is minimum among all spanning trees of G. One can efficiently compute a minimum spanning tree, for instance via Prim's or Kruskal's algorithm, see e.g., [5].

The tree doubling algorithm seems to be folklore. The next lemma is the key for proving the upper bound on the approximation performance of the tree doubling algorithm.

Lemma 1 *Let T be a minimum spanning tree of $G = (V, E, w)$. Then $w(T) \leq \mathrm{OPT}$.*

Proof If one deletes any edge of a Hamiltonian tour of G, one gets a spanning tree of G. □

Algorithm 1 Tree doubling algorithm

Input: a complete loopless edge weighted undirected graph $G = (V, E, w)$ with weight function $w : E \to \mathbb{Q}_{\geq 0}$ that fulfills the triangle inequality

Output: a Hamiltonian tour of G that is a 2" approximation

1: Compute a minimum spanning tree T of G.
2: Duplicate each edge of T and obtain a Eulerian multigraph T'.
3: Compute a Eulerian tour of T' (for instance via a depth first search in T). Whenever a node is visited in the Eulerian tour that was already visited, this node is skipped and one proceeds with the next unvisited node along the Eulerian cycle. (This process is called *shortcutting*.) Return the resulting Hamiltonian tour H.

Theorem 2 *Algorithm 1 alwaysreturns a Hamiltonian tour whose weight is at most twice the weight of an optimum tour. Its running time is polynomial.*

Proof By Lemma 1, $w(T) \leq \mathrm{OPT}$. Since one duplicates each edge of T, the weight of T' equals $w(T') = 2w(T) \leq 2\mathrm{OPT}$. When taking shortcuts in step 3, a path in T' is replaced by a single edge. By the triangle inequality, the sum of the weights of the edges in such a path is at least the weight of the edge it is replaced by. (Here, the algorithm breaks down for arbitrary weight functions.) Thus $w(H) \leq w(T')$. This proves the claim about the approximation performance.

The running time is dominated by the time needed to compute a minimum spanning tree. This is clearly polynomial. □

Christofides' algorithm (Algorithm 2) is a clever refinement of the tree doubling algorithm. It first computes a minimum spanning tree. On the nodes that have an odd degree in T, it then computes a minimum weight perfect matching. A matching M of G is called a matching on $U \subseteq V$ if all edges of M consist of two nodes from U. Such a matching is called *perfect* if every node of U is incident with an edge of M.

Lemma 3 *Let$U \subseteq V, \#U$ even. Let M be a minimum weight perfect matching on U. Then $w(M) \leq \mathrm{OPT}/2$.*

Algorithm 2 Christofides' algorithm

Input: a complete loopless edge weighted undirected graph $G = (V, E, w)$ with weight function $w : E \to \mathbb{Q}_{\geq 0}$ that fulfills the triangle inequality

Output: a Hamiltonian tour of G that is a 3/2" approximation

1: Compute a minimum spanning tree T of G.
2: Let $U \subseteq V$ be the set of all nodes that have odd degree in T. In G, compute a minimum weight perfect matching M on U.
3: Compute a Eulerian tour of $T \cup M$ (considered as a multigraph).
4: Take shortcuts in this Eulerian tour to a Hamiltonian tour H.

Proof Let H be an optimum Hamiltonian tour of G. One takes shortcuts in H to get a tour H' on $G|_U$ as follows: H induces a permutation of the nodes in U, namely the order in which the nodes are visited by H. One connects the nodes of U in the order given by the permutation. To every edge of H' corresponds a path in H connecting the two nodes of this edge. By the triangle inequality, $w(H') \leq w(H)$. Since $\#U$ is even, H' is the union of two matchings. The lighter one of these two has a weight of at most $w(H')/2 \leq \mathrm{OPT}/2$. □

One can compute a minimum weight perfect matching in time $O(n^3)$, see for instance [5].

Theorem 4 *Algorithm 2 is a 3/2-approximation algorithm with polynomial running time.*

Proof First observe that the number of odd degree nodes of the spanning tree is even, since the sum of the degrees of all nodes equals $2(n-1)$, which is even. Thus a perfect matching on U exists. The weight of the Eulerian tour is obviously $w(T) + w(M)$. By Lemma 1, $w(T) \leq \mathrm{OPT}$. By Lemma 3, $w(M) \leq \mathrm{OPT}/2$. The weight $w(H)$ of the computed tour H is at most the weight of the Eulerian tour by the triangle inequality, i.e., $w(H) \leq \frac{3}{2}\mathrm{OPT}$. Thus the algorithm is a 3/2-approximation algorithm. Its running time is $O(n^3)$. □

Applications

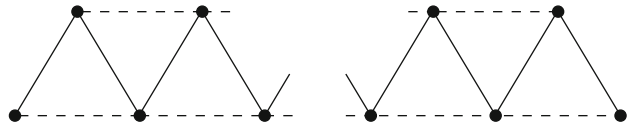
Experimental analysis shows that Christofides' algorithm itself deviates by 10 % to 15 % from the optimum tour [3]. However, it can serve as a good starting tour for other heuristics like the Lin–Kernigham heuristic.

Open Problems

The analysis of Algorithm 2 is tight; an example is the metric completion of the graph depicted in Fig. 1. The unique minimum spanning tree consists of all solid edges. It has only two nodes of odd degree. The edge between these two nodes has weight $(1+\epsilon)(n+1)$. No shortcuts are needed, and the weight of the tour produced by the algorithm is $\approx 3n$. An optimum tour consists of all dashed edges plus the leftmost and rightmost solid edge. The weight of this tour is $(2n-1)(1+\epsilon)+2 \approx 2n$.

The question whether there is an approximation algorithm with a better performance guarantee is a major open problem in the theory of approximation algorithms.

Held and Karp [2] design an LP based algorithm that computes a lower bound for the weight of an optimum TSP tour. It is conjectured that the weight of an optimum TSP tour is at most a factor of 4/3 larger than this lower bound, but this conjecture is unproven for more than three decades. An algorithmic proof of this conjecture



Metric TSP, Fig. 1 A tight example for Christofides' algorithm. There are $2n+1$ nodes. *Solid edges* have a weight of one, *dashed ones* have a weight of $1+\epsilon$

would yield an 4/3-approximation algorithm for the Metric TSP.

Experimental Results

See e.g., [3], where a deviation of 10 % to 15 % of the optimum (more precisely of the Held–Karp bound) is reported for various sorts of instances.

Data Sets

The webpage of the 8th DIMACS implementation challenge, www.research.att.com/~dsj/chtsp/, contains a lot of instances.

Cross-References

▸ Minimum Spanning Trees

Recommended Reading

Christofides never published his algorithm. It is usually cited as one of two technical reports from Carnegie Mellon University, TR 388 of the Graduate School of Industrial Administration (now Tepper School of Business) and CS-93-13. None of them seem to be available at Carnegie Mellon University anymore [Frank Balbach, personal communication, 2006]. A one-page abstract was published in a conference record. But his algorithm quickly found his way into standard textbooks on algorithm theory, see [7] for a recent one.

1. Christofides N (1976) Worst case analysis of a new heuristic for the traveling salesman problem, Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh. Also: Carnegie-Mellon University Technical Report CS-93-13, 1976. Abstract in Traub JF (ed) Symposium on new directions and recent results in algorithms and complexity, pp 441. Academic, New York (1976)
2. Held M, Karp RM (1970) The traveling salesman problem and minimum spanning trees. Oper Res 18:1138–1162
3. Johnson DS, McGeoch LA (2002) Experimental analysis of heuristics for the STSP. In: Gutin G, Punnen AP (eds) The traveling salesman problem and its variations. Kluwer, Dordrecht
4. Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB (eds) (1985) The traveling salesman problem. A guided tour of combinatorial optimization. Wiley, Chichester
5. Papadimitriou C, Steiglitz K (1982) Combinatorial optimization: algorithms and complexity. Prentice-Hall, Englewood Cliffs
6. Sahni S, Gonzalez T (1976) P-complete approximation problems. J ACM 23:555–565
7. Vazirani VV (2001) Approximation algorithms. Springer, Berlin
8. Traveling Salesman Problem (2006). www.tsp.gatech.edu. Accessed 28 Mar 2008

Metrical Task Systems

Manor Mendel
Department of Mathematics and Computer Science, The Open University of Israel, Raanana, Israel

Keywords

MTS

Years and Authors of Summarized Original Work

1992; Borodin, Linial, Saks

Problem Definition

Metrical task systems (MTS), introduced by Borodin, Linial, and Saks [5], is a cost minimization problem defined on a metric space (X, d_X) and informally described as follows: A given *system* has a set of internal states X. The aim of the system is to serve a given sequence of tasks. The servicing of each task has a certain cost that depends on the task and the state of the system. The system may switch states before serving the task, and the total cost for servicing the task is the sum of the service cost of the task in the new state and the distance between

the states in a metric space defined on the set of states. Following Manasse, McGeoch, and Sleator [11], an extended model is considered here, in which the set of allowable tasks may be restricted.

Notation

Let T^* denote the set of finite sequences of elements from a set T. For $x, y \in T^*$, $x \circ y$ is the concatenation of the sequences x and y, and $|x|$ is the length of the sequence x.

Definition 1 (Metrical Task System) Fix a metric space (X, d_X). Let $\Gamma = \{(r_x)_{x \in X} : \forall x \in X, r(x) \in [0, \infty]\}$ be the set of all possible tasks. Let $T \subseteq \Gamma$ be a subset of tasks, called *allowable tasks*.
MTS$((X, d_X), T, a_0 \in X)$:
INPUT: A finite sequence of tasks $\tau = (\tau_1, \ldots, \tau_m) \in T^*$.
OUTPUT: A sequence of points $a = (a_1, \ldots, a_m) \in X^*$,
$|a| = |\tau|$.
OBJECTIVE: minimize

$$\mathsf{cost}(\tau, a) = \sum_{i=1}^{m} (d_X(a_{i-1}, a_i) + \tau_i(a_i)).$$

When $T = \Gamma$, the MTS problem is called *general*.

When X is finite and the task sequence $\tau \in T^*$ is given in advance, a dynamic programming algorithm can compute an optimal solution in space $O(|X|)$ and time $O(|\tau| \cdot |X|)$. MTS, however, is most interesting in an online setting, where the system must respond to a task τ_i with a state $a_i \in X$ without knowing the future tasks in τ. Formally,

Definition 2 (Online algorithms for MTS) A deterministic algorithm for a MTS$((X, d_X), T, a_0)$ is a mapping $S : T^* \to X^*$ such that for every $\tau \in T$, $|S(\tau)| = |\tau|$. A deterministic algorithm $S : T^* \to X^*$ is called *online* if for every $\tau, \sigma \in T^*$, there exists $a \in X^*$, $|a| = |\sigma|$ such that $S(\tau \circ \sigma) = S(\tau) \circ a$. A randomized online algorithm is a probability distribution over deterministic online algorithms.

Online algorithms for MTS are evaluated using *(asymptotic) competitive analysis*, which is, roughly speaking, the worst ratio of the algorithm's cost to the optimal cost taken over all possible task sequences.

Definition 3 A randomized online algorithm R for MTS$((X, d_X), a_0)$ is called c-competitive (against oblivious adversaries) if there exists $b = b(X) \in \mathbb{R}$ such that for any task sequence $\tau \in T^*$, and any point sequence $a \in X^*$, $|a| = |\tau|$,

$$\mathbb{E}[\mathsf{cost}(\tau, R(\tau))] \leq c \cdot \mathsf{cost}(\tau, a) + b,$$

where the expectation is taken over the distribution R.

The competitive ratio of an online algorithm R is the infimum over $c \geq 1$ for which R is c-competitive. The deterministic [respectively, randomized] competitive ratio of MTS$((X, d_X), T, a_0)$ is the infimum over the competitive ratios of all deterministic [respectively, randomized] online algorithms for this problem. Note that because of the existential quantifier over b, the asymptotic competitive ratio (both randomized and deterministic) of a MTS$((X, d_X), T, a_0)$ is independent of a_0, and it can therefore be dropped from the notation.

Key Results

Theorem 1 ([5]) *The deterministic competitive ratio of the general MTS problem on any n-point metric space is* $2n - 1$.

In contrast to the deterministic case, the understanding of randomized algorithms for general MTS is not complete, and generally no sharp bounds such as Theorem 1 are known.

Theorem 2 ([5, 10]) *The randomized competitive ratio of the general MTS problem on n-point uniform space (where all distances are equal) is at least* $H_n = \sum_{i=1}^{n-1} i^{-1}$, *and at most* $(1 + o(1))H_n$.

The best bounds currently known for general n-point metrics are proved in two steps: First the given metric is approximated by an *ultrametric*, and then a bound on the competitive ratio of general MTS on ultrametrics is proved.

Theorem 3 ([8, 9]) *For any n-point metric space (X, d_X), there exists an $O(\log^2 n \log\log n)$ competitive randomized algorithm for the general MTS on (X, d_X).*

The metric approximation component in the proof of Theorem 3 is called *probabilistic embedding*. An optimal $O(\log n)$ probabilistic embedding is shown by Fakcheroenphol, Rao and Talwar before [8] improving on results by Alon, Karp, Peleg, and West and by Bartal, where this notion was invented. A different type of metric approximation with better bounds for metrics of low *aspect ratio* is given in [3].

Fiat and Mendel [9] show a $O(\log n \log\log n)$ competitive algorithm for n-point ultrametrics, improving (and using) a result of Bartal, Blum, Burch, and Tomkins [1], where the first polylogarithmic (or even sublinear) competitive randomized algorithm for general MTS on general metric spaces is presented.

Theorem 4 ([2, 12]) *For any n-point metric space (X, d_X), the randomized competitive ratio of the general MTS on (X, d_X) is at least $\Omega(\log n / \log\log n)$.*

The metric approximation component in the proof of Theorem 4 is called *Ramsey subsets*. It was first used in this context by Karloff, Rabani, and Ravid, later improved by Blum, Karloff, Rabani and Saks, and Bartal, Bollobás, and Mendel [2]. A tight result on Ramsey subsets is proved by Bartal, Linial, Mendel, and Naor. For a simpler (and stronger) proof, see [12].

A lower bound of $\Omega(\log n / \log\log n)$ on the competitive ratio of any randomized algorithm for general MTS on n-point ultrametrics is proved in [2], improving previous results of Karloff, Rabani, and Ravid, and Blum, Karloff, Rabani and Saks.

The last theorem is the only one not concerning general MTSs.

Theorem 5 ([6]) *It is PSPACE hard to determine the competitive ratio of a given MTS instance $((X, d_X), a_0 \in X, T)$, even when d_X is the uniform metric. On the other hand, when d_X is uniform, there is a polynomial time deterministic online algorithm for MTS $((X, d_X), a_0 \in X, T)$ whose competitive ratio is $O(\log |X|)$ times the deterministic competitive ratio of the MTS((X, d_X), a_0, T). Here it is assumed that the instance ((X, d_X), a_0, T) is given explicitly.*

Applications

Metrical task systems were introduced as an abstraction for online computation, they generalize many concrete online problems such as paging, weighted caching, k-server, and list update. Historically, it served as an indicator for a general theory of competitive online computation.

The main technical contribution of the MTS model is the development of the work function algorithm used to prove the upper bound in Theorem 1. This algorithm was later analyzed by Koutsoupias and Papadimitriou in the context of the k-server problem, and was shown to be $2k - 1$ competitive. Furthermore, although the MTS model generalizes the k-server problem, the general MTS problem on the n-point metric is essentially equivalent to the $(n - 1)$-server problem on the same metric [2]. Hence, lower bounds on the competitive ratio of general MTS imply lower bounds for the k-server problem, and algorithms for general MTS may constitute a first step in devising an algorithm for the k-server problem, as is the case with the work function algorithm.

The metric approximations used in Theorem 3, and Theorem 4 have found other algorithmic applications.

Open Problems

There is still an obvious gap between the upper bound and lower bound known on the randomized competitive ratio of general MTS on general finite metrics. It is known that, contrary to the deterministic case, the randomized competitive

ratio is *not* constant across all metric spaces of the same size. However, in those cases where exact bounds are known, the competitive ratio is $\Theta(\log n)$. An obvious conjecture is that the randomized competitive is $\Theta(\log n)$ for any n-point metric. Arguably, the simplest classes of metric spaces for which no upper bound on the randomized competitive ratio better than $O(\log^2 n)$ is known, are paths and cycles.

Also lacking is a "middle theory" for MTS. On the one hand, general MTS are understood fairly well. On the other hand, specialized MTS such as list update, deterministic k-server algorithms, and deterministic weighted-caching, are also understood fairly well, and have a much better competitive ratio than the corresponding general MTS. What may be missing are "in between" models of MTS that can explain the low competitive ratios for some of the concrete online problems mentioned above.

It would be also nice to strengthen Theorem 5, and obtain a polynomial time deterministic online algorithm whose competitive ratio on any MTS instance on *any* n-point metric space is at most poly-log(n) times the deterministic competitive ratio of that MTS instance.

Cross-References

- Algorithm DC-TREE for k-Servers on Trees
- Approximating Metric Spaces by Tree Metrics
- Online List Update
- Online Paging and Caching
- Ski Rental Problem
- Work-Function Algorithm for k-Servers

Recommended Reading

1. Bartal Y, Blum A, Burch C, Tomkins A (1997) A polylog(n)-competitive algorithm for metrical task systems. In: Proceedings of the 29th annual ACM symposium on the theory of computing. ACM, New York, pp 711–719
2. Bartal Y, Bollobás B, Mendel M (2006) Ramsey-type theorems for metric spaces with applications to online problems. J Comput Syst Sci 72:890–921
3. Bartal Y, Mendel M (2004) Multiembedding of metric spaces. SIAM J Comput 34:248–259
4. Borodin A, El-Yaniv R (1998) Online computation and competitive analysis. Cambridge University Press, Cambridge, UK
5. Borodin A, Linial N, Saks ME (1992) An optimal on-line algorithm for metrical task system. J ACM 39:745–763
6. Burley WR, Irani S (1997) On algorithm design for metrical task systems. Algorithmica 18:461–485
7. Chrobak M, Larmore LL (1998) Chapter 4, Metrical task systems, the server problem and the work function algorithm. In: Fiat A, Woeginger GJ (eds) Online algorithms. The state of the art. LNCS, vol 1442. Springer, London, pp 74–96
8. Fakcharoenphol J, Rao S, Talwar K (2004) A tight bound on approximating arbitrary metrics by tree metrics. J Comput Syst Sci 69:485–497
9. Fiat A, Mendel M (2003) Better algorithms for unfair metrical task systems and applications. SIAM J Comput 32:1403–1422
10. Irani S, Seiden SS (1998) Randomized algorithms for metrical task systems. Theor Comput Sci 194:163–182
11. Manasse MS, McGeoch LA, Sleator DD (1990) Competitive algorithms for server problems. J Algorithms 11:208–230
12. Mendel M, Naor A (2007) Ramsey partitions and proximity data structures. J Eur Math Soc 9(2):253–275

Min-Hash Sketches

Edith Cohen
Tel Aviv University, Tel Aviv, Israel
Stanford University, Stanford, CA, USA

Keywords

Approximate distinct counting; Bottom-k sketches; k-mins sketches; k-partition sketches; Similarity estimation; Summary structures

Years and Authors of Summarized Original Work

1985; Flajolet, Martin
1997; Broder
1997; Cohen

Problem Definition

MINHASH sketches (also known as min-wise sketches) are randomized summary structures of subsets which support set union operations and approximate processing of cardinality and similarity queries.

Set-union support, also called *mergeability*, means that a sketch of the union of two sets can be computed from the sketches of the two sets. In particular, this applies when the second set is a single element. The queries supported by MINHASH sketches include cardinality (of a subset from its sketch) and similarity (of two subsets from their sketches).

Sketches are useful for massive data analysis. Working with sketches often means that instead of explicitly maintaining and manipulating very large subsets (or equivalently 0/1 vectors), we can instead maintain the much smaller sketches and can still query properties of these subsets.

We denote the universe of elements by U and its size by $n = |U|$. We denote by $S(X)$ the sketch of the subset $X \subset U$.

Set Operations

- **Inserting an element:** Given a set X and element $y \in U$, a sketch $S(X \cup \{y\})$ can be computed from $S(X)$ and y.
- **Merging two sets:** For two (possibly overlapping) sets X and Y, we can obtain a sketch of their union $S(X \cup Y)$ from $S(X)$ and $S(Y)$.

Support for insertion makes the sketches suitable for streaming, where elements (potentially with repeated occurrences) are introduced sequentially. Support for merges is important for parallel or distributed processing: We can sketch a data set that has multiple parts by sketching each part and combining the sketches. We can also compute the sketches by partitioning the data into parts, sketching each of the parts concurrently, and finally merging the sketches of the parts to obtain a sketch of the full data set.

Queries

From the sketches of subsets, we would like to (approximately) answer queries on the original data. More precisely, for a set pf subsets $\{X_i\}$, we are interested in estimating a function $f(\{X_i\})$. To do this, we apply an *estimator* $\hat{f}$ to the respective set of sketches $\{S(X_i)\}$.

We would like our estimators to have certain properties: When estimating nonnegative quantities (such as cardinalities or similarities), we would want the estimator to be nonnegative as well. We are often interested in unbiased estimators and always in *admissible* estimators, which are Pareto optimal in terms of variance (variance on one instance cannot be improved without increasing variance on another). We also seek good *concentration*, meaning that the probability of error decreases exponentially with the relative error.

We list some very useful queries that are supported by MINHASH sketches:

- **Cardinality:** The number of elements in the set $f(X) = |X|$.
- **Similarity:** The Jaccard coefficient $f(X, Y) = |X \cap Y|/|X \cup Y|$, cosine similarity $f(X, Y) = |X \cap Y|/\sqrt{|X||Y|}$, or cardinality of the union $f(X, Y) = |X \cup Y|$.
- **Complex relations:** Cardinality of the union of multiple sets $|\bigcup_i X_i|$, number of elements occurring in at least 2 sets $|\{j \mid \exists i_1 \neq i_2,\ j \in X_{i_1} \cap X_{i_2}\}|$, set differences, etc.
- **Domain queries:** When elements have associated metadata (age, topic, activity level), we can include this information in the sketch, which becomes a random sample of the set. Including this information allows us to process domain queries, which depend on the matadata. For example, "the number of Californians in the union of two (or more) sets."

Key Results

MINHASH sketches had been proposed as summary structures which satisfy the above requirements. There are multiple variants of the MINHASH sketch, which are optimized for different applications. The common thread is that the elements $x \in U$ of the universe U are assigned

random *rank* values $r(x)$ (which are typically produced by a random hash function). The MinHash sketch $S(X)$ of a set X includes order statistics (maximum, minimum, or top-/bottom-k values) of the set of ranks $\{r(x) \mid x \in X\}$. Note that when we sketch multiple sets, the same random rank assignment is common to all sketches (we refer to this as *coordination*).

Before stating precise definitions for the different MinHash sketch structures, we provide some intuition for the power of order statistics. We first consider cardinality estimation. The minimum rank value $\min_{x \in X} r(x)$ is the minimum of $|X|$ independent random variables, and therefore, its expectation should be smaller when the cardinality $|X|$ is larger. Thus, the minimum rank carries information on $|X|$. We next consider the sketches of two sets X and Y. Recall that they are computed with respect to the same assignment r. Therefore, the minimum rank values carry information on the similarity of the sets: in particular, when the sets are more similar, their minimum ranks are more likely to be equal. Finally, the minimum rank element of a set is a random sample from the set and, therefore, as such, can support estimation of statistics of the set.

The variations of MinHash sketches differ in the particular structure: how the rank assignment is used and the domain and distribution of the ranks $r(x) \sim D$.

Structure

MinHash sketches are parameterized by an integer $k \geq 1$, which controls a trade-off between the size of the sketch representation and the accuracy of approximate query results.

MinHash sketches come in the following three common flavors:

- A *k-mins sketch* [6, 13] includes the smallest rank in each of k independent rank assignments. There are k different rank functions r_i and the sketch $S(X) = (\tau_1, \ldots, \tau_k)$ has $\tau_i = \min_{y \in X} r_i(y)$. When viewed as a sample, it corresponds to sampling k times with replacement.
- A *k-partition sketch* [13, 14, 18], which in the context of cardinality estimation is called *stochastic averaging*, uses a single rank assignment together with a uniform at random mapping of items to k buckets. We use $b : U \to [k]$ for the bucket mapping and r for the rank assignment. The sketch $(\tau_1, \ldots, \tau_k)$ then includes the item with minimum rank in each bucket. That is, $\tau_i = \min_{y \in X | b(y)=i} r_i(y)$. If the set is empty, the entry is typically defined as the supremum of the domain of r.
- A *bottom-k sketch* [4, 6] $\tau_1 < \cdots < \tau_k$ includes the k items with smallest rank in $\{r(y) \mid y \in X\}$. Interpreted as a sample, it corresponds to sampling k elements without replacement. Related uses of the same method include KMV sketch [2], coordinated order samples [3, 19, 21], or conditional random sampling [17].

Note that all three flavors are the same when $k = 1$.

With all three flavors, the sketch represents k random elements of D. When viewed as random samples, MinHash sketches of different subsets X are *coordinated*, since they are generated using the same random rank assignments to the domain U. The notion of coordination is very powerful. It means that similar subsets have similar sketches (a locality sensitive hashing property). It also allows us to support merges and similarity estimation much more effectively. Coordination in the context of survey sampling was introduced in [3] and was applied for sketching data in [4, 6].

Rank Distribution

Since we typically use a random hash function $H(x) \sim D$ to generate the ranks, it always suffices to store element identifiers instead of ranks, which means the representation of each rank value is $\lceil \log_2 n \rceil$ bits and the bit size of the sketch is at most $k \log n$. This representation size is necessary when we want to support domain queries – the sketch of each set should identify the element associated with each included rank, so we can retrieve the metadata needed to evaluate a selection predicate.

For the applications of estimating cardinalities or pairwise similarities, however, we can work with ranks that are not unique to elements and, in particular, come from a smaller discrete domain. Working with smaller ranks allows us to use sketches of a much smaller size and also replace the dependence of the sketch on the domain size ($O(\log n)$ per entry) by dependence on the subset sizes. In particular, we can support cardinality estimation and similarity estimation of subsets of size at most m with ranks of size $O(\log \log m)$. Since the k rank values used in the sketch are typically highly correlated, the sketch $S(X)$ can be stored using $O(\log \log m + k)$ bits in expectation ($O(\log \log m + k \log k)$ bits for similarity). This is useful when we maintain sketches of many sets and memory is at a premium, as when collecting traffic statistics in IP routers.

For analysis, it is often convenient to work with continuous ranks, which without loss of generality are $r \sim U[0, 1]$ [6], since there is a monotone (order preserving) transformation from any other continuous distribution. Using ranks of size $O(\log n)$ is equivalent to working with continuous ranks.

In practice, we work with discrete ranks, for example, values restricted to $1/2^i$ for integral $i > 0$ [13] or more generally using a *base* $b > 1$ and using $1/b^i$. This is equivalent to drawing a continuous rank and rounding it down to the largest discrete point of the form $1/b^i$.

Streaming: Number of Updates

Consider now maintaining a MinHash sketch in the streaming model. We maintain a sketch S of the elements X that we had seen until now. When we process a new element y, then if $y \in X$, the sketch is not modified. We can show that the number of times the sketch is modified is in expectation at most $k \ln n$, where $n = |X|$ is the number of distinct elements in the prefix. We provide the argument for bottom-k sketches. It is similar with other flavors. The probability that a new element has a rank value that is smaller than the kth smallest rank in X is the probability that it is in one of the first k positions in a permutation of size $n + 1$. That is, the probability is 1 if $n < k$ and is $k/(n + 1$ otherwise. Summing over new distinct elements $n = 1, \ldots, |X|$, we obtain $\sum_{i=1}^{n} k/i \leq k \ln n$.

Inserting an Element

We now consider inserting an element y, that is, obtaining a sketch $S(X \cup \{y\})$ from $S(X)$ and y. The three sketch flavors have different properties and trade-offs in terms of insertion costs. We distinguish between insertions that result in an actual update of the sketch and insertions where the sketch is not modified.

- k-mins sketch: We need to generate the rank of y, $r_i(y)$, in each of k different assignments (k hash computations). We can then compare, coordinate-wise, each rank with the respective one in the sketch, taking the minimum of the two values. This means that each insertion, whether the sketch is updated or not, results in $O(k)$ operations.
- Bottom-k sketch: We apply our hash function to generate $r(y)$. We then compare $r(y)$ with τ_k. If the sketch contains fewer than k ranks ($|S| < k$ or τ_k is the rank domain supremum), then $r(y)$ is inserted to S.

  Otherwise, the sketch is updated only if $r(y) < \tau_k$. In this case, the largest sketch entry τ_k is discarded and $r(y)$ is inserted to the sketch S. When the sketch is not modified, the operation is $O(1)$. Otherwise, it can be $O(\log k)$.
- k-partition sketch: We apply the hash functions to y to determine the bucket $b(y) \in [k]$ and the rank $r(y)$. To determine if an update is needed, we compare $r(y)$ and $\tau_{b(y)}$. If the latter is empty ($\tau_{b(y)}$ is the domain supremum) or if $r(y) < \tau_{b(y)}$, we assign $\tau_{b(y)} \leftarrow r(y)$.

Merging

We now consider computing the sketch $S(X \cup Y)$ from the sketches $S(X)$ and $S(Y)$ of two sets X, Y.

For k-mins and k-partition sketches, the sketch of $S(X \cup Y)$ is simply the coordinate-wise minimum $(\min\{\tau_1, \tau_1'\}, \ldots, \min\{\tau_k, \tau_k'\})$

of the sketches $S(X) = (\tau_1, \ldots, \tau_k)$ and $S(Y) = (\tau'_1, \ldots, \tau'_k)$. For bottom-$k$ sketches, the sketch of $S(X \cup Y)$ includes the k smallest rank values in $S(X) \cup S(Y)$.

Estimators

Estimators are typically specifically derived for a given sketch flavor and rank distribution.

Cardinality estimators were pioneered by Flajolet and Martin [13], continuous ranks were considered in [6], and lower bounds were presented in [1]. State-of-the-art practical solutions include [7, 14]. Cardinality estimation can be viewed in the context of the theory of point estimation: estimating the parameter of a distribution (the cardinality) from the sketch (the "outcome"). Estimation theory implies that current estimators are optimal (minimize variance) for the sketch [7]. Recently, *historic inverse probability (HIP)* estimators were proposed, which apply with all sketch types and improve variance by maintaining an approximate count alongside the MinHash sketch [7], which is updated when the sketch is modified.

Estimators for set relations were first considered in [6] (cardinality of union, by computing a sketch of the union and applying a cardinality estimator) and [4] (the Jaccard coefficient, which is the ratio of intersection to union size). The Jaccard coefficient can be estimated on all sketch flavors (when ranks are not likely to have collisions) by simply looking at the respective ratio in the sketches themselves. In general, many set relations can be estimated from the sketches, and state-of-the-art derivation is given in [8, 10].

Applications

Approximate distinct counters are widely used in practice. Applications include statistics collection at IP routers and counting distinct search queries [15].

An important application of sketches, and their first application to estimate set relations, was introduced in [6]. Given a directed graph, and a node v, we can consider the set $R(v)$ of all reachable nodes. It turns out that sketches $S(R(v))$ for all nodes v in a graph can be computed very efficiently, in nearly linear time. The approach naturally extends to sketching neighborhoods in a graph. The sketches of nodes support efficient estimation of important graph properties, such as the distance distribution, node similarities (compare their relations to other nodes), and influence of a set of nodes [11, 12].

Similarity estimation using sketches was applied to identify near-duplicate Web pages by sketching "shingles" that are consecutive lists or words [4]. Since then, MinHash sketches are extensively applied for similarity estimation of text documents and other entities.

Extensions

Weighted Elements

MinHash sketches are summaries of sets or of 0/1 vectors. In many applications, each element $x \in U$ has a different intrinsic nonnegative weight $w(x) > 0$, and queries are formulated with respect to these weights: Instead of cardinality estimates we can consider the respective weighted sum $\sum_{x \in X} w(x)$. Instead of the Jaccard for the similarity of two sets X and Y, we may be interested in the weighted version $\sum_{x \in X \cap Y} w(x) / \sum_{x \in X \cup Y} w(x)$. When this is the case, to obtain more accurate query results, we use sketches so that the inclusion probability of an element increases with its weight. The sketch in this case would correspond to a weighted sample. This is implemented by using ranks which are drawn from a distribution that depends on the weights $w(y)$ [6, 9, 19–21].

Hash Functions

We assumed here the availability of truly random hash function. In practice, observed performance is consistent with this assumption. We mention however that the amount of independence needed was formally studied using *min-wise independent* hash functions [5, 16].

Cross-References

- ▸ All-Distances Sketches of graphs.
- ▸ Coordinated Sampling. They are also a building block of
- ▸ MIN-HASH Sketches can be viewed as

Recommended Reading

1. Alon N, Matias Y, Szegedy M (1999) The space complexity of approximating the frequency moments. J Comput Syst Sci 58:137–147
2. Bar-Yossef Z, Jayram TS, Kumar R, Sivakumar D, Trevisan L (2002) Counting distinct elements in a data stream. In: RANDOM, Cambridge. ACM
3. Brewer KRW, Early LJ, Joyce SF (1972) Selecting several samples from a single population. Aust J Stat 14(3):231–239
4. Broder AZ (1997) On the resemblance and containment of documents. In: Proceedings of the compression and complexity of sequences, Salerno. IEEE, pp 21–29
5. Broder AZ, Charikar M, Frieze AM, Mitzenmacher M (2000) Min-wise independent permutations. J Comput Syst Sci 60(3):630–659
6. Cohen E (1997) Size-estimation framework with applications to transitive closure and reachability. J Comput Syst Sci 55:441–453
7. Cohen E (2014) All-distances sketches, revisited: HIP estimators for massive graphs analysis. In: PODS, Snowbird. ACM. http://arxiv.org/abs/1306.3284
8. Cohen E (2014) Estimation for monotone sampling: competitiveness and customization. In: PODC, Paris. ACM. http://arxiv.org/abs/1212.0243, full version http://arxiv.org/abs/1212.0243
9. Cohen E, Kaplan H (2007) Summarizing data using bottom-k sketches. In: PODC, Portland. ACM
10. Cohen E, Kaplan H (2009) Leveraging discarded samples for tighter estimation of multiple-set aggregates. In: SIGMETRICS, Seattle. ACM
11. Cohen E, Delling D, Fuchs F, Goldberg A, Goldszmidt M, Werneck R (2013) Scalable similarity estimation in social networks: closeness, node labels, and random edge lengths. In: COSN, Boston. ACM
12. Cohen E, Delling D, Pajor T, Werneck RF (2014) Sketch-based influence maximization and computation: scaling up with guarantees. In: CIKM. ACM. http://research.microsoft.com/apps/pubs/?id=226623, full version http://research.microsoft.com/apps/pubs/?id=226623
13. Flajolet P, Martin GN (1985) Probabilistic counting algorithms for data base applications. J Comput Syst Sci 31:182–209
14. Flajolet P, Fusy E, Gandouet O, Meunier F (2007) Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In: Analysis of algorithms (AOFA), Juan des Pins
15. Heule S, Nunkesser M, Hall A (2013) HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. In: EDBT, Genoa
16. Indyk P (1999) A small approximately min-wise independent family of hash functions. In: Proceedings of the 10th ACM-SIAM symposium on discrete algorithms, Baltimore. ACM-SIAM
17. Li P, Church KW, Hastie T (2008) One sketch for all: theory and application of conditional random sampling. In: NIPS, Vancouver
18. Li P, Owen AB, Zhang CH (2012) One permutation hashing. In: NIPS, Lake Tahoe
19. Ohlsson E (1998) Sequential poisson sampling. J Off Stat 14(2):149–162
20. Rosén B (1972) Asymptotic theory for successive sampling with varying probabilities without replacement, I. Ann Math Stat 43(2):373–397. http://www.jstor.org/stable/2239977
21. Rosén B (1997) Asymptotic theory for order sampling. J Stat Plan Inference 62(2):135–158

Minimal Dominating Set Enumeration

Mamadou Moustapha Kanté and
Lhouari Nourine
Clermont-Université, Université Blaise Pascal, LIMOS, CNRS, Aubière, France

Keywords

Dominating set; Enumeration; Transversal hypergraph

Years and Authors of Summarized Original Work

2011–2014; Kanté, Limouzy, Mary, Nourine, Uno

Problem Definition

Let G be a graph on n vertices and m edges. An edge is written xy (equivalently yx). A *dominating set* in G is a set of vertices D

such that every vertex of G is either in D or is adjacent to some vertex of D. It is said to be *minimal* if it does not contain any other dominating set as a proper subset. For every vertex x, let $N[x]$ be $\{x\} \cup \{y | xy \in E\}$ and for every $S \subseteq V$ let $N[S] := \bigcup_{x \in S} N[x]$. For $S \subseteq V$ and $x \in S$ we call any $y \in N[x] \backslash N[S \backslash x]$, a *private neighbor of x with respect to S*. The set of minimal dominating sets of G is denoted by $\mathcal{D}(G)$. We are interested in an output-polynomial algorithm for enumerating $\mathcal{D}(G)$, i.e., listing, without repetitions, all the elements of $\mathcal{D}(G)$ in time bounded by $p\left(n+m, \sum_{D \in \mathcal{D}(G)} |D|\right)$ (DOM-ENUM for short).

It is easy to see that DOM-ENUM is a special case of HYPERGRAPH DUALIZATION. Let $\mathcal{N}(G)$, called the *closed neighborhood hypergraph*, be the hypergraph with hyperedges $\{N[x] | x \in V\}$. It is easy to see that D is a dominating set of G if and only if D is a transversal of $\mathcal{N}(G)$. Hence, DOM-ENUM is a special case of HY-PERGRAPH DUALIZATION. For several graph classes their closed neighborhood hypergraphs are subclasses of hypergraph classes where an output-polynomial algorithm is known for HYPERGRAPH DUALIZATION, e.g., minor-closed classes of graphs, graphs of bounded degree, graphs of bounded conformality, graphs of bounded degeneracy, graphs of logarithmic degeneracy [11, 12, 19]. So, DOM-ENUM seems more tractable than HYPERGRAPH DUALIZATION since there exist families of hypergraphs that are not closed neighborhoods of graphs [1].

Key Results

Contrary to several special cases of HYPERGRAPH DUALIZATION in graphs, (e.g., enumeration of maximal independent sets, enumeration of spanning forests, etc.) DOM-ENUM is equivalent to HYPERGRAPH DUALIZATION. Indeed, it is proved in [14] that with every hypergraph $\mathcal{H}$, one can associate a co-bipartite graph $\mathcal{B}(\mathcal{H})$ such that every minimal dominating set of $\mathcal{B}(\mathcal{H})$ is either a transversal of $\mathcal{H}$ or has size at most 2. A consequence is that there exists a polynomial delay polynomial space algorithm for HYPERGRAPH DUALIZATION if and only if there exists one for DOM-ENUM, even in co-bipartite graphs. The reduction is moreover asymptotically tight (with respect to polynomial delay reductions as defined in [19]) in the sense that there exist hypergraphs $\mathcal{H}$ such that for every graph G we cannot have $tr(\mathcal{H}) = \mathcal{D}(G)$ [14]. This intriguing result has the advantage of bringing tools from graph structural theory to tackle the difficult and widely open problem HYPERGRAPH DUALIZATION. Furthermore, until recently the most graph classes where DOM-ENUM is known to be tractable were those for which closed neighborhood hypergraphs were subclasses of some of the tractable hypergraph classes for HYPERGRAPH DUALIZATION. We will give examples of graph classes where graph theory helps a lot to solve DOM-ENUM, and sometimes allows to introduce new techniques for the enumeration.

It is widely known now that every monadic second-order formula can be checked in polynomial time in graph classes of bounded *clique-width* [3,20]. Courcelle proved in [2] that one can also enumerate, with linear delay linear space, the solutions of every monadic second-order formula. Since one can express in monadic second-order logic that a subset D of vertices is a minimal dominating set, DOM-ENUM has a linear delay linear space in graph classes of bounded clique-width. The algorithm by Courcelle is quite ingenious: it firsts constructs a DAG, some subtrees of which correspond to the positive runs of the tree-automata associated with the formula on the given graph and then enumerate these subtrees.

Many graph classes do not have bounded clique-width (interval graphs, permutation graphs, unit-disk graphs, etc.) and many such graph classes have nice structures that helped in the past for solving combinatorial problems, e.g., the clique-tree of chordal graphs, permutation models, etc. For some of these graph classes structural results can help to solve DOM-ENUM.

A common tool in enumeration area is the *parsimonious reduction*. One wants to enumerate a set of objects $\mathcal{O}$ and instead constructs a bijective function $b : \mathcal{O} \to \mathcal{T}$ such that there is an efficient algorithm to enumerate $\mathcal{T}$. For instance it is proved in [11, 14] that every minimal dominating set D of a split graph G can be characterized by $D \cap C(G)$ where $C(G)$ is the clique of G. A consequence is that in a split graph G there is a bijection between $\mathcal{D}(G)$ and the set $\{S \subseteq C(G) | \forall x \in S, x$ has a private neighbor$\}$, and since this latter set is an independent system, DOM-ENUM in split graphs admits a linear delay polynomial space algorithm.

One can obtain other parsimonious reductions using graph structures. For instance, it is easy to check that every minimal dominating set in an interval graph is a collection of paths. Moreover, using the interval model (and ordering intervals from their left endpoints) every minimal dominating set can be constructed greedily by keeping track of the last two chosen vertices. Indeed it is proved in [13] that with every interval graph G one can associate a DAG, the maximal paths of which are in bijection with the minimal dominating sets of G. The nodes of the DAG are pairs (x, y) such that $x < y$ and such that x and y can be both in a minimal dominating set, and the arcs are $((x, y), (y, z))$ such that (1) $\{x, y, z\}$ can be in a minimal dominating set, (2) there is no vertex between y and z that is not dominated by y or z, sources are pairs (x, y) where every interval before x is dominated by x, and sinks are pairs (x, y) where every interval after y is dominated by y. This reduction to maximal paths of a DAG can be adapted to several other graph classes having a linear structure similar to the interval model, e.g. permutation graphs, circular-arc graphs [13]. In general, if for every graph G in a graph class $\mathcal{C}$ one can associate an ordering of the vertices such that for every subset $S \subseteq V$ the possible ways to extend S into a minimal dominating set depends only on the last k vertices of S, for some fixed constant k depending only on $\mathcal{C}$, then for every $G \in \mathcal{C}$ the enumeration of $\mathcal{D}(G)$ can be reduced to the enumeration of paths in a DAG as for interval graphs and thus DOM-ENUM is tractable in $\mathcal{C}$ [19]. This seems for instance to be the case for d-trapezoid graphs.

Parsimonious reductions between graph classes can be also defined. For instance, the *completion* of a graph G, i.e., the set of edges that can be added to G without changing $\mathcal{D}(G)$ are characterized in [11, 14], this characterization lead the authors to prove that the completion of every P_6-free chordal graph is a split graph, which results in a linear delay polynomial space algorithm for DOM-ENUM in P_6-free chordal graphs.

The techniques developed by the HYPERGRAPH DUALIZATION community combined with graph structural theory can give rise to new tractable cases of DOM-ENUM. For instance, the main drawback of Berge's algorithm is that at some level computed transversals are not necessarily subsets of solutions and this prevents from obtaining an output-polynomial algorithm since the computed set may be arbitrary large compared to the solution set [21]. One way to overcome this difficulty consists in choosing some levels $l_1 \ldots, l_k$ of Berge's algorithm such that every computed set at level l_j is a subset of a solution at level l_{j+1}. A difficulty with that scheme is to compute all the descendants in level l_{j+1} of a transversal in level l_j. This idea combined with the structure of minimal dominating sets in line graphs is used to derive a polynomial delay polynomial space algorithm for DOM-ENUM in line graphs [15]. A consequence is that there is a polynomial delay polynomial space algorithm to list the set of *minimal edge dominating sets* in graphs.

Another famous technique in enumeration area is the *back tracking*. Start from the empty set, and in each iteration choose a vertex x and partition the problem into two sub-problems: the enumeration of minimal dominating sets containing x and the enumeration of those not containing x, at each step we have a set X to include in the solution and a set Y not to include. If at each step one can solve the EXTENSION PROBLEM, i.e., whether there is a minimal dominating set containing X and not intersecting Y, then DOM-ENUM admits a polynomial delay polynomial space

algorithm. However, the EXTENSION PROBLEM is NP-complete in general [19] and even in split graphs [16]. But, sometimes structure helps. For instance, in split graphs whenever $X \cup Y \subseteq C(G)$, the EXTENSION PROBLEM is polynomial [11, 14] and was the key for the linear delay algorithm. Another special case of the EXTENSION PROBLEM is proved to be polynomial in chordal graphs using the *clique tree* of chordal graphs and is also the key to prove that DOM-ENUM in chordal graphs admits a polynomial delay polynomial space algorithm [16]. The algorithm uses deeply the clique tree and is a nested combination of several enumeration algorithms.

Open Problems

1. The first major challenge is to find an output-polynomial algorithm for DOM-ENUM, even in co-bipartite graphs. One way to address this problem is to understand the structure of minimal dominating sets in a graph. Failing to solve this problem, can graphs help to improve the quasi-polynomial time algorithm by Fredman and Khachiyan [7]?
2. Until now if the techniques used to solve DOM-ENUM in many graph classes are well-known, deep structural theory of graphs is not used and the used graph structures are more or less ad hoc. Can we unify all these results and obtain at the same time new positive results? Indeed, there are several well-studied graph classes where the status of DOM-ENUM is still open: bipartite graphs, unit-disk graphs, graphs of bounded expansion to cite a few. Are developed tools sufficient to address these graph classes?
3. There are several well-studied variants of the dominating set problem, in particular *total dominating set* and *connected dominating set* (see the monographs [9, 10]). It is proved in [14] that the enumeration of minimal total dominating sets and minimal connected dominating sets in split graphs is equivalent to HYPERGRAPH DUALIZATION. This is somehow surprising and we do not yet understand why such small variations make the problem difficult even in split graphs. Can we explain this situation?
4. From [14] we know that the enumeration of minimal connected dominating sets is harder than HYPERGRAPH DUALIZATION. Are both problems equivalent? Can we find a graph class $\mathcal{C}$ where each graph in $\mathcal{C}$ has a non-exponential number of minimal connected dominating sets, but minimum connected dominating set is NP-complete? Notice that if a class of graphs $\mathcal{C}$ has a polynomially bounded number of minimal separators, then the enumeration of minimal connected dominating sets can be reduced to DOM-ENUM [14].
5. A related question to DOM-ENUM is a tight bound for the number of minimal dominating sets in graphs. The best upper bound is $O(1.7159^n)$ and the best lower bound is $15^{n/6}$ [6]. For several graph classes, tight bounds were obtained [4, 8]. Prove that $15^{n/6}$ is the upper bound or find the tight bound.
6. Another related subject to DOM-ENUM is the counting of (minimal) dominating sets in time polynomial in the input graph. If the counting of dominating sets is a #P-hard problem and have been investigated in the past [5, 17, 18], not so much is known for the counting of minimal dominating sets, one can cite few examples: graphs of bounded clique-width [2], and interval, permutation and circular-arc graphs [13]. If we define for G the *minimal domination polynomial* $MD(G, x)$ that is the generating function of its minimal dominating sets, for which graph classes this polynomial can be computed? Does it have a (linear) recursive definition? For which values x can we evaluate it?

Cross-References

- ▸ Beyond Hypergraph Dualization
- ▸ Efficient Polynomial Time Approximation Scheme for Scheduling Jobs on Uniform Processors
- ▸ Reverse Search; Enumeration Algorithms

Recommended Reading

1. Brandstädt A, Van Le B, Spinrad JP (1999) Graph classes a survey. SIAM monographs on discrete mathematics and applications. SIAM, Philadelphia
2. Courcelle B (2009) Lineai delay enumeration and monadic second-order logic. Discret Appl Math 157:2675–2700
3. Courcelle B, Makowsky JA, Rotics U (2000) Linear time solvable optimization problems on graphs of bounded clique-width. Theory Comput Syst 33(2):125–150
4. Couturier J-F, Heggernes P, van 't Hof P, Kratsch D (2013) Minimal dominating sets in graph classes: combinatorial bounds and enumeration. Theor Comput Sci 487:82–94
5. Dohmen K, Tittmann P (2012) Domination reliability. Electron J Comb 19(1):P15
6. Fomin FV, Grandoni F, Pyatkin AV, Stepanov AA (2008) Combinatorial bounds via measure and conquer: bounding minimal dominating sets and applications. ACM Trans Algorithms 5(1)
7. Fredman ML, Khachiyan L (1996) On the complexity of dualization of monotone disjunctive normal forms. J Algorithms 21(3):618–628
8. Golovach PA, Heggernes P, Kanté MM, Kratsch D, Villanger Y (2014) Minimal dominating sets in interval graphs and trees. Submitted
9. Haynes TW, Hedetniemi ST, Slater PJ (1998) Fundamentals of domination in graphs. Volume 208 of pure and applied mathematics. Marcel Dekker, New York
10. Haynes TW, Hedetniemi ST, Slaterv PJ (1998) Domination in graphs: advanced topics. Volume 209 of pure and applied mathematics. Marcel Dekker, New York
11. Kanté MM, Limouzy V, Mary A, Nourine L (2011) Enumeration of minimal dominating sets and variants. In: FCT 2011, Oslo, pp 298–309
12. Kanté MM, Limouzy V, Mary A, Nourine L (2012) On the neighbourhood helly of some graph classes and applications to the enumeration of minimal dominating sets. In: ISAAC 2012, Taipei, pp 289–298
13. Kanté MM, Limouzy V, Mary A, Nourine L, Uno T (2013) On the enumeration and counting of minimal dominating sets in interval and permutation graphs. In: ISAAC 2013, Hong Kong, pp 339–349
14. Kanté MM, Limouzy V, Mary A, Nourine L (2014) On the enumeration of minimal dominating sets and related notions. Accepted for publication at SIAM Journal on Discrete Mathematics
15. Kanté MM, Limouzy V, Mary A, Nourine L, Uno T (2014) Polynomial delay algorithm for listing minimal edge dominating sets in graphs. In: CoRR. abs/1404.3501
16. Kanté MM, Limouzy V, Mary A, Nourine L, Uno T (2014) A polynomial delay algorithm for enumerating minimal dominating sets in chordal graphs. CoRRabs/1404.3501
17. Kijima S, Okamoto Y, Uno T (2011) Dominating set counting in graph classes. In: COCOON 2011, Dallas, pp 13–24
18. Kotek T, Preen J, Simon F, Tittmann P, Trinks M (2012) Recurrence relations and splitting formulas for the domination polynomial. Electron J Comb 19(3):P47
19. Mary A (2013) Énumération des Dominants Minimaux d'un graphe. PhD thesis, Université Blaise Pascal
20. Oum S, Seymour PD (2006) Approximating clique-width and branch-width. J Comb Theory Ser B 96(4):514–528
21. Takata K (2007) A worst-case analysis of the sequential method to list the minimal hitting sets of a hypergraph. SIAM J Discret Math 21(4):936–946

Minimal Perfect Hash Functions

Paolo Boldi and Sebastiano Vigna
Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy

Keywords

Minimal perfect hash functions

Years and Authors of Summarized Original Work

1984; Fredman, Komlós
1984; Mehlhorn
1996; Majewski, Wormald, Havas, Czech
2001; Hagerup, Tholey
2004; Chazelle, Kilian, Rubinfeld, Tal
2007; Botelho, Pagh, Ziviani
2009; Belazzougui, Botelho, Dietzfelbinger

Problem Definition

A minimal perfect hash function (MPHF) is a (data structure providing a) bijective map from a set S of n keys to the set of the first n natural numbers. In the static case (i.e., when the set S is known in advance), there is a wide spectrum of solutions available, offering different trade-offs in

terms of construction time, access time, and size of the data structure.

Problem Formulation

Let $[x]$ denote the set of the first x natural numbers. Given a positive integer $u = 2^w$, and a set $S \subseteq [u]$ with $|S| = n$, a function $h : S \to [m]$ is *perfect* if and only if it is injective and *minimal* if and only if $m = n$. An (M)PHF is a data structure that allows one to evaluate a (minimal) perfect function of this kind.

When comparing different techniques for building MPHFs, one should be aware of the trade-offs between construction time, evaluation time, and space needed to store the function. A general tenet is that evaluation should happen in constant time (with respect to n), whereas construction is only required to be feasible in a practical sense.

Space is often the most important aspect when a construction is taken into consideration; usually space is computed in an exact (i.e., non-asymptotic) way. Some exact space lower bounds for this problem are known (they are pure space bounds and do not consider evaluation time): Fredman and Komlós proved [4] that no MPHF can occupy less than $n \log e + \log\log u + O(\log n)$ bits, as soon as $u \geq n^{2+\epsilon}$; this bound is essentially tight [9, Sect. III.2.3, Thm. 8], disregarding evaluation time.

Key Results

One fundamental question is how close to the space lower bound $n \log e + \log\log u$ one can stay if the evaluation must be performed in constant time. The best theoretical results in this direction are given in [6], where an $n \log e + \log\log u + O(n(\log\log n)^2/\log n + \log\log\log u)$ technique is provided (optimal up to an additive factor) whose construction takes linear time in expectation. The technique is only of theoretical relevance, though, as it yields a low number of bits per key only for unrealistically large values of n.

We will describe two practical solutions: the first one provides a structure that is simple, constant time, and asymptotically space optimal (i.e., $O(n)$); its actual space requirement is about twice the lower bound. The second one can potentially approach the lower bound, even if in practice this would require an unfeasibly long construction time; nonetheless, it provides the smallest known practical data structure – it occupies about 1.44 times the lower bound.

We present the two constructions in some detail below; they both use the idea of building an MPHF out of a PHF that we explain first.

From a PHF to a MPHF

Given a set $T \subseteq [m]$ of size $|T| = n$, define $\text{rank}_T : [m] \to [n]$ by letting

$$\text{rank}_T(p) = |\{i \in T \mid i < p\}|.$$

Clearly, every PHF $g : S \to [m]$ can be combined with $\text{rank}_{g(S)} : [m] \to [n]$ to obtain an MPHF. Jacobson [7] offers a constant-time implementation for the rank data structure that uses $o(m)$ additional bits besides the set T represented as an array of m bits; furthermore, constant-time solutions exist that require as little as $O(n/(\log n)^c)$ (for any desired c) over the information-theoretical lower bound $\log \binom{m}{n}$ [10].

For practical solutions, see [5, 11]. For very sparse sets T, the Elias-Fano scheme can be rewarding in terms of space, but query time becomes $O(\log(m/n))$.

The Hypergraph-Based Construction

We start by recalling the hypergraph-based construction presented in [8]. Their method, albeit originally devised only for order-preserving MPHF, can be used to store compactly an *arbitrary* r-bit function $f : S \to [2^r]$. The construction draws three hash functions $h_0, h_1, h_2 : S \to [\gamma n]$ (with $\gamma \approx 1.23$) and builds a 3-hypergraph with one hyperedge $(h_0(x), h_1(x), h_2(x))$ for every $x \in S$. With positive probability, this hypergraph does not have a nonempty 2-core, that is, its hyperedges can be sorted in such a way that every hyperedge contains (at least) a vertex that never appeared before, called the *hinge*. Equivalently, the set of equations (in the variables a_i)

$$f(x) = \left(a_{h_0(x)} + a_{h_1(x)} + a_{h_2(x)}\right) \bmod 2^r$$

has a solution that can be found by a hypergraph-peeling process in time $O(n)$. Storing the function consists in storing γn integers of r bits each (the array a_i), so γrn bits are needed (excluding the bits needed for the hash functions); function evaluation takes constant time.

In [3] the authors (which were not aware of [8]) present a "mutable Bloomier filter," which is formed by a PHF and a data storage indexed by the output of the PHF. The idea is to let $r = 2$ and to decide f *after* the hinges have been successfully determined, letting $f(x)$ be the index of the hinge of the hyperedge associated with x, that is, the index $i \in \{0, 1, 2\}$ such that $h_i(x)$ is the hinge of $(h_0(x), h_1(x), h_2(x))$. This way, the function $g : S \to [m]$ defined by $g(x) = h_{f(x)}(x)$ is a PHF, and it is stored in $2\gamma n$ bits.

The fact that combining such a construction with a ranking data structure might actually provide an MPHF was noted in [2] (whose authors did not know [3]). An important implementation trick that makes it possible to get $\approx$2.65 bits per key is the fact that $r = 2$, but actually we need to store three values. Thus, when assigning values to hinges, we can use 3 (which modulo 3 is equivalent to zero) in place of 0: in this way, hinges are exactly associated to those a_i that are nonzero, which makes it possible to build a custom ranking structure that does not use an additional bit vector, but rather ranks directly nonzero pairs of bits.

The "Hash, Displace, and Compress" Construction

A completely different approach is suggested in [1]: once more, they first build a PHF $h : S \to [m]$ where $m = (1 + \epsilon)n$ for some $\epsilon > 0$. The set S is first divided into r buckets by means of a first-level hash function $g : S \to [r]$; the r buckets $g^{-1}(0), \ldots, g^{-1}(r-1)$ are sorted by their cardinalities, with the largest buckets first.

Let $B_0, \ldots, B_{r-1}$ be the buckets and let $\phi_0, \phi_1, \phi_2, \ldots$ be a sequence of independent fully random hash functions $S \to [m]$. For every $i = 0, \ldots, r-1$, the construction algorithm determines the smallest index p_i such that ϕ_{p_i} is injective when applied to B_i and moreover $\phi_{p_i}(B_i)$ is disjoint from $\cup_{j<i} \phi_{p_j}(B_j)$. A careful analysis shows that this construction takes linear time in expectation (the choice of r impacts on construction time) and that the expected p_i is bounded by a constant, so the indices can be stored in $O(\log(1/\epsilon)n)$ space.

In practice, if r is chosen so that the average bucket size is $\approx$5, it is possible to obtain an MPHF using $\approx$2.05 bits per key with a construction time that is still feasible, albeit an order of magnitude larger than the hypergraph-based construction.

The authors of [1] also discuss a variant that can directly build MPHFs, but the construction time is no longer linear in expectation; moreover, from a practical viewpoint it is useful to enlarge slightly the buckets so that they have a prime size (this makes it easier to generate a good sequence of hash functions [1]).

Open Problems

Improving construction and query time in practice and getting closer to the space lower bound keeping the construction feasible are the main open problems about MPHFs, as there are already known constructions that close the gap asymptotically.

URLs to Code and Data Sets

The Sux4J library (http://sux4j.di.unimi.it/) provides Java implementations of the methods we discussed. The CMPH library (http://cmph.sourceforge.net/) provides C implementations.

Cross-References

- Monotone Minimal Perfect Hash Functions
- Rank and Select Operations on Bit Strings

Recommended Reading

1. Belazzougui D, Botelho FC, Dietzfelbinger M (2009) Hash, displace, and compress. In: Fiat A, Sanders P (eds) Algorithms – ESA 2009, 17th annual European

symposium, Copenhagen, 7–9 Sept 2009, proceedings, pp 682–693
2. Botelho FC, Pagh R, Ziviani N (2007) Simple and space-efficient minimal perfect hash functions. In: Dehne FKHA, Sack JR, Zeh N (eds) Proceedings of the WADS 2007, 10th international workshop on algorithms and data structures, Halifax. Lecture notes in computer science, vol 4619. Springer, pp 139–150
3. Chazelle B, Kilian J, Rubinfeld R, Tal A (2004) The Bloomier filter: an efficient data structure for static support lookup tables. In: Munro JI (ed) Proceedings of the fifteenth annual ACM-SIAM symposium on discrete algorithms, SODA 2004, New Orleans. SIAM, pp 30–39
4. Fredman ML, Komlós J (1984) On the size of separating systems and families of perfect hash functions. SIAM J Algebr Discret Methods 5(1):61–68
5. Gog S, Petri M (2014) Optimized succinct data structures for massive data. Softw Pract Exp 44(11):1287–1314
6. Hagerup T, Tholey T (2001) Efficient minimal perfect hashing in nearly minimal space. In: Ferreira A, Reichel H (eds) STACS 2001, 18th annual symposium on theoretical aspects of computer science, Dresden, 15–17 Feb 2001, proceedings, pp 317–326
7. Jacobson G (1989) Space-efficient static trees and graphs. In: 30th annual symposium on foundations of computer science (FOCS '89), Research Triangle Park. IEEE Computer Society, pp 549–554
8. Majewski BS, Wormald NC, Havas G, Czech ZJ (1996) A family of perfect hashing methods. Comput J 39(6):547–554
9. Mehlhorn K (1984) Data structures and algorithms 1: sorting and searching. EATCS monographs on theoretical computer science, vol 1. Springer, Berlin/New York
10. Patrascu M (2008) Succincter. In: 49th annual IEEE symposium on foundations of computer science, Philadelphia. IEEE Computer Society, pp 305–313
11. Vigna S (2008) Broadword implementation of rank/select queries. In: McGeoch CC (ed) 7th international workshop on experimental algorithms, WEA 2008, Provincetown. Lecture notes in computer science, vol 5038. Springer, pp 154–168

Minimum Bisection

Robert Krauthgamer
Weizmann Institute of Science, Rehovot, Israel
IBM Almaden Research Center, San Jose, CA, USA

Keywords

Graph bisection

Years and Authors of Summarized Original Work

1999; Feige, Krauthgamer

Problem Definition

Overview

Minimum bisection is a basic representative of a family of discrete optimization problems dealing with partitioning the vertices of an input graph. Typically, one wishes to minimize the number of edges going across between the different pieces, while keeping some control on the partition, say by restricting the number of pieces and/or their size. (This description corresponds to an edge-cut of the graph; other variants correspond to a vertex-cut with similar restrictions.) In the minimum bisection problem, the goal is to partition the vertices of an input graph into two equal-size sets, such that the number of edges connecting the two sets is as small as possible.

In a seminal paper in 1988, Leighton and Rao [14] devised for MINIMUM-BISECTION a logarithmic-factor bicriteria approximation algorithm. (A bicriteria approximation algorithm partitions the vertices into two sets each containing at most 2/3 of the vertices, and its value, i.e., the number of edges connecting the two sets, is compared against that of the best partition into equal-size sets.) Their algorithm has found numerous applications, but the question of finding a true approximation with a similar factor remained open for over a decade later. In 1999, Feige and Krauthgamer [6] devised the first polynomial-time algorithm that approximates this problem within a factor that is polylogarithmic (in the graph size).

Cuts and Bisections

Let $G = (V, E)$ be an undirected graph with $n = |V|$ vertices, and assume for simplicity that n is even. For a subset S of the vertices, let $\bar{S} = V \setminus S$. The *cut* (also known as *cutset*) $(S, \bar{S})$ is defined as the set of all edges with one endpoint in S and one endpoint in $\bar{S}$. These edges are said to *cross* the cut, and the two sets S and $\bar{S}$ are called the two *sides* of the cut.

Assume henceforth that G has nonnegative edge-weights. (In the unweighted version, every edge has a unit weight.) The *cost* of a cut $(S, \bar{S})$ is then defined to be the total edge-weight of all the edges crossing the cut.

A cut $(S, \bar{S})$ is called a *bisection* of G if its two sides have equal cardinality, namely $|S| = |\bar{S}| = n/2$. Let $b(G)$ denote the minimum cost of a bisection of G.

Problem 1 (MINIMUM-BISECTION)
INPUT: An undirected graph G with nonnegative edge-weights.
OUTPUT: A bisection $(S, \bar{S})$ of G that has minimum cost.

This definition has a crucial difference from the classical MINIMUM-CUT problem (see e.g., [10] and references therein), namely, there is a restriction on the sizes of the two sides of the cut. As it turns out, MINIMUM-BISECTION is NP-hard (see [9]), while MINIMUM-CUT can be solved in polynomial time.

Balanced Cuts and Edge Separators

The above rather basic definition of minimum bisection can be extended in several ways. Specifically, one may require only an upper bound on the size of each side. For $0 < \beta < 1$, a cut $(S, \bar{S})$ is called β-balanced if $\max\{|S|, |\bar{S}|\} \leq \beta n$. Note the latter requirement implies $\min\{|S|, |\bar{S}|\} \geq (1 - \beta)n$. In this terminology, a bisection is a 1/2-balanced cut.

Problem 2 (β-BALANCED-CUT)
INPUT: An undirected graph G with nonnegative edge-weights.
OUTPUT: A β-balanced cut $(S, \bar{S})$ of G with $\max\{|S|, |\bar{S}|\} \leq \beta n$, that has cost as small as possible.

The special case of $\beta = 2/3$ is commonly refered to as the EDGE-SEPARATOR problem.

In general, the sizes of the two sides may be specified in advance arbitrarily (rather than being equal); in this case the input contains a number k, and the goal is to find a cut $(S, \bar{S})$ such that $|S| = k$. One may also wish to divide the graph into more than two pieces of equal size and then the input contains a number $r \geq 2$, or alternatively, to divide the graph into r pieces of whose sizes are $k_1, \ldots, k_r$, where the numbers k_i are prescribed in the input; in either case, the goal is to minimize the number of edges crossing between different pieces.

Problem 3 (PRESCRIBED-PARTITION)
INPUT: An undirected graph $G = (V, E)$ with nonnegative edge-weights, and integers $k_1, \ldots, k_r$ such that $\sum_i k_i = |V|$.
OUTPUT: A partition $V = V_1 \cup \cdots \cup V_r$ of G with $|V_i| = k_i$ for all i, such that the total edge-weight of edges whose endpoints lie in different sets V_i is as small as possible.

Key Results

The main result of Feige and Krauthgamer [6] is an approximation algorithm for MINIMUM-BISECTION. The approximation factor they originally claimed is $O(\log^2 n)$, because it used the algorithm of Leighton and Rao [14]; however, by using instead the algorithm of [2], the factor immediately improves to $O(\log^{1.5} n)$.

Theorem 1 Minimum-Bisection *can be approximated in polynomial time within* $O(\log^{1.5} n)$ *factor. Specifically, the algorithm produces for an input graph* G *a bisection* $(S, \bar{S})$ *whose cost is at most* $O(\log^{1.5} n) \cdot b(G)$.

The algorithm immediately extends to similar results for related and/or more general problems that are defined above.

Theorem 2 β-Balanced-Cut *(and in particular Edge-Separator) can be approximated in polynomial time within* $O(\log^{1.5} n)$ *factor.*

Theorem 3 Prescribed-Partition *can be approximated in time* $n^{O(r)}$ *to within* $O(\log^{1.5} n)$ *factor.*

For all three problems above, the approximation ratio improves to $O(\log n)$ for the family of graphs excluding a fixed minor (which includes in particular planar graphs). For simplicity, this result is stated for Minimum-Bisection.

Theorem 4 *In graphs excluding a fixed graph as a minor (e.g., planar graphs), the problems (i)* Minimum-Bisection, *(ii)* β-Balanced-Cut, *and (iii)* Prescribed-Partition *with fixed r can all be approximated in polynomial time within factor* $O(\log n)$.

It should be noted that all these results can be generalized further, including vertex-weights and terminals-vertices ($s-t$ pairs), see [Sect. 5 in 6].

Related Work

A bicriteria approximation algorithm for β-balanced cut returns a cut that is β'-balanced for a predetermined $\beta' > \beta$. For bisection, for example, $\beta = 1/2$ and typically $\beta' = 2/3$.

The algorithms in the above theorems use (in a black-box manner) an approximation algorithm for a problem called minimum quotient-cuts (or equivalently, sparsest-cut with uniform-demands). For this problem, the best approximation currently known is $O(\sqrt{\log n})$ for general graphs due to Arora, Rao, and Vazirani [2], and $O(1)$ for graphs excluding a fixed minor due to Klein, Plotkin, and Rao [13]. These approximation algorithms for minimum quotient-cuts immediately give a polynomial time bicriteria approximation (sometimes called pseudo-approximation) for MINIMUM-BISECTION. For example, in general graphs the algorithm is guaranteed to produce a 2/3-balanced cut whose cost is at most $O(\sqrt{\log n}) \cdot b(G)$. Note however that a 2/3-balanced cut does not provide a good approximation for the value of $b(G)$. For instance, if G consists of three disjoint cliques of equal size, an optimal 2/3-balanced cut has no edges, whereas $b(G) = \Omega(n^2)$. For additional related work, including approximation algorithms for dense graphs, for directed graphs, and for other graph partitioning problems, see [Sect. 1 in 6] and the references therein.

Applications

One major motivation for MINIMUM-BISECTION, and graph partitioning in general, is a divide-and-conquer approach to solving a variety of optimization problems, especially in graphs, see e.g., [15, 16]. In fact, these problems arise naturally in a wide range of practical settings such as VLSI design and image processing; sometimes, the motivation is described differently, e.g., as a clustering task.

Another application of MINIMUM-BISECTION is in assignment problems, of a form that is common in parallel systems and in scientific computing: jobs need to be assigned to machines in a balanced way, while assigning certain pairs of jobs the same machine, as much as possible. For example, consider assigning n jobs to 2 machines, when the amount of communication between every two jobs is known, and the goal is to have equal load (number of jobs) on each machine, and bring to minimum the total communication that goes between the machines. Clearly, this last problem can be restated as MINIMUM-BISECTION in a suitable graph.

It should be noted that in many of these settings, a true approximation is not absolutely necessary, and a bicriteria approximation may suffice. Nevertheless, the algorithms stated in section "Key Results" have been used to design algorithms for other problems, such as (1) an approximation algorithm for minimum bisection in k-uniform hypergraphs [3]; (2) an approximation algorithm for a variant of the minimum multicut problem [17]; and (3) an algorithm that efficiently certifies the unsatisfiability of random $2k$-SAT with sufficiently many clauses [5].

From a practical perspective, numerous heuristics (algorithms without worst-case guarantees) for graph partitioning have been proposed and studied, see [1] for an extensive survey. For example, one of the most famous heuristics is Kerninghan and Lin's local search heuristic for minimum bisection [11].

Open Problems

Currently, there is a large gap between the $O(\log^{1.5} n)$ approximation ratio for MINIMUM-BISECTION achieved by Theorem 1 and the hardness of approximation results known for

it. As mentioned above, MINIMUM-BISECTION is known to be NP-hard (see [9]).

The problem is not known to be APX-hard but several results provide evidence towards this possibility. Bui and Jones [4] show that for every fixed $\epsilon > 0$, it is NP-hard to approximate the minimum bisection within an *additive* term of $n^{2-\epsilon}$. Feige [7] showed that if refuting 3SAT is hard on average on a natural distribution of inputs, then for every fixed $\varepsilon > 0$ there is no $4/3 - \varepsilon$ approximation algorithm for minimum bisection. Khot [12] proved that minimum bisection does not admit a polynomial-time approximation scheme (PTAS) unless NP has randomized sub-exponential time algorithms.

Taking a broader perspective, currently there is a (multiplicative) gap of $O(\log n)$ between the approximation ratio for MINIMUM-BISECTION and that of minimum quotient-cuts (and thus also to the factor achieved by bicriteria approximation). It is interesting whether this gap can be reduced, e.g., by using the algorithm of [2] in a non-black box manner.

The vertex-cut version of MINIMUM-BISECTION is defined as follows: the goal is to partition the vertices of the input graph into $V = A \cup B \cup S$ with $|S|$ as small as possible, under the constraints that $\max\{|A|, |B|\} \leq n/2$ and no edge connects A with B. It is not known whether a polylogarithmic factor approximation can be attained for this problem. It should be noted that the same question regarding the directed version of MINIMUM-BISECTION was answered negatively by Feige and Yahalom [8].

Cross-References

See entry on the paper by Arora, Rao, and Vazirani [2].

- ▸ Separators in Graphs
- ▸ Sparsest Cut

Recommended Reading

1. Alpert CJ, Kahng AB (1995) Recent directions in netlist partitioning: a survey. Integr VLSI J 19(1–2):1–81
2. Arora S, Rao S, Vazirani U (2004) Expander flows, geometric embeddings, and graph partitionings. In: 36th annual symposium on the theory of computing, Chicago, June 2004, pp 222–231
3. Berman P, Karpinski M (2003) Approximability of hypergraph minimum bisection. ECCC report TR03-056, Electronic Colloquium on Computational Complexity, vol 10
4. Bui TN, Jones C (1992) Finding good approximate vertex and edge partitions is NP-hard. Inform Process Lett 42(3):153–159
5. Coja-Oghlan A, Goerdt A, Lanka A, Schädlich F (2004) Techniques from combinatorial approximation algorithms yield efficient algorithms for random 2k-SAT. Theory Comput Sci 329(1–3):1–45
6. Feige U (2002) Relations between average case complexity and approximation complexity. In: 34th annual ACM symposium on the theory of computing, Montréal, 19–21 May 2002, pp 534–543
7. Feige U, Krauthgamer R (2006) A polylogarithmic approximation of the minimum bisection. SIAM Rev 48(1):99–130, Previous versions appeared in Proceedings of 41st FOCS, 1999; and in SIAM J Comput 2002
8. Feige U, Yahalom O (2003) On the complexity of finding balanced oneway cuts. Inf Process Lett 87(1):1–5
9. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman
10. Karger DR (2000) Minimum cuts in near-linear time. J ACM 47(1):46–76
11. Kernighan BW, Lin S (1970) An efficient heuristic procedure for partitioning graphs. Bell Syst Tech J 49(2):291–307
12. Khot S (2004) Ruling out PTAS for graph min-bisection, densest subgraph and bipartite clique. In: 45th annual IEEE symposium on foundations of computer science, Georgia Institute of Technology, Atlanta, 17–19 Oct 2004, pp 136–145
13. Klein P, Plotkin SA, Rao S (1993) Excluded minors, network decomposition, and multicommodity flow. In: 25th annual ACM symposium on theory of computing, San Diego, 16–18 May 1993, pp 682–690
14. Leighton T, Rao S (1999) Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. J ACM 46(6):787–832, 29th FOCS, 1988
15. Lipton RJ, Tarjan RE (1980) Applications of a planar separator theorem. SIAM J Comput 9(3): 615–627
16. Rosenberg AL, Heath LS (2001) Graph separators, with applications. Frontiers of computer science. Kluwer/Plenum, New York
17. Svitkina Z, Tardos É (2004) Min-Max multiway cut. In: 7th international workshop on approximation algorithms for combinatorial optimization (APPROX), Cambridge, 22–24 Aug 2004, pp 207–218

Minimum Congestion Redundant Assignments

Dimitris Fotakis[1] and Paul (Pavlos) Spirakis[2,3,4]
[1]Department of Information and Communication Systems Engineering, University of the Aegean, Samos, Greece
[2]Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece
[3]Computer Science, University of Liverpool, Liverpool, UK
[4]Computer Technology Institute (CTI), Patras, Greece

Keywords

Maximum fault-tolerant partition; Minimum fault-tolerant congestion

Years and Authors of Summarized Original Work

2002; Fotakis, Spirakis

Problem Definition

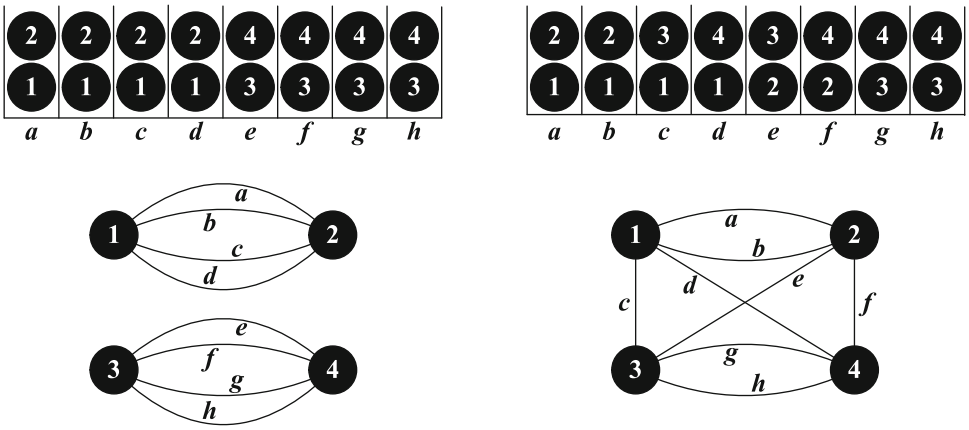
This problem is concerned with the most efficient use of redundancy in load balancing on faulty parallel links. More specifically, this problem considers a setting where some messages need to be transmitted from a source to a destination through some faulty parallel links. Each link fails independently with a given probability, and in case of failure, none of the messages assigned to it reaches the destination. (This assumption is realistic if the messages are split into many small packets transmitted in a round-robin fashion. Then the successful delivery of a message requires that all its packets should reach the destination.) An assignment of the messages to the links may use redundancy, i.e., assign multiple copies of some messages to different links. The reliability of a redundant assignment is the probability that every message has a copy on some active link, thus managing to reach the destination. Redundancy increases reliability, but also increases the message load assigned to the links. A good assignment should achieve high reliability and keep the maximum load of the links as small as possible.

The reliability of a redundant assignment depends on its structure. In particular, the reliability of different assignments putting the same load on every link and using the same number of copies for each message may vary substantially (e.g., compare the reliability of the assignments in Fig. 1). The crux of the problem is to find an efficient way of exploiting redundancy in order to achieve high reliability and low maximum load. (If one does not insist on minimizing the maximum load, a reliable assignment is constructed by assigning every message to the most reliable links.)

The work of Fotakis and Spirakis [1] formulates the scenario above as an optimization problem called *Minimum Fault-Tolerant Congestion* and suggests a simple and provably efficient approach of exploiting redundancy. This approach naturally leads to the formulation of another interesting optimization problem, namely that of computing an efficient fault-tolerant partition of a set of faulty parallel links. [1] presents polynomial-time approximation algorithms for computing a fault-tolerant partition of the links and proves that combining fault-tolerant partitions with standard load balancing algorithms results in a good approximation to Minimum Fault-Tolerant Congestion. To the best knowledge of the entry authors, this work is the first to consider the approximability of computing a redundant assignment that minimizes the maximum load of the links subject to the constraint that random faults should be tolerated with a given probability.

Notations and Definitions

Let M denote a set of m faulty parallel links connecting a source s to a destination t, and let J denote a set of n messages to be transmitted from s to t. Each link i has a rational capacity $c_i \geq 1$ and a rational failure probability $f_i \in (0, 1)$. Each message j has a rational



Minimum Congestion Redundant Assignments, Fig. 1 Two redundant assignments of 4 unit size messages to 8 identical links. Both assign every message to 4 links and 2 messages to every link. The corresponding graph is depicted below each assignment. The assignment on the *left* is the most reliable 2-partitioning assignment ϕ_2. Lemma 3 implies that for every failure probability f, ϕ_2 is at least as reliable as any other assignment ϕ with $\text{Cong}(\phi) \leq 2$. For instance, ϕ_2 is at least as reliable as the assignment on the *right*. Indeed the reliability of the assignment on the right is $1 - 4f^4 + 2f^6 + 4f^7 - 3f^8$, which is bounded from above by $\text{Rel}(\phi_2) = 1 - 2f^4 + f^8$ for all $f \in [0, 1]$

size $s_j \geq 1$. Let $f_{\max} \equiv \max_{i \in M}\{f_i\}$ denote the failure probability of the most unreliable link. Particular attention is paid to the special case of identical capacity links, where all capacities are assumed to be equal to 1.

The reliability of a set of links M', denoted $\text{Rel}(M')$, is the probability that there is an active link in M'. Formally, $\text{Rel}(M') \equiv 1 - \prod_{i \in M'} f_i$. The reliability of a collection of disjoint link subsets $\mathcal{M} = \{M_1, \ldots, M_v\}$, denoted $\text{Rel}(\mathcal{M})$, is the probability that there is an active link in every subset of $\mathcal{M}$. Formally,

$$\text{Rel}(\mathcal{M}) \equiv \prod_{\ell=1}^{v} \text{Rel}(M_\ell) = \prod_{\ell=1}^{v} \left(1 - \prod_{i \in M_\ell} f_i\right) .$$

A redundant assignment $\phi : J \mapsto 2^M \setminus \emptyset$ is a function that assigns every message j to a non-empty set of links $\phi(j) \subseteq M$. An assignment ϕ is feasible for a set of links M' if for every message j, $\phi(j) \cap M' \neq \emptyset$. The *reliability* of an assignment ϕ, denoted $\text{Rel}(\phi)$, is the probability that ϕ is feasible for the actual set of active links. Formally,

$$\text{Rel}(\phi) \equiv \sum_{\substack{M' \subseteq M \\ \forall j \in J, \phi(j) \cap M' \neq \emptyset}} \left(\prod_{i \in M'} (1 - f_i) \prod_{i \in M \setminus M'} f_i \right)$$

The *congestion* of an assignment ϕ, denoted $\text{Cong}(\phi)$, is the maximum load assigned by ϕ to a link in M. Formally,

$$\text{Cong}(\phi) \equiv \max_{i \in M} \left\{ \sum_{j : i \in \phi(j)} \frac{s_j}{c_i} \right\} .$$

Problem 1 (Minimum Fault-Tolerant Congestion)
INPUT: *A set of faulty parallel links* $M = \{(c_1, f_1), \ldots, (c_m, f_m)\}$, *a set of messages* $J = \{s_1, \ldots, s_n\}$, *and a rational number* $\epsilon \in (0, 1)$.
OUTPUT: *A redundant assignment* $\phi : J \mapsto 2^M \setminus \emptyset$ *with* $\text{Rel}(\phi) \geq 1 - \epsilon$ *that minimizes* $\text{Cong}(\phi)$.

Minimum Fault-Tolerant Congestion is **NP**-hard because it is a generalization of minimizing makespan on (reliable) parallel machines. The decision version of Minimum Fault-Tolerant Congestion belongs to **PSPACE**, but it is not clear whether it belongs to **NP**. The reason is that computing the reliability of a redundant

assignment and deciding whether it is a feasible solution is **#P**-complete.

The work of Fotakis and Spirakis [1] presents polynomial-time approximation algorithms for Minimum Fault-Tolerant Congestion based on a simple and natural class of redundant assignments whose reliability can be computed easily. The high level idea is to separate the reliability aspect from load balancing. Technically, the set of links is partitioned in a collection of disjoint subsets $\mathcal{M} = \{M_1, \ldots, M_\nu\}$ with $\mathrm{Rel}(\mathcal{M}) \geq 1 - \epsilon$. Every subset $M_\ell \in \mathcal{M}$ is regarded as a *reliable link* of *effective capacity* $c(M_\ell) \equiv \min_{i \in M_\ell}\{c_i\}$. Then any algorithm for load balancing on reliable parallel machines can be used for assigning the messages to the subsets of $\mathcal{M}$, thus computing a redundant assignment ϕ with $\mathrm{Rel}(\phi) \geq 1 - \epsilon$.

The assignments produced by this approach are called *partitioning assignments*. More precisely, an assignment $\phi : J \mapsto 2^M \setminus \emptyset$ is a *ν-partitioning assignment* if for every pair of messages j, j', either $\phi(j) = \phi(j')$ or $\phi(j) \cap \phi(j') = \emptyset$, and ϕ assigns the messages to ν different link subsets.

Computing an appropriate fault-tolerant collection of disjoint link subsets is an interesting optimization problem by itself. A feasible solution $\mathcal{M}$ satisfies the constraint that $\mathrm{Rel}(\mathcal{M}) \geq 1 - \epsilon$. For identical capacity links, the most natural objective is to maximize the number of subsets in $\mathcal{M}$ (equivalently, the number of reliable links used by the load balancing algorithm). For arbitrary capacities, this objective generalizes to maximizing the total effective capacity of $\mathcal{M}$.

Problem 2 (Maximum Fault-Tolerant Partition)
INPUT: A set of faulty parallel links $M = \{(c_1, f_1), \ldots, (c_m, f_m)\}$, and a rational number $\epsilon \in (0, 1)$.
OUTPUT: A collection $\mathcal{M} = \{M_1, \ldots, M_\nu\}$ of disjoint subsets of M with $\mathrm{Rel}(\mathcal{M}) \geq 1 - \epsilon$ that maximizes $\sum_{\ell=1}^{\nu} c(M_\ell)$.

The problem of Maximum Fault-Tolerant Partition is **NP**-hard. More precisely, given m identical capacity links with rational failure probabilities and a rational number $\epsilon \in (0, 1)$, it is **NP**-complete to decide whether the links can be partitioned into sets M_1 and M_2 with $\mathrm{Rel}(M_1) \cdot \mathrm{Rel}(M_2) \geq 1 - \epsilon$.

Key Results

Theorem 1 *There is a 2-approximation algorithm for Maximum Fault-Tolerant Partition of identical capacity links. The time complexity of the algorithm is* $O((m - \sum_{i \in M} \ln f_i) \ln m)$.

Theorem 2 *For every constant* $\delta > 0$*, there is a* $(8+\delta)$*-approximation algorithm for Maximum Fault-Tolerant Partition of capacitated links. The time complexity of the algorithm is polynomial in the input size and* $1/\delta$.

To demonstrate the efficiency of the partitioning approach for Maximum Fault-Tolerant Congestion, Fotakis and Spirakis prove that for certain instances, the reliability of the most reliable partitioning assignment bounds from above the reliability of any other assignment with the same congestion (see Fig. 1 for an example).

Lemma 3 *For any positive integers* Λ, ν, μ *and any rational* $f \in (0, 1)$*, let* ϕ *be a redundant assignment of* $\Lambda\nu$ *unit size messages to* $\nu\mu$ *identical capacity links with failure probability* f*. Let* ϕ_ν *be the* ν*-partitioning assignment that assigns* Λ *messages to each of* ν *disjoint subsets consisting of* μ *links each. If* $\mathrm{Cong}(\phi) \leq \Lambda = \mathrm{Cong}(\phi_\nu)$*, then* $\mathrm{Rel}(\phi) \leq (1 - f^\mu)^\nu = \mathrm{Rel}(\phi_\nu)$.

Based on the previous upper bound on the reliability of any redundant assignment, [1] presents polynomial-time approximation algorithms for Maximum Fault-Tolerant Congestion.

Theorem 4 *There is a quasi-linear-time 4-approximation algorithm for Maximum Fault-Tolerant Congestion on identical capacity links.*

Theorem 5 *There is a polynomial-time* $2\lceil \ln(m/\epsilon)/\ln(1/f_{\max}) \rceil$*-approximation algorithm for Maximum Fault-Tolerant Congestion on instances with unit size messages and capacitated links.*

Applications

In many applications dealing with faulty components (e.g., fault-tolerant network design, fault-tolerant routing), a combinatorial structure (e.g., a graph, a hypergraph) should optimally tolerate random faults with respect to a given property (e.g., connectivity, non-existence of isolated points). For instance, Lomonosov [5] derived tight upper and lower bounds on the probability that a graph remains connected under random edge faults. Using the bounds of Lomonosov, Karger [3] obtained improved theoretical and practical results for the problem of estimating the reliability of a graph. In this work, Lemma 3 provides a tight upper bound on the probability that isolated nodes do not appear in a not necessarily connected hypergraph with $\Lambda\nu$ nodes and $\nu\mu$ "faulty" hyperedges of cardinality Λ.

More precisely, let ϕ be any assignment of $\Lambda\nu$ unit size messages to $\nu\mu$ identical links that assigns every message to μ links and Λ messages to every link. Then ϕ corresponds to a hypergraph H_ϕ, where the set of nodes consists of $\Lambda\nu$ elements corresponding to the unit size messages and the set of hyperedges consists of $\nu\mu$ elements corresponding to the identical links. Every hyperedge contains the messages assigned to the corresponding link and has cardinality Λ (see Fig. 1 for a simple example with $\Lambda = 2$, $\nu = 2$, and $\mu = 4$). Clearly, an assignment ϕ is feasible for a set of links $M' \subseteq M$ iff the removal of the hyperedges corresponding to the links in $M \setminus M'$ does not leave any isolated nodes (For a node v, let $\deg_H(v) \equiv |\{e \in E(H) : v \in e\}|$. A node v is isolated in H if $\deg_H(v) = 0$) in H_ϕ. Lemma 3 implies that the hypergraph corresponding to the most reliable ν-partitioning assignment maximizes the probability that isolated nodes do not appear when hyperedges are removed equiprobably and independently.

The previous work on fault-tolerant network design and routing mostly focuses on the worst-case fault model, where a feasible solution must tolerate any configuration of a given number of faults. The work of Gasieniec et al. [2] studies the fault-tolerant version of minimizing congestion of virtual path layouts in a complete ATM network. In addition to several results for the worst-case fault model, [2] constructs a virtual path layout of logarithmic congestion that tolerates random faults with high probability. On the other hand, the work of Fotakis and Spirakis shows how to construct redundant assignments that tolerate random faults with a probability given as part of the input and achieve a congestion close to optimal.

Open Problems

An interesting research direction is to determine the computational complexity of Minimum Fault-Tolerant Congestion and related problems. The decision version of Minimum Fault-Tolerant Congestion is included in the class of languages decided by a polynomial-time non-deterministic Turing machine that reduces the language to a single call of a **#P** oracle. After calling the oracle once, the Turing machine rejects if the oracle's outcome is less than a given threshold and accepts otherwise. This class is denoted $\mathbf{NP}^{\mathbf{\#P}[1,\text{comp}]}$ in [1]. In addition to Minimum Fault-Tolerant Congestion, $\mathbf{NP}^{\mathbf{\#P}[1,\text{comp}]}$ includes the decision version of Stochastic Knapsack considered in [4]. A result of Toda and Watanabe [6] implies that $\mathbf{NP}^{\mathbf{\#P}[1,\text{comp}]}$ contains the entire Polynomial Hierarchy. A challenging open problem is to determine whether the decision version of Minimum Fault-Tolerant Congestion is complete for $\mathbf{NP}^{\mathbf{\#P}[1,\text{comp}]}$.

A second direction for further research is to consider the generalizations of other fundamental optimization problems (e.g., shortest paths, minimum connected subgraph) under random faults. In the fault-tolerant version of minimum connected subgraph for example, the input consists of a graph whose edges fail independently with given probabilities, and a rational number $\epsilon \in (0, 1)$. The goal is to compute a spanning subgraph with a minimum number of edges whose reliability is at least $1 - \epsilon$.

Cross-References

- Approximation Schemes for Bin Packing
- Bin Packing
- Connectivity and Fault Tolerance in Random Regular Graphs
- List Scheduling

Recommended Reading

1. Fotakis D, Spirakis P (2002) Minimum congestion redundant assignments to tolerate random faults. Algorithmica 32:396–422
2. Gasieniec L, Kranakis E, Krizanc D, Pelc A (1996) Minimizing congestion of layouts for ATM networks with faulty links. In: Penczek W, Szalas A (eds) Proceedings of the 21st international symposium on mathematical foundations of computer science. Lecture notes in computer science, vol 1113. Springer, Berlin, pp 372–381
3. Karger D (1999) A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. SIAM J Comput 29:492–514
4. Kleinberg J, Rabani Y, Tardos E (2000) Allocating bandwidth for bursty connections. SIAM J Comput 30:191–217
5. Lomonosov M (1974) Bernoulli scheme with closure. Probl Inf Transm 10:73–81
6. Toda S, Watanabe O (1992) Polynomial-time 1-turing reductions from #PH to #P. Theor Comput Sci 100:205–221

Minimum Connected Sensor Cover

Lidong Wu[1] and Weili Wu[2,3,4]
[1]Department of Computer Science, The University of Texas, Tyler, TX, USA
[2]College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi Province, China
[3]Department of Computer Science, California State University, Los Angeles, CA, USA
[4]Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Keywords

Approximation algorithm; Connected sensor cover; Performance ratio

Years and Authors of Summarized Original Work

2013; Wu, Du, Wu, Li, Lv, Lee

Problem Definition

Nowadays, the sensor exists everywhere. The wireless sensor network has been studied extensively. In view of this type of networks, there are two most important properties, coverage and connectivity. In fact, the sensor is often used for collecting information, and hence, its sensing area has to cover the target (points or area). Usually, for a wireless sensor, its sensing area is a disk with the center at the sensor. The radius of this disk is called the sensing radius. After information is collected, the sensor has to send to central station for analysis. This requires all active sensors to form a connected communication network. Actually, every sensor has also a communication function, and it can send information to other sensors located in its communication area, which is also a disk with center at the sensor. The radius of the communication disk is called the communication radius.

The sensor is often very small and energy is often supplied with batteries. Therefore, energy efficiency is a big issue in the study of wireless sensor networks. A sensor network is said to be *homogeneous* if all sensors in the network have the same size of sensing radius and the same size of communication radius. For a homogeneous wireless sensor network, the energy consumption can be measured by the number of active sensors. The minimum connected sensor cover problem is a classic optimization problem based on the above consideration in the study of wireless sensor networks, which is described as follows.

Consider a homogeneous wireless sensor network in the Euclidean plane. Given a connected target area Ω, find the minimum number of sensors satisfying the following two conditions:

[Coverage] The target area Ω is covered by selected sensors.

[Connectivity] Selected sensors induce a connected network.

A subset of sensors is called a *sensor cover* if it satisfies the coverage condition and called a *connected sensor cover* if it satisfies both the coverage condition and the connectivity condition.

The minimum connected sensor cover problem is NP-hard. The study on approximation solutions of this problem has attracted many researchers.

Key Results

The minimum connected sensor cover problem was first proposed by Gupta, Das, and Gu [8]. They presented a greedy algorithm with performance ratio $O(r \ln n)$ where n is the number of sensors and r is the link radius of the sensor network, i.e., for any two sensors with overlapping sensing disks, their hop distance in the communication network is at most r.

Zhang and Hou [12] studied a special case that the communication radius is at least twice of the sensing radius, and they showed that in this case, the coverage of a connected region implied the connectivity. In this case, they presented a polynomial-time constant-approximation.

Das and Gupta [13] and Xing et al. [11] explored more about the relationship between coverage and connectivity. Bai et al. [2] studied a sensor deployment problem regarding the coverage and connectivity. Alam and Haas [1] studied the minimum connected sensor cover problem in three-dimensional sensor networks.

Funke et al. [5] allow sensors to vary their sensing radius. With variable sensing radius and communication radius, Zhou, Das, and Gupta [14] designed a polynomial-time approximation with performance ratio $O(\log n)$. Chosh and Das [6] designed a greedy approximation using the maximal independent set and Voronoi diagram. They determined the size of connected sensor cover produced by their algorithm. However, no comparison with optimal solution, that is, no analysis on approximation performance ratio, is given. In fact, none of the above efforts give an improvement on the approximation performance ratio $O(r \log n)$ given by Gupta, Das, and Gu [8].

Wu et al. [10] made the first improvement. They present two polynomial-time approximations. The first one has performance ratio $O(r)$. This approximation is designed based on a polynomial-time constant-approximation [4] or a polynomial-time approximation scheme [9] for the minimum target coverage problem as follows: given a homogeneous set of sensors and a set of target points in the Euclidean plane, find the minimum number of sensors covering all given target points.

The $O(r)$-approximation consists of three steps. In the first step, it replaces the target area by $O(n^2)$ target points such that the target area is covered by a subset of sensors if and only if those target points are all covered by this subset of sensors. In the second step, it computes a constant-approximation solution, say c-approximation solution S for the minimum target coverage problem with those target points as input. Since the optimal solution for the minimum connected sensor cover problem must be a feasible solution for the minimum target coverage problem. $|S|$ is within a factor c of the size of a minimum connected sensor cover, i.e., $|S| \le c \cdot \mathrm{opt}_{\mathrm{mcsc}}$.

In the third step, the algorithm employs a polynomial-time 1.39-approximation algorithm for the network Steiner tree problem [3], and apply this algorithm on the input consisting of a graph with unit weight for each edge, which is the communication network of sensors, and a terminal set S. Note that in a graph with unit weight for each edge, the total edge weight of a tree is the number of vertices in the tree minus one. Therefore, the result obtained in the third step is a connected sensor cover with cardinality upper bounded by $|S|$ plus $(1 + 1.39 \times$ (size of minimum network Steiner tree on $S))$.

To estimate the size of minimum network Steiner tree on S, consider a minimum connected sensor cover S^*. Let T be a spanning tree in the subgraph induced by S^*. For each sensor $s \in S$, there must exist a sensor $s' \in S^*$ with sensing disk overlapping with the sensing disk of s. Therefore, there is a path P_s, with distance at most r, connecting s and s' in the communication network. Clearly, $T \cup (\cup_{s \in S} P_s)$ is a Steiner tree on S. Hence, the size of minimum network

Steiner tree on S is at most $|S| - 1 + |S| \cdot r$. Therefore, the connected sensor cover obtained in the third step has size at most

$$\begin{aligned} |S| + 1 + 1.39 \cdot (|S|(r+1) - 1) &\leq |S|(1.39r + 2.39) \\ &\leq c(1.39r + 2.39) \cdot \mathrm{opt}_{\mathrm{mcsc}} \\ &= O(r) \cdot \mathrm{opt}_{\mathrm{mcsc}}. \end{aligned}$$

The second polynomial-time approximation designed by Wu et al. [10] is a random algorithm with performance ratio $O(\log^3 n)$. This approximation is obtained by the following two observations: (1) The minimum connected sensor cover problem is a special case of the minimum connected set cover problem. (2) The minimum connected set-cover problem has a close relationship to the group Steiner tree. Therefore, some results on group Steiner trees can be transformed into connected sensor covers.

In conclusion, Wu et al. [10] obtained the following:

Theorem 1 *There exists a polynomial-time $O(r)$-approximation for the minimum connected sensor cover problem. There exists also a polynomial-time random algorithm with performance $O(\log^3 n)$ for the minimum connected sensor cover problem.*

Open Problems

For two approximations in Theorem 1, one has performance ratio $O(r)$ independent from n and the other one has performance ratio $O(\log^3 n)$ independent from r. This fact suggests that either n or r is closely related or there exists a polynomial-time constant-approximation. Therefore, we have the following conjecture:

Conjecture 1 There exists a polynomial-time $O(1)$-approximation for the minimum connected sensor cover problem.

Cross-References

- Connected Set-Cover and Group Steiner Tree
- Performance-Driven Clustering

Recommended Reading

1. Alam SMN, Haas ZJ (2006) Coverage and connectivity in three-dimensional networks. In: MobiCom'06, Los Angeles
2. Bai X, Kumar S, Xuan D, Yun Z, Lai TH (2006) DEploying wireless sensors to achieve both coverage and connectivity. In: MobiHoc'06, Florence, pp 131–142
3. Byrka J, Grandoni F, Rothvoss T, Sanita L (2010) An improved LP-based approximation for Steiner tree. In: STOC'10, Cambridge, 5–8 June, pp 583–592
4. Ding L, Wu W, Willson JK, Wu L, Lu Z, Lee W (2012) Constant-approximation for target coverage problem in wireless sensor networks. In: Proceedings of the 31st annual joint conference of IEEE communication and computer society (INFOCOM), Miami
5. Funke S, Kesselman A, Kuhn F, Lotker Z, Segal M (2007) Improved approximation algorithms for connected sensor cover. Wirel Netw 13:153–164
6. Ghosh A, Das SK (2005) A distributed greedy algorithm for connected sensor cover in dense sensor networks. LNCS 3560:340–353
7. Ghosh A, Das SK (2006) Coverage and connectivity issues in wireless sensor networks. In: Shorey R, Ananda AL, Chan MC, Ooi WT (eds) Mobile, wireless, and sensor networks: technology, applications, and future directions. Wiley, Hoboken, pp 221–256
8. Gupta H, Das SR, Gu Q (2003) Connected sensor cover: self-organization of sensor networks for efficient query execution. In: MobiHoc'03, Annapolis, pp 189–200
9. Mustafa N, Ray S (2009) PTAS for geometric hitting set problems via local search. In: Proceedings of the SoCG 2009, Aarhus, pp 17–22. ACM, New York
10. Wu L, Du H, Wu W, Li D, Lv J, Lee W (2013) Approximations for minimum connected sensor cover. In: INFOCOM, Turin
11. Xing G, Wang X, Zhang Y, Liu C, Pless R, Gill C (2005) Integrated coverage and connectivity configuration for energy conservation in sensor networks. ACM Trans Sens Netw 1(1):36–72
12. Zhang H, Hou JC (2005) Maintaining sensing coverage and connectivity in large sensor networks. Ad Hoc Sens Wirel Netw 1:89–124
13. Zhou Z, Das S, Gupta H (2004) Connected k-coverage problem in sensor networks. In: Proceedings of the 13th international conference on computer communications and networks, Chicago, pp 373–378
14. Zhou Z, Das SR, Gupta H (2009) Variable radii connected sensor cover in sensor networks. ACM Trans Sens Netw 5(1):1–36

Minimum Energy Broadcasting in Wireless Geometric Networks

Christoph Ambühl
Department of Computer Science, University of Liverpool, Liverpool, UK

Keywords

Energy-efficient broadcasting in wireless networks

Years and Authors of Summarized Original Work

2005; Ambühl

Problem Definition

In the most common model for wireless networks, stations are represented by points in $\mathbb{R}^d$. They are equipped with a omnidirectional transmitter and receiver which enables them to communicate with other stations. In order to send a message from a station s to a station t, station s needs to emit the message with enough power such that t can receive it. It is usually assumed that the power required by a station s to transmit data directly to station t is $\|st\|^\alpha$, for some constant $\alpha \geq 1$, where $\|st\|$ denotes the distance between s and t.

Because of the omnidirectional nature of the transmitters and receivers, a message sent by a station s with power r^α can be received by all stations within a disc of radius r around s. Hence the energy required to send a message from a station s directly to a set of stations S' is determined by $\max_{v \in S'} \|sv\|^\alpha$.

An instance of the *minimum energy broadcast routing problem in wireless networks* (MEBR) consists of a set of stations S and a constant $\alpha \geq 1$. One of the stations in S is designated as the source station s_0. The goal is to send a message at minimum energy cost from s_0 to all other stations in S. This operation is called *broadcast*.

In the case $\alpha = 1$, the optimal solution is to send the message directly from s_0 to all other stations. For $\alpha > 1$, sending the message via intermediate stations which forward it to other stations is often more energy efficient.

A solution of the MEBR instance can be described in terms of a so-called *broadcast tree*. That is, a directed spanning tree of S which contains directed paths from s_0 to all other vertices. The solution described by a broadcast tree T is the one in which every station forwards the message to all its out-neighbors in T.

Problem 1 (MEBR)
INSTANCE: A set S of points in R^d, $s_0 \in S$ designated as the source, and a constant α.
SOLUTION: A broadcast tree T of S.
MEASURE: The objective is to minimize the total energy needed to broadcast a message from s_0 to all other nodes,which can be expressed by

$$\sum_{u \in S} \max_{v \in \delta(u)} \|uv\|^\alpha, \tag{1}$$

where $\delta(u)$ denotes the set of out-neighbors of station u in T.

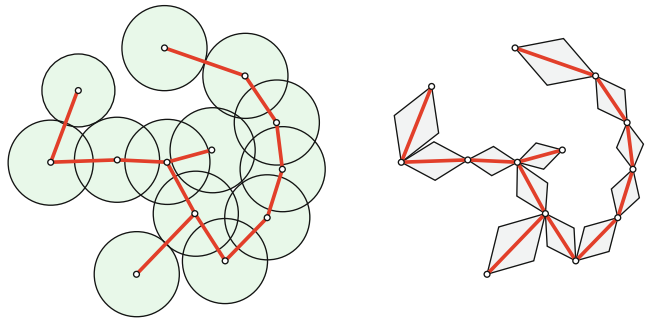
The MEBR problem is known to be NP-hard for $d \geq 2$ and $\alpha > 1$ [2]. APX-hardness is known for $d \geq 3$ [5].

Key Results

Numerous heuristics have been proposed for this problem. Only a few of them have been analyzed theoretically. The one which attains the best approximation guarantee is the so-called MST-heuristic [12].

MST-HEURISTIC: Compute a minimum spanning tree of S (*mst(S)*) and turn it into an broadcast tree by directing the edges.

Theorem 1 ([1]) *In the Euclidean plane, the MST-heuristic is a 6 approximation algorithm for MEBR for all $\alpha \geq 2$.*



Minimum Energy Broadcasting in Wireless Geometric Networks, Fig. 1 Illustration of the first and second approach for bounding *w(S)*. In both approaches, *w(S)* is bounded in terms of the total area covered by the shapes

Theorem 2 ([9]) *In the Euclidean three-dimensional space, the MST-heuristic is a 18.8 approximation algorithm for MEBR for all* $\alpha \geq 3$.

For $\alpha < d$, the MST-heuristic does not provide a constant approximation ratio. The d-dimensional kissing numbers represent lower bounds for the performance of the MST-heuristic. Hence the analysis for $d = 2$ is tight, whereas for $d = 3$ the lower bound is 12.

Analysis

The analysis of the MST-heuristic is based on good upper bounds for

$$w(S) := \sum_{e \in mst(S)} \|e\|^{\alpha}, \qquad (2)$$

which obviously is an upper bound on (1). The radius of an instance of MEBR is the distance between s_0 to the station furthest from s_0. It turns out that the MST-heuristic performs worst on instances of radius 1 whose optimal solution is to broadcast the message directly from s_0 to all other stations. Since the optimal value for such instances is 1, the approximation ratio follows from good upper bounds on *w(S)* for instances with radius 1.

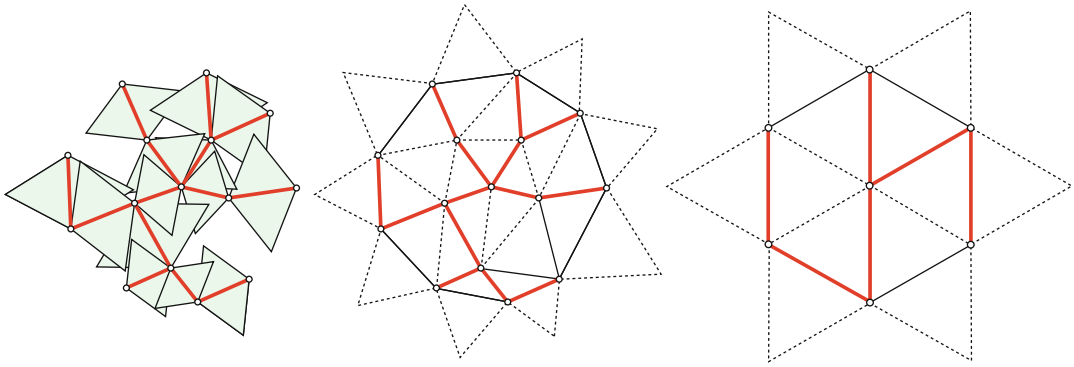
The rest of this section focuses on the case $d = \alpha = 2$. There are two main approaches for upper bounding *w(S)*. In both approaches, *w(S)* is upper bounded in terms of the area of particular kinds of shapes associated with either the stations or with the edges of the MST.

In the first approach, the shapes are disks of radius $m/2$ placed around every station of S, where m is the length of the longest edge of *mst(S)*. Let A denote the area covered by the disks. One can prove $w(S) \leq \frac{4}{\pi}\left(A - \pi(m/2)^2\right)$. Assuming that S has radius 1, one can prove $w(S) \leq 8$ quite easily [4]. This approach can even be extended to obtain $w(S) \leq 6.33$ [8], and it can be generalized for $d \geq 2$.

In the second approach [7, 11], *w(S)* is expressed in terms of shapes associated with the edges of *mst(S)*, e.g., diamond shapes as shown on the right of Fig. 1. The area of a diamond for an edge e is equal to $\|e\|^2/(2\sqrt{3})$. Since one can prove that the diamonds never intersect, one obtains $w(S) = A/(2\sqrt{3})$. For instances with radius 1, one can get $w(S) \leq 12.15$.

For the 2-dimensional case, one can even obtain a matching upper bound [1]. The shapes used in this proof are equilateral triangles, arranged in pairs along every edge of the MST. As can be seen on the left of Fig. 2, these shapes do intersect. Still one can obtain a good upper bound on their total area by means of the convex hull of S:

Let the *extended convex hull of S* be the convex hull of S extended by equilateral triangles along the border of the convex hull. One can prove that the total area generated by the equilateral triangle shapes along the edges of *mst(S)* is upper bounded by the area of the extended convex hull



Minimum Energy Broadcasting in Wireless Geometric Networks, Fig. 2 Illustration of the tight bound for $d = 2$. The total area of the equilateral triangles on the left is bounded by its extended convex hull shown in the middle. The point set that maximizes area of the extended convex hull is the star shown on the right

of S. By showing that for instances of radius 1 the area of the extended convex hull is maximized by the point configuration shown on the right of Fig. 2, the matching upper bound of 6 can be established.

Applications

The MEBR problem is a special case of a large class of problems called *range assignment problems*. In all these problems, the goal is to assign a range to each station such that a certain type of communication operation such as broadcast, all-to-1 (gathering), all-to-all (gossiping), can be accomplished. See [3] for a survey on range assignment problems.

It is worth noting that the problem of upper bounding *w(S)* has already been considered in different contexts. The idea of using diamond shapes to upper bound the length of an MST has already been used by Gilbert and Pollak in [6]. Steele [10] makes use of space filling curves to bound *w(S)*.

Open Problems

An obvious open problem is to close the gap in the analysis of the MST-heuristic for the three dimensional case. This might be very difficult, as the lower bound from the kissing number might not be tight.

Even in the plane, the approximation ratio of the MST-heuristic is quite large. It would be interesting to see a different approach for the problem, maybe based on LP-rounding. It is still not known whether MEBR is APX-hard for instances in the Euclidean plane. Hence there might exist a PTAS for this problem.

Cross-References

- Broadcasting in Geometric Radio Networks
- Deterministic Broadcasting in Radio Networks
- Geometric Spanners
- Minimum Geometric Spanning Trees
- Minimum Spanning Trees
- Randomized Broadcasting in Radio Networks
- Randomized Gossiping in Radio Networks

Recommended Reading

1. Ambühl C (2005) An optimal bound for the MST algorithm to compute energy efficient broadcast trees in wireless networks. In: Proceedings of 32th international colloquium on automata, languages and programming (ICALP). Lecture notes in computer science, vol 3580. Springer, Berlin, pp 1139–1150
2. Clementi A, Crescenzi P, Penna P, Rossi G, Vocca P (2001) On the complexity of computing minimum energy consumption broadcast subgraphs. In: Proceedings of the 18th annual symposium on theoretical aspects of computer science (STACS), pp 121–131

3. Clementi A, Huiban G, Penna P, Rossi G, Verhoeven Y (2002) Some recent theoretical advances and open questions on energy consumption in ad-hoc wireless networks. In: Proceedings of the 3rd workshop on approximation and randomization algorithms in communication networks (ARACNE), pp 23–38
4. Flammini M, Klasing R, Navarra A, Perennes S (2004) Improved approximation results for the minimum energy broadcasting problem. In: Proceedings of the 2004 joint workshop on foundations of mobile computing
5. Fuchs B (2006) On the hardness of range assignment problems. In: Proceedings of the 6th Italian conference on algorithms and complexity (CIAC), pp 127–138
6. Gilbert EN, Pollak HO (1968) Steiner minimal trees. SIAM J Appl Math 16:1–29
7. Klasing R, Navarra A, Papadopoulos A, Perennes S (2004) Adaptive broadcast consumption (ABC), a new heuristic and new bounds for the minimum energy broadcast routing problem. In: Proceedings of the 3rd IFIP-TC6 international networking conference (NETWORKING), pp 866–877
8. Navarra A (2005) Tighter bounds for the minimum energy broadcasting problem. In: Proceedings of the 3rd international symposium on modeling and optimization in mobile, ad-hoc and wireless networks (WiOpt), pp 313–322
9. Navarra A (2006) 3-D minimum energy broadcasting. In: Proceedings of the 13th colloquium on structural information and communication complexity (SIROCCO), pp 240–252
10. Steele JM (1989) Cost of sequential connection for points in space. Oper Res Lett 8:137–142
11. Wan P-J, Calinescu G, Li X-Y, Frieder O (2002) Minimum-energy broadcasting in static ad hoc wireless networks. Wirel Netw 8:607–617
12. Wieselthier JE, Nguyen GD, Ephremides A (2002) Energy-efficient broadcast and multicast trees in wireless networks. Mob Netw Appl 7:481–492

Minimum Energy Cost Broadcasting in Wireless Networks

Peng-Jun Wan, Xiang-Yang Li, and Ophir Frieder
Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA

Keywords

MEB; Minimum energy broadcast; MST

Years and Authors of Summarized Original Work

2001; Wan, Calinescu, Li, Frieder

Problem Definition

Ad hoc wireless networks have received significant attention in recent years due to their potential applications in battlefield, emergency disaster relief and other applications [11, 15]. Unlike wired networks or cellular networks, no wired backbone infrastructure is installed in ad hoc wireless networks. A communication session is achieved either through a single-hop transmission if the communication parties are close enough, or through relaying by intermediate nodes otherwise. Omni-directional antennas are used by all nodes to transmit and receive signals. They are attractive in their broadcast nature. A single transmission by a node can be received by many nodes within its vicinity. This feature is extremely useful for multicasting/broadcasting communications. For the purpose of energy conservation, each node can dynamically adjust its transmitting power based on the distance to the receiving node and the background noise. In the most common power-attenuation model [10], the signal power falls as $\frac{1}{r^\kappa}$, where r is the distance from the transmitter antenna and κ is a real *constant* between 2 and 4 dependent on the wireless environment. Assume that all receivers have the same power threshold for signal detection, which is typically normalized to one. With these assumptions, the power required to support a link between two nodes separated by a distance r is r^κ. A key observation here is that relaying a signal between two nodes may result in lower total transmission power than communicating over a large distance due to the nonlinear power attenuation. They assume the network nodes are given as a finite point (The terms node, point and vertex are interchangeable here: node is a network term, point is a geometric term, and vertex is a graph term.) set P in a two-dimensional plane. For any real number κ, they use $G^{(\kappa)}$ to denote the weighted

complete graph over P in which the weight of an edge e is $\|\mathbf{e}\|^\kappa$.

The minimum-energy unicast routing is essentially a shortest-path problem in $G^{(\kappa)}$. Consider any unicast path from a node $\mathbf{p} = \mathbf{p}_0 \in P$ to another node $\mathbf{q} = \mathbf{p}_m \in P$: $\mathbf{p}_0\mathbf{p}_1 \cdots \mathbf{p}_{m-1}\mathbf{p}_m$. In this path, the transmission power of each node $\mathbf{p}_i$, $0 \le i \le m-1$, is $\|\mathbf{p}_i\mathbf{p}_{i+1}\|^\kappa$ and the transmission power of $\mathbf{p}_m$ is zero. Thus the total transmission energy required by this path is $\sum_{i=0}^{m-1} \|\mathbf{p}_i\mathbf{p}_{i+1}\|^\kappa$, which is the total weight of this path in G^κ. So by applying any shortest-path algorithm such as the Dijkstra's algorithm [5], one can solve the minimum-energy unicast routing problem.

However, for broadcast applications (in general multicast applications), Minimum-Energy Routing is far more challenging. Any broadcast routing is viewed as an arborescence (a directed tree) T, rooted at the source node of the broadcasting, that spans all nodes. Use $f_T(\mathbf{p})$ to denote the transmission power of the node $\mathbf{p}$ required by T. For any leaf node $\mathbf{p}$ of T, $f_T(\mathbf{p}) = 0$. For any internal node $\mathbf{p}$ of T,

$$f_T(\mathbf{p}) = \max_{\mathbf{pq} \in T} \|\mathbf{pq}\|^\kappa,$$

in other words, the κ-th power of the longest distance between $\mathbf{p}$ and its children in T. The total energy required by T is $\sum_{\mathbf{p} \in P} f_T(\mathbf{p})$. Thus the minimum-energy broadcast routing problem is different from the conventional link-based minimum spanning tree (MST) problem. Indeed, while the MST can be solved in polynomial time by algorithms such as Prim's algorithm and Kruskal's algorithm [5], it is NP-hard [4] to find the minimum-energy broadcast routing tree for nodes placed in two-dimensional plane. In its general graph version, the minimum-energy broadcast routing can also be shown to be NP-hard [7], and even worse, it can not be approximated within a factor of $(1-\epsilon)\log\Delta$, unless $NP \subseteq DTIME\left[n^{O(\log\log n)}\right]$, by an approximation-preserving reduction from the Connected Dominating Set problem [8], where Δ is the maximal degree and ϵ is any arbitrary small positive constant.

Three greedy heuristics have been proposed for the minimum-energy broadcast routing problem by [15]. The MST heuristic first applies the Prim's algorithm to obtain a MST, and then orient it as an arborescence rooted at the source node. The SPT heuristic applies the Dijkstra's algorithm to obtain a SPT rooted at the source node. The BIP heuristic is the node version of Dijkstra's algorithm for SPT. It maintains, throughout its execution, a single arborescence rooted at the source node. The arborescence starts from the source node, and new nodes are added to the arborescence one at a time on the minimum incremental cost basis until all nodes are included in the arborescence. The incremental cost of adding a new node to the arborescence is the minimum additional power increased by some node in the current arborescence to reach this new node. The implementation of BIP is based on the standard Dijkstra's algorithm, with one fundamental difference on the operation whenever a new node q is added. Whereas the Dijkstra's algorithm updates the node weights (representing the current knowing distances to the source node), BIP updates the cost of each link (representing the incremental power to reach the head node of the directed link). This update is performed by subtracting the cost of the added link pq from the cost of every link qr that starts from q to a node r not in the new arborescence.

Key Results

The performance of these three greedy heuristics have been evaluated in [15] by simulation studies. However, their analytic performances in terms of the approximation ratio remained open until [13]. The work of Wan et al. [13] derived the bounds on their approximation ratios.

Let us begin with the SPT algorithm. Let ϵ be a sufficiently small positive number. Consider m nodes $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m$ evenly distributed on a cycle of radius 1 centered at a node $\mathbf{o}$. For $1 \le i \le m$, let $\mathbf{q}_i$ be the point in the line segment $\mathbf{op}_i$ with $\|\mathbf{oq}_i\| = \epsilon$. They consider a broadcasting from the node $\mathbf{o}$ to these $n = 2m$ nodes $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m, \mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_m$. The SPT is the

superposition of paths $\mathbf{oq}_i\mathbf{p}_i$, $1 \le i \le m$. Its total energy consumption is $\epsilon^2 + m(1-\epsilon)^2$. On the other hand, if the transmission power of node $\mathbf{o}$ is set to 1, then the signal can reach all other points. Thus the minimum energy consumed by all broadcasting methods is at most 1. So the approximation ratio of SPT is at least $\epsilon^2 + m(1-\epsilon)^2$. As $\epsilon \longrightarrow 0$, this ratio converges to $\frac{n}{2} = m$.

They [13] also proved that

Theorem 1 *The approximation ratio of MST is at least 6 for any $\kappa \ge 2$.*

Theorem 2 *The approximation ratio of BIP is at least $\frac{13}{3}$ for any $\kappa = 2$.*

They then derived the upper bounds by extensively using the geometric structures of Euclidean MSTs (EMST). They first observed that as long as the cost of a link is an increasing function of the Euclidean length of the link, the set of MSTs of any point set *coincides* with the set of Euclidean MSTs of the same point set. They proved a key result about an upper bound on the parameter $\sum_{e \in MST(P)} \|e\|^2$ for any finite point set P inside a disk with radius one.

Theorem 3 *Let c be the supreme of $\sum_{e \in MST(P)} \|e\|^2$ over all such point sets P. Then $6 \le c \le 12$.*

The following lemma proved in [13] is used to bound the energy cost for broadcast when each node can dynamically adjust its power.

Lemma 4 For any point set P in the plane, the total energy required by any broadcasting among P is at least $\frac{1}{c}\sum_{e \in MST(P)} \|e\|^\kappa$.

Lemma 5 For any broadcasting among a point set P in a two-dimensional plane, the total energy required by the arborescence generated by the BIP algorithm is at most $\sum_{e \in MST(P)} \|e\|^\kappa$.

Thus, they conclude the following two theorems.

Theorem 6 *The approximation ratio of EMST is at most c, and therefore is at most 12.*

Theorem 7 *The approximation ratio of BIP is at most c, and therefore is at most 12.*

Later, Wan et al. [14] studied the energy efficient multicast for wireless networks when each node can dynamically adjust its power. Given a set of receivers Q, the problem Min-Power Asymmetric Multicast seeks, for any given communication session, an arborescence T of minimum total power which is rooted at the source node s and reaches all nodes in Q. As a generalization of Min-Power Asymmetric Broadcast Routing, Min-Power Asymmetric Multicast Routing is also NP-hard. Wieselthier et al. [15] adapted their three broadcasting heuristics to three multicasting heuristics by a technique of pruning, which was called as pruned minimum spanning tree (P-MST), pruned shortest-path tree (P-SPT), and pruned broadcasting incremental power (P-BIP), respectively in [14]. The idea is as follows. They first obtain a spanning tree rooted at the source of a given multicast session by applying any of the three broadcasting heuristics. They then eliminate from the spanning arborescence all nodes which do not have any descendant in Q. They [14] show by constructing examples that all structures P-SPT, P-MST and P-BIP could have approximation ratio as large as $\Theta(n)$ in the worst case for multicast. They then further proposed a multicast scheme with a constant approximation ratio on the total energy consumption. Their protocol for Min-Power Asymmetric Multicast Routing is based on Takahashi-Matsuyama Steiner tree heuristic [12]. Initially, the multicast tree T contains only the source node. At each iterative step, the multicast tree T is grown by one path from some node in T to some destination node from Q that is not yet in the tree T. The path must have the least total power among all such paths from a node in T to a node in $Q - T$. This procedure is repeated until all required nodes are included in T. This heuristic is referred to as Shortest Path First (SPF).

Theorem 8 *For asymmetric multicast communication, the approximation ratio of SPF is between 6 and 2c, which is at most 24.*

Applications

Broadcasting and multicasting in wireless ad hoc networks are critical mechanisms in various applications such as information diffusion, wireless networks, and also for maintaining consistent global network information. Broadcasting is often necessary in MANET routing protocols. For example, many unicast routing protocols such as Dynamic Source Routing (DSR), Ad Hoc On Demand Distance Vector (AODV), Zone Routing Protocol (ZRP), and Location Aided Routing (LAR) use broadcasting or a derivation of it to establish routes. Currently, these protocols all rely on a simplistic form of broadcasting called *flooding*, in which each node (or all nodes in a localized area) retransmits each received unique packet exactly one time. The main problems with flooding are that it typically causes unproductive and often harmful bandwidth congestion, as well as inefficient use of node resources. Broadcasting is also more efficient than sending multiple copies the same packet through unicast. It is highly important to use power-efficient broadcast algorithms for such networks since wireless devises are often powered by batteries only.

Open Problems

There are some interesting questions left for further study. For example, the exact value of the constant c remains unsolved. A tighter upper bound on c can lead to tighter upper bounds on the approximation ratios of both the link-based MST heuristic and the BIP heuristic. They conjecture that the exact value for c is 6, which seems to be true based on their extensive simulations. The second question is what is the approximation lower bound for minimum energy broadcast? Is there a PTAS for this problem?

So far, all the known theoretically good algorithms either assume that the power needed to support a link uv is proportional to $\|uv\|^\kappa$ or is a fixed cost that is independent of the neighboring nodes that it will communicate with. In practice, the energy consumption of a node is neither solely dependent on the distance to its farthest neighbor, nor totally independent of its communication neighbor. For example, a more general power consumption model for a node u would be $c_1 + c_2 \cdot \|uv\|^\kappa$ for some constants $c_1 \geq 0$ and $c_2 \geq 0$ where v is its farthest communication neighbor in a broadcast structure. No theoretical result is known about the approximation of the optimum broadcast or multicast structure under this model. When $c_2 = 0$, this is the case where all nodes have a fixed power for communication. Minimizing the total power used by a reliable broadcast tree is equivalent to the minimum connected dominating set problem (MCDS), i.e., minimize the number of nodes that relay the message, since all relaying nodes of a reliable broadcast form a connected dominating set (CDS). Notice that recently a PTAS [2] has been proposed for MCDS in UDG graph.

Another important question is how to find efficient broadcast/multicast structures such that the delay from the source node to the last node receiving message is bounded by a predetermined value while the total energy consumption is minimized. Notice that here the delay of a broadcast/multicast based on a tree is not simply the height of the tree: many nodes cannot transmit simultaneously due to the interference.

Cross-References

- Broadcasting in Geometric Radio Networks
- Deterministic Broadcasting in Radio Networks

Recommended Reading

1. Ambühl C (2005) An optimal bound for the MST algorithm to compute energy efficient broadcast trees in wireless networks. In: Proceedings of 32th international colloquium on automata, languages and programming (ICALP). LNCS, vol 3580, pp 1139–1150
2. Cheng X, Huang X, Li D, Du D-Z (2003) Polynomial-time approximation scheme for minimum connected dominating set in ad hoc wireless networks. Networks 42:202–208
3. Chvátal V (1979) A greedy heuristic for the set-covering problem. Math Oper Res 4(3):233–235

4. Clementi A, Crescenzi P, Penna P, Rossi G, Vocca P (2001) On the complexity of computing minimum energy consumption broadcast subgraphs. In: 18th annual symposium on theoretical aspects of computer science. LNCS, vol 2010, pp 121–131
5. Cormen TJ, Leiserson CE, Rivest RL (1990) Introduction to algorithms. MIT/McGraw-Hill, Columbus
6. Flammini M, Navarra A, Klasing R, Pérennes A (2004) Improved approximation results for the minimum energy broadcasting problem. DIALM-POMC. ACM, New York, pp 85–91
7. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman and Company, New York
8. Guha S, Khuller S (1998) Approximation algorithms for connected dominating sets. Algorithmica 20:347–387
9. Preparata FP, Shamos MI (1985) Computational geometry: an introduction. Springer, New York
10. Rappaport TS (1996) Wireless communications: principles and practices. Prentice Hall/IEEE, Piscataway
11. Singh S, Raghavendra CS, Stepanek J (1999) Power-aware broadcasting in mobile ad hoc networks. In: Proceedings of IEEE PIMRC'99, Osaka
12. Takahashi H, Matsuyama A (1980) An approximate solution for Steiner problem in graphs. Math Jpn 24(6):573–577
13. Wan P-J, Calinescu G, Li X-Y, Frieder O (2002) Minimum-energy broadcast routing in static ad hoc wireless networks. ACM Wirel Netw 8(6):607–617, Preliminary version appeared in IEEE INFOCOM (2000)
14. Wan P-J, Calinescu G, Yi C-W (2004) Minimum-power multicast routing in static ad hoc wireless networks. IEEE/ACM Trans Netw 12:507–514
15. Wieselthier JE, Nguyen GD, Ephremides A (2000) On the construction of energy-efficient broadcast and multicast trees in wireless networks. IEEE Infocom 2:585–594

Minimum Flow Time

Nikhil Bansal
Eindhoven University of Technology,
Eindhoven, The Netherlands

Keywords

Response time; Sojourn time

Years and Authors of Summarized Original Work

1997; Leonardi, Raz

Problem Definition

The problem is concerned with efficiently scheduling jobs on a system with multiple resources to provide a good quality of service. In scheduling literature, several models have been considered to model the problem setting, and several different measures of quality have been studied. This note considers the following model: There are several identical machines, and jobs are released over time. Each job is characterized by its size, which is the amount of processing it must receive to be completed, and its release time, before which it cannot be scheduled. In this model, Leonardi and Raz studied the objective of minimizing the average flow time of the jobs, where the flow time of a job is the duration of time since it is released until its processing requirement is met. Flow time is also referred to as response time or sojourn time and is a very natural and commonly used measure of the quality of a schedule.

Notations

Let $\mathcal{J} = \{1, 2, \ldots, n\}$ denote the set of jobs in the input instance. Each job j is characterized by its release time r_j and its processing requirement p_j. There is a collection of m identical machines, each having the same processing capability. A schedule specifies which job executes at what time on each machine. Given a schedule, the completion time c_j of a job is the earliest time at which job j receives p_j amount of service. The flow time f_j of j is defined as $c_j - r_j$. A schedule is said to be preemptive if a job can be interrupted arbitrarily and its execution can be resumed later from the point of interruption without any penalty. A schedule is non-preemptive if a job cannot be interrupted once it is started. In the context of multiple machines, a schedule is said to be migratory if a job can be moved from one

machine to another during its execution without any penalty. In the off-line model, all the jobs J are given in advance. In scheduling algorithms, the online model is usually more realistic than the off-line model.

Key Results

For a single machine, it is a folklore result that the Shortest Remaining Processing Time (SRPT) policy that at any time works on the job with the least remaining processing time is optimal for minimizing the average flow time. Note that SRPT is an online algorithm and is a preemptive scheduling policy.

If no preemption is allowed, Kellerer, Tautenhahn, and Woeginger [6] gave an $O(n^{1/2})$ approximation algorithm for minimizing the flow time on a single machine and also showed that no polynomial time algorithm can have an approximation ratio of $n^{1/2-\varepsilon}$ for any $\varepsilon > 0$ unless P = NP.

Leonardi and Raz [8] gave the first nontrivial results for minimizing the average flow time on multiple machines. Later, a simpler presentation of this result was given by Leonardi [7]. The main result of [8] is the following.

Theorem 1 ([8]) *On multiple machines, the SRPT algorithm is* $O(\min(\log(n/m), \log P))$ *competitive for minimizing average flow time, where* P *is the maximum to minimum job size ratio.*

They also gave a matching lower bound (up to constant factors) on the competitive ratio.

Theorem 2 ([8]) *For the problem of minimizing flow time on multiple machines, any online algorithm has a competitive ratio of* $\Omega(\min(\log(n/m), \log P))$*, even when randomization is allowed.*

Note that minimizing the average flow time is equivalent to minimizing the total flow time. Suppose each job pays \$1 at each time unit it is alive (i.e., unfinished), then the total payment received is equal to the total flow time. Summing up the payment over each time step, the total flow time can be expressed as the summation over the number of unfinished jobs at each time unit. As SRPT works on jobs that can be finished as soon as possible, it seems intuitively that it should have the least number of unfinished jobs at any time. While this is true for a single machine, it is not true for multiple machines (as shown in an example below). The main idea of [8] was to show that at any time, the number of unfinished jobs under SRPT is "essentially" no more than $O(\min(\log P))$ times that under any other algorithm. To do this, they developed a technique of grouping jobs into a logarithmic number of classes according to their remaining sizes and arguing about the total unfinished work in these classes. This technique has found a lot of uses since then to obtain other results. To obtain a guarantee in terms of n, some additional ideas are required.

The instance below shows how SRPT could deviate from optimum in the case of multiple machines. This instance is also the key component in the lower bound construction in Theorem 2 above. Suppose there are two machines, and three jobs of size 1, 1, and 2 arrive at time $t = 0$. SRPT would schedule the two jobs of size 1 at $t = 0$ and then work on size 2 job at time $t = 1$. Thus, it has one unit of unfinished work at $t = 2$. However, the optimum could schedule the size 2 job at time 0 and finish all these jobs by time 2. Now, at time $t = 2$, three more jobs with sizes 1/2, 1/2, and 1 arrive. Again, SRPT will work on size 1/2 jobs first, and it can be seen that it will have two unfinished jobs with remaining work 1/2 each at $t = 3$, whereas the optimum can finish all these jobs by time 3. This pattern is continued by giving three jobs of size 1/4, 1/4, and 1/2 at $t = 3$ and so on. After k steps, SRPT will have k jobs with sizes $1/2, 1/4, 1/8, \ldots, 1/2^{k-2}, 1/2^{k-1}, 1/2^{k-1}$, while the optimum has no jobs remaining. Now the adversary can give 2 jobs of size $1/2^k$ each every $1/2^k$ time units for a long time, which implies that SRPT could be $\Omega(\log P)$ worse than optimum.

Leonardi and Raz also considered off-line algorithms for the non-preemptive setting in their paper.

Theorem 3 ([8]) *There is a polynomial time off-line algorithm that achieves an approximation ratio of* $O(n^{1/2} \log n/m)$ *for minimizing average flow time on m machines without preemption.*

To prove this result, they give a general technique to convert a preemptive schedule to a non-preemptive one at the loss of an $O(n^{1/2})$ factor in the approximation ratio. They also showed an almost matching lower bound. In particular,

Theorem 4 ([8]) *No polynomial time algorithm for minimizing the total flow time on multiple machines without preemption can have an approximation ratio of* $O(n^{1/3-\varepsilon})$ *for any* $\varepsilon > 0$, *unless* $P = NP$.

Extensions

Since the publication of these results, they have been extended in several directions. Recall that SRPT is both preemptive and migratory. Awerbuch, Azar, Leonardi, and Regev [2] gave an online scheduling algorithm that is nonmigratory and still achieves a competitive ratio of $O(\min(\log(n/m), \log P))$. Avrahami and Azar [1] gave an even more restricted $O(\min(\log P, \log(n/m)))$ competitive online algorithm. Their algorithm, in addition to being nonmigratory, dispatches a job immediately to a machine upon its arrival. Recently, Garg and Kumar [4, 5] have extended these results to a setting where machines have nonuniform speeds. Other related problems and settings such as stretch minimization (defined as the flow time divided by the size of a job), weighted flow time minimization, general cost functions such as weighted norms, and the non-clairvoyant setting where the size of a job is not unknown upon its arrival have also been investigated. The reader is referred to the relatively recent survey [9] for more details.

Applications

The flow time measure considered here is one of the most widely used measures of quality of service, as it corresponds to the amount of time one has to wait to get the job done. The scheduling model considered here arises very naturally when there are multiple resources and several agents that compete for service from these resources. For example, consider a computing system with multiple homogeneous processors where jobs are submitted by users arbitrarily over time. Keeping the average response time low also keeps the frustration levels of the users low. The model is not necessarily limited to computer systems. At a grocery store, each cashier can be viewed as a machine, and the users lining up to check out can be viewed as jobs. The flow time of a user is time spent waiting until she finishes her transaction with the cashier. Of course, in many applications, there are additional constraints such as it may be infeasible to preempt jobs or, if customers expect a certain fairness, such people might prefer to be serviced in a first-come-first-served manner at a grocery store.

Open Problems

The online algorithm of Leonardi and Raz is also the best-known off-line approximation algorithm for the problem. In particular, it is not known whether an $O(1)$ approximation exists even for the case of two machines. Settling this would be very interesting. In related work, Bansal [3] considered the problem of finding nonmigratory schedules for a constant number of machines. He gave an algorithm that produces a $(1 + \varepsilon)$-approximate solution for any $\varepsilon > 0$ in time $n^{O(\log n/\varepsilon 2)}$. This suggests the possibility of a polynomial time approximation scheme for the problem, at least for the case of a constant number of machines.

Cross-References

- Flow Time Minimization
- Multilevel Feedback Queues
- Shortest Elapsed Time First Scheduling

Recommended Reading

1. Avrahami N, Azar Y (2003) Minimizing total flow time and completion time with immediate dispacthing. In: Proceedings of 15th SPAA, pp 11–18

2. Awerbuch B, Azar Y, Leonardi S, Regev O (2002) Minimizing the flow time without migration. SIAM J Comput 31:1370–1382
3. Bansal N (2005) Minimizing flow time on a constant number of machines with preemption. Oper Res Lett 33:267–273
4. Garg N, Kumar A (2006) Better algorithms for minimizing average flow-time on related machines. In: Proceesings of ICALP, pp 181–190
5. Garg N, Kumar A (2006) Minimizing average flow time on related machines. In: ACM symposium on theory of compuring (STOC), pp 730–738
6. Kellerer H, Tautenhahn T, Woeginger GJ (1999) Approximability and nonapproximability results for minimizing total flow time on a single machine. SIAM J Comput 28:1155–1166
7. Leonardi S (2003) A simpler proof of preemptive flow-time approximation. In: Bampis E (ed) Approximation and on-line algorithms. Lecture notes in computer science. Springer, Berlin
8. Leonardi S, Raz D (1997) Approximating total flow time on parallel machines. In: ACM symposium on theory of computing (STOC), pp 110–119
9. Pruhs K, Sgall J, Torng E (2004) Online scheduling. In: Handbook on scheduling: algorithms, models and performance analysis. CRC, Boca Raton. Symposium on theory of computing (STOC), 1997, pp 110–119

Minimum Geometric Spanning Trees

Christos Levcopoulos
Department of Computer Science, Lund University, Lund, Sweden

Keywords

EMST; Euclidean minimum spanning trees; Minimum length spanning trees; Minimum weight spanning trees; MST

Years and Authors of Summarized Original Work

1999; Krznaric, Levcopoulos, Nilsson

Problem Definition

Let S be a set of n points in d-dimensional real space where $d \geq 1$ is an integer constant. A *minimum spanning tree* (MST) of S is a connected acyclic graph with vertex set S of minimum total edge length. The length of an edge equals the distance between its endpoints under some metric. Under the so-called L_p metric, the distance between two points x and y with coordinates $(x_1, x_2, \ldots, x_d)$ and $(y_1, y_2, \ldots, y_d)$, respectively, is defined as the pth root of the sum $\sum_{i=1}^{d} |x_i - y_i|^p$.

Key Results

Since there is a very large number of papers concerned with geometric MSTs, only a few of them will be mentioned here.

In the common Euclidean L_2 metric, which simply measures straight-line distances, the MST problem in two dimensions can be solved optimally in time $O(n \log n)$, by using the fact that the MST is a subgraph of the Delaunay triangulation of the input point set. The latter is in turn the dual of the Voronoi diagram of S, for which there exist several $O(n \log n)$-time algorithms. The term "optimally" here refers to the algebraic computation tree model. After computation of the Delaunay triangulation, the MST can be computed in only $O(n)$ additional time, by using a technique by Cheriton and Tarjan [6].

Even for higher dimensions, i.e., when $d > 2$, it holds that the MST is a subgraph of the dual of the Voronoi diagram; however, this fact cannot be exploited in the same way as in the two-dimensional case, because this dual may contain $\Omega(n^2)$ edges. Therefore, in higher dimensions, other geometric properties are used to reduce the number of edges which have to be considered. The first subquadratic-time algorithm for higher dimensions was due to Yao [15]. A more efficient algorithm was later proposed by Agarwal et al. [1]. For $d = 3$, their algorithm runs in randomized expected time $O((n \log n)^{4/3})$ and for $d \geq 4$, in expected time $O\left(n^{2-2/(\lceil d/2 \rceil)+1+\epsilon}\right)$, where ϵ stands for an arbitrarily small positive constant.

The algorithm by Agarwal et al. builds on exploring the relationship between computing an MST and finding a closest pair between n red points and m blue points, which is called the *bichromatic closest pair* problem. They

showed that if $T_d(n,m)$ denotes the time to solve the latter problem, then an MST can be computed in $O(T_d(n,n)\log^d n)$ time. Later, Callahan and Kosaraju [4] improved this bound to $O(T_d(n,n)\log n)$. Both methods achieve running time $O(T_d(n,n))$, if $T_d(n,n) = \Omega(n^{1+\alpha})$, for some $\alpha > 0$. Finally, Krznaric et al. [11] showed that the two problems, i.e., computing an MST and computing the bichromatic closest pair, have the same worst-case time complexity (up to constant factors) in the commonly used algebraic computation tree model and for any fixed L_p metric. The hardest part to prove is that an MST can be computed in time $O(T_d(n,n))$. The other part, which is that the bichromatic closest pair problem is not harder than computing the MST, is easy to show: if one first computes an MST for the union of the $n + m$ red and blue points, one can then find a closest bichromatic pair in linear time, because at least one such pair has to be connected by some edge of the MST.

The algorithm proposed by Krznaric et al. [11] is based on the standard approach of joining trees in a forest with the shortest edge connecting two different trees, similar to the classical Kruskal's and Prim's MST algorithms for graphs. To reduce the number of candidates to be considered as edges of the MST, the algorithm works in a sequence of phases, where in each phase only edges of equal or similar length are considered, within a factor of 2.

The initial forest is the set S of points, that is, each point of the input constitutes an individual edgeless tree. Then, as long as there is more than one tree in the forest, two trees are merged by producing an edge connecting two nodes, one from each tree. After this procedure, the edges produced comprise a single tree that remains in the forest, and this tree constitutes the output of the algorithm.

Assume that the next edge that the algorithm is going to produce has length l. Each tree T in the forest is partitioned into groups of nodes, each group having a specific node representing the group. The representative node in such a group is called a *leader*. Furthermore, every node in a group including the leader has the property that it lies within distance $\epsilon \cdot l$ from its leader, where ε is a real constant close to zero.

Instead of considering all pairs of nodes which can be candidates for the next edge to produce, first, only pairs of leaders are considered. Only if a pair of leaders belong to different trees and the distance between them is approximately l, then the closest pair of points between their two respective groups is computed, using the algorithm for the bichromatic closest pair problem.

Also, the following invariant is maintained: for any phase producing edges of length $\Theta(l)$ and for any leader, there is only a constant number of other leaders at distance $\Theta(l)$. Thus, the total number of pairs of leaders to consider is only linear in the number of leaders.

Nearby leaders for any given leader can be found efficiently by using bucketing techniques and data structures for dynamic closest pair queries [3], together with extra artificial points which can be inserted and removed for probing purposes at various small boxes at distance $\Theta(l)$ from the leader. In order to maintain the invariant, when moving to subsequent phases, one reduces the number of leaders accordingly, as pairs of nearby groups merge into single groups. Another tool which is also needed to consider the right types of pairs is to organize the groups according to the various directions in which there can be new candidate MST edges adjacent to nodes in the group. For details, please see the original paper by Krznaric et al. [11].

There is a special version of the bichromatic closest point problem which was shown by Krznaric et al. [11] to have the same worst-case time complexity as computing an MST, namely, the problem for the special case when both the set of red points and the set of blue points have a very small diameter compared with the distance between the closest bichromatic pair. This ratio can be made arbitrarily small by choosing a suitable ε as the parameter for creating the groups and leaders mentioned above. This fact was exploited in order to derive more efficient algorithms for the three-dimensional case.

For example, in the L_1 metric, it is possible to build in time $O(n\log n)$ a special kind of a planar Voronoi diagram for the blue points on a plane separating the blue from the red points having the following property: for each query point q in the half-space including the red points, one can

use this Voronoi diagram to find in time $O(\log n)$ the blue point which is closest to q under the L_1 metric. (This planar Voronoi diagram can be seen as defined by the vertical projections of the blue points onto the plane containing the diagram, and the size of a Voronoi cell depends on the distance between the corresponding blue point and the plane.) So, by using subsequently every red point as a query point for this data structure, one can solve the bichromatic closest pair problem for such well-separated red-blue sets in total $O(n \log n)$ time.

By exploiting and building upon this idea, Krznaric et al. [11] showed how to find an MST of S in optimal $O(n \log n)$ time under the L_1 and L_∞ metrics when $d = 3$. This is an improvement over previous bounds due to Gabow et al. [10] and Bespamyatnikh [2], who proved that, for $d = 3$, an MST can be computed in $O(n \log n \log \log n)$ time under the L_1 and L_∞ metrics.

The main results of Krznaric et al. [11] are summarized in the following theorem.

Theorem *In the algebraic computation tree model, for any fixed L_p metric and for any fixed number of dimensions, computing the MST has the same worst-case complexity, within constant factors, as solving the bichromatic closest pair problem. Moreover, for three-dimensional space under the L_1 and L_∞ metrics, the MST (as well as the bichromatic closest pair) can be computed in optimal $O(n \log n)$ time.*

Approximate and Dynamic Solutions

Callahan and Kosaraju [4] showed that a spanning tree of length within a factor $1 + \epsilon$ from that of an MST can be computed in time $O(n(\log n + \epsilon^{-d/2} \log \epsilon^{-1}))$. Approximation algorithms with worse trade-off between time and quality had earlier been developed by Clarkson [7], Vaidya [14] and Salowe [13]. In addition, if the input point set is supported by certain basic data structures, then the approximate length of an MST can be computed in randomized sublinear time [8]. Eppstein [9] gave fully dynamic algorithms that maintain an MST when points are inserted or deleted.

Applications

MSTs belong to the most basic structures in computational geometry and in graph theory, with a vast number of applications.

Open Problems

Although the complexity of computing MSTs is settled in relation to computing bichromatic closest pairs, this means also that it remains open for all cases where the complexity of computing bichromatic closest pairs remains open, e.g., when the number of dimensions is greater than 3.

Experimental Results

Narasimhan and Zachariasen [12] have reported experiments with computing geometric MSTs via well-separated pair decompositions. More recent experimental results are reported by Chatterjee, Connor, and Kumar [5].

Cross-References

- Minimum Spanning Trees
- Parallel Connectivity and Minimum Spanning Trees
- Randomized Minimum Spanning Tree
- Steiner Trees

Recommended Reading

1. Agarwal PK, Edelsbrunner H, Schwarzkopf O, Welzl E (1991) Euclidean minimum spanning trees and bichromatic closest pairs. Discret Comput Geom 6:407–422
2. Bespamyatnikh S (1997) On constructing minimum spanning trees in R_1^k. Algorithmica 18(4): 524–529
3. Bespamyatnikh S (1998) An optimal algorithm for closest-pair maintenance. Discret Comput Geom 19(2):175–195
4. Callahan PB, Kosaraju SR (1993) Faster algorithms for some geometric graph problems in higher dimensions. In: SODA, Austin, pp 291–300

5. Chatterjee S, Connor M, Kumar P (2010) Geometric minimum spanning trees with GeoFilterKruskal. In: Experimental Algorithms: proceedings of SEA, Ischia Island. LNCS, vol 6049, pp 486–500
6. Cheriton D, Tarjan RE (1976) Finding minimum spanning trees. SIAM J Comput 5(4):724–742
7. Clarkson KL (1984) Fast expected-time and approximation algorithms for geometric minimum spanning trees. In: Proceedings of STOC, Washington, DC, pp 342–348
8. Czumaj A, Ergün F, Fortnow L, Magen A, Newman I, Rubinfeld R, Sohler C (2005) Approximating the weight of the Euclidean minimum spanning tree in sublinear time. SIAM J Comput 35(1):91–109
9. Eppstein D (1995) Dynamic Euclidean minimum spanning trees and extrema of binary functions. Discret Comput Geom 13:111–122
10. Gabow HN, Bentley JL, Tarjan RE (1984) Scaling and related techniques for geometry problems. In: STOC, Washington, DC, pp 135–143
11. Krznaric D, Levcopoulos C, Nilsson BJ (1999) Minimum spanning trees in d dimensions. Nord J Comput 6(4):446–461
12. Narasimhan G, Zachariasen M (2001) Geometric minimum spanning trees via well-separated pair decompositions. ACM J Exp Algorithms 6:6
13. Salowe JS (1991) Constructing multidimensional spanner graphs. Int J Comput Geom Appl 1(2):99–107
14. Vaidya PM (1988) Minimum spanning trees in k-Dimensional space. SIAM J Comput 17(3):572–582
15. Yao AC (1982) On constructing minimum spanning trees in k-Dimensional spaces and related problems. SIAM J Comput 11(4):721–736

Minimum *k*-Connected Geometric Networks

Artur Czumaj[1] and Andrzej Lingas[2]
[1]Department of Computer Science, Centre for Discrete Mathematics and Its Applications, University of Warwick, Coventry, UK
[2]Department of Computer Science, Lund University, Lund, Sweden

Keywords

Geometric networks; Geometric optimization; k-connectivity; Network design; Survivable network design

Synonyms

Euclidean graphs; Geometric graphs

Years and Authors of Summarized Original Work

2000; Czumaj, Lingas

Problem Definition

The following classical optimization problem is considered: for a given undirected weighted geometric network, find its minimum-cost subnetwork that satisfies a priori given multi-connectivity requirements. This problem restricted to *geometric networks* is considered in this entry.

Notations

Let $G = (V, E)$ be a geometric network, whose vertex set V corresponds to a set of n points in $\mathbb{R}^d$ for certain integer d, $d \geq 2$ and whose edge set E corresponds to a set of straight-line segments connecting pairs of points in V. G is called complete if E connects all pairs of points in V.

The cost $\delta(x, y)$ of an edge connecting a pair of points $x, y \in \mathbb{R}^d$ is equal to the Euclidean distance between points x and y, that is, $\delta(x, y) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$, where $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$. More generally, the cost $\delta(x, y)$ could be defined using other norms, such as ℓ_p norms for any $p > 1$, i.e., $\delta(x, y) = \left(\sum_{i=1}^{p}(x_i - y_i)^p\right)^{1/p}$. The cost of the network is equal to the sum of the costs of the edges of the network, $\text{cost}(G) = \sum_{(x,y)\in E} \delta(x, y)$.

A network $G = (V, E)$ *spans* a set S of points if $V = S$. $G = (V, E)$ is *k-vertex connected* if for any set $U \subseteq V$ of fewer than k vertices, the network $(V \setminus U,\ E \cap ((V \setminus U) \times (V \setminus U)))$ is connected. Similarly, G is *k-edge connected* if for any set $\mathcal{E} \subseteq E$ of fewer than k edges, the network $(V, E \setminus \mathcal{E})$ is connected.

The *(Euclidean) minimum-cost k-vertex-connected spanning network problem*: for a given set S of n points in the Euclidean space $\mathbb{R}^d$, find a minimum-cost k-vertex-connected Euclidean network spanning points in S.

The *(Euclidean) minimum-cost k-edge-connected spanning network problem*: for a given set S of n points in the Euclidean space $\mathbb{R}^d$, find a minimum-cost k-edge-connected Euclidean network spanning points in S.

A variant that allows *parallel edges* is also considered:

The *(Euclidean) minimum-cost k-edge-connected spanning multi-network problem*: for a given set S of n points in the Euclidean space $\mathbb{R}^d$, find a minimum-cost k-edge-connected Euclidean multi-network spanning points in S (where the multi-network can have parallel edges).

The concept of minimum-cost k-connectivity naturally extends to include that of *Euclidean Steiner k-connectivity* by allowing the use of additional vertices, called *Steiner points*. For a given set S of points in $\mathbb{R}^d$, a geometric network G is a *Steiner k-vertex connected (or Steiner k-edge connected) for* S if the vertex set of G is a *superset* of S and for every pair of points from S there are k internally vertex-disjoint (edge-disjoint, respectively) paths connecting them in G.

The *(Euclidean) minimum-cost Steiner k-vertex/edge connectivity* problem: find a minimum-cost network on a superset of S that is Steiner k-vertex/edge connected for S.

Note that for $k = 1$, it is simply the *Steiner minimal tree* problem, which has been very extensively studied in the literature (see, e.g., [15]).

In a more general formulation of multi-connectivity graph problems, nonuniform connectivity constraints have to be satisfied.

The *survivable network design* problem: for a given set S of points in $\mathbb{R}^d$ and a connectivity requirement function $r : S \times S \to \mathbb{N}$, find a minimum-cost geometric network spanning points in S such that for any pair of vertices $p, q \in S$ the subnetwork has $r_{p,q}$ internally vertex-disjoint (or edge-disjoint, respectively) paths between p and q.

In many applications of this problem, often regarded as the most interesting ones [10, 14], the connectivity requirement function is specified with the help of a one-argument function which assigns to each vertex p its connectivity type $r_v \in \mathbb{N}$. Then, for any pair of vertices $p, q \in S$, the connectivity requirement $r_{p,q}$ is simply given as $\min\{r_p, r_q\}$ [13, 14, 18, 19]. This includes the *Steiner tree problem* (see, e.g., [2]), in which $r_p \in \{0, 1\}$ for any vertex $p \in S$.

A *polynomial-time approximation scheme (PTAS)* is a family of algorithms $\{\mathcal{A}_\varepsilon\}$ such that, for each fixed $\varepsilon > 0$, $\mathcal{A}_\varepsilon$ runs in time polynomial in the size of the input and produces a $(1 + \varepsilon)$-approximation.

Related Work

For a very extensive presentation of results concerning problems of finding minimum-cost k-vertex- and k-edge-connected spanning subgraphs, nonuniform connectivity, connectivity augmentation problems, and geometric problems, see [1, 3, 4, 12, 16].

Despite the practical relevance of the multi-connectivity problems for geometrical networks and the vast amount of practical heuristic results reported (see, e.g., [13, 14, 18, 19]), very little theoretical research had been done towards developing efficient approximation algorithms for these problems until a few years ago. This contrasts with the very rich and successful theoretical investigations of the corresponding problems in general metric spaces and for general weighted graphs. And so, until 1998, even for the simplest and most fundamental multi-connectivity problem that of finding a minimum-cost 2-vertex-connected network spanning a given set of points in the Euclidean plane, obtaining approximations achieving better than a $\frac{3}{2}$ ratio had been elusive (the ratio $\frac{3}{2}$ is the best polynomial-time approximation ratio known for general networks whose weights satisfy the triangle inequality [9]; for other results, see, e.g., [5, 16]).

Key Results

The first result is an extension of the well-known $\mathcal{NP}$-hardness result of minimum-cost 2-connectivity in general graphs (see, e.g., [11]) to geometric networks.

Theorem 1 *The problem of finding a minimum-cost 2-vertex-/2-edge-connected geometric network spanning a set of n points in the plane is $\mathcal{NP}$-hard.*

Next result shows that if one considers the minimum-cost multi-connectivity problems in an enough high dimension, the problems become APX-hard.

Theorem 2 ([6]) *There exists a constant $\xi > 0$ such that it is $\mathcal{NP}$-hard to approximate within $1 + \xi$ the minimum-cost 2-connected geometric network spanning a set of n points in $\mathbb{R}^{\lceil \log_2 n \rceil}$*

This result extends also to any ℓ_p norm.

Theorem 3 ([6]) *For integer $d \geq \log n$ and for any fixed $p \geq 1$, there exists a constant $\xi > 0$ such that it is $\mathcal{NP}$-hard to approximate within $1 + \xi$ the minimum-cost 2-connected network spanning a set of n points in the ℓ_p metric in $\mathbb{R}^d$.*

Since the minimum-cost multi-connectivity problems are hard, the research turned into the study of approximation algorithms. By combining some of the ideas developed for the polynomial-time approximation algorithms for TSP due to Arora [2] (see also [17]) together with several new ideas developed specifically for the multi-connectivity problems in geometric networks, Czumaj and Lingas obtained the following results.

Theorem 4 ([6,7]) *Let k and d be any integers, $k, d \geq 2$, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm that in time $n \cdot (\log n)^{(kd/\varepsilon)^{\mathcal{O}(d)}} \cdot 2^{2^{(kd/\varepsilon)^{\mathcal{O}(d)}}}$ with probability at least 0.99 finds a k-vertex-connected (or k-edge-connected) spanning network for S whose cost is at most $(1 + \varepsilon)$-time optimal.*

Furthermore, this algorithm can be derandomized in polynomial time to return a k-vertex-connected (or k-edge-connected) spanning network for S whose cost is at most $(1 + \varepsilon)$ times the optimum.

Observe that when all d, k, and ε are constant, then the running times are $n \cdot \log^{\mathcal{O}(1)} n$.

The results in Theorem 4 give a PTAS for small values of k and d.

Theorem 5 (PTAS for vertex/edge connectivity [6,7]) *Let $d \geq 2$ be any constant integer. There is a certain positive constant $c < 1$ such that for all k such that $k \leq (\log \log n)^c$, the problems of finding a minimum-cost k-vertex-connected spanning network and a k-edge-connected spanning network for a set of points in $\mathbb{R}^d$ admit PTAS.*

The next theorem deals with multi-networks where feasible solutions are allowed to use parallel edges.

Theorem 6 ([7]) *Let k and d be any integers, $k, d \geq 2$, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm that in time $n \cdot \log n \cdot (d/\varepsilon)^{\mathcal{O}(d)} + n \cdot 2^{2^{(k^{\mathcal{O}(1)} \cdot (d/\varepsilon)^{\mathcal{O}(d^2)})}}$, with probability at least 0.99 finds a k-edge-connected spanning multi-network for S whose cost is at most $(1 + \varepsilon)$ times the optimum. The algorithm can be derandomized in polynomial time.*

Combining this theorem with the fact that parallel edges can be eliminated in case $k = 2$, one obtains the following result for 2-connectivity in networks.

Theorem 7 (Approximation schemes for 2-connected graphs, [7]) *Let d be any integer, $d \geq 2$, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm that in time $n \cdot \log n \cdot (d/\varepsilon)^{\mathcal{O}(d)} + n \cdot 2^{(d/\varepsilon)^{\mathcal{O}(d^2)}}$ with probability at least 0.99 finds a 2-vertex-connected (or 2-edge-connected) spanning network for S whose cost is at most $(1 + \varepsilon)$ times the optimum. This algorithm can be derandomized in polynomial time.*

For constant d, the running time of the randomized algorithms is $n \log n \cdot (1/\varepsilon)^{\mathcal{O}(1)} + 2^{(1/\varepsilon)^{\mathcal{O}(1)}}$.

Theorem 8 ([8]) *Let d be any integer, $d \geq 2$, and let ε be any positive real. Let S be a set*

of n points in $\mathbb{R}^d$. There is a randomized algorithm that in time $n \cdot \log n \cdot (d/\varepsilon)^{\mathcal{O}(d)} + n \cdot 2^{(d/\varepsilon)^{\mathcal{O}(d^2)}} + n \cdot 2^{2^{d^{\mathcal{O}(1)}}}$ with probability at least 0.99 finds a Steiner 2-vertex-connected (or 2-edge-connected) spanning network for S whose cost is at most $(1+\varepsilon)$ times the optimum. This algorithm can be derandomized in polynomial time.

Theorem 9 ([8]) *Let d be any integer, $d \geq 2$, and let ε be any positive real. Let S be a set of n points in $\mathbb{R}^d$. There is a randomized algorithm that in time $n \cdot \log n \cdot (d/\varepsilon)^{\mathcal{O}(d)} + n \cdot 2^{(d/\varepsilon)^{\mathcal{O}(d^2)}} + n \cdot 2^{2^{d^{\mathcal{O}(1)}}}$ with probability at least 0.99 gives a $(1+\varepsilon)$-approximation for the geometric network survivability problem with $r_v \in \{0, 1, 2\}$ for any $v \in V$. This algorithm can be derandomized in polynomial time.*

Applications

Multi-connectivity problems are central in algorithmic graph theory and have numerous applications in computer science and operation research, see, e.g., [1, 12, 14, 19]. They also play very important role in the design of networks that arise in practical situations, see, e.g., [1, 14]. Typical application areas include telecommunication, computer, and road networks. Low degree connectivity problems for geometrical networks in the plane can often closely *approximate* such practical connectivity problems (see, e.g., the discussion in [14, 18, 19]). The survivable network design problem in geometric networks also arises in many applications, e.g., in telecommunication, communication network design, VLSI design, etc. [13, 14, 18, 19].

Open Problems

The results discussed above lead to efficient algorithms only for small connectivity requirements k; the running time is polynomial only for the value of k up to $(\log \log n)^c$ for certain positive constant $c < 1$. It is an interesting open problem if one can obtain polynomial-time approximation scheme algorithms also for large values of k.

It is also an interesting open problem if the multi-connectivity problems in geometric networks can have practically fast approximation schemes.

Cross-References

- Euclidean Traveling Salesman Problem
- Minimum Geometric Spanning Trees

Recommended Reading

1. Ahuja RK, Magnanti TL, Orlin JB, Reddy MR (1995) Applications of network optimization. In: Ball MO, Magnanti TL, Monma CL and Nemhauser GL (eds) Handbooks in operations research and management science, vol 7: network models chapter 1. North-Holland, pp 1–83
2. Arora S (1998) Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J ACM 45(5):753–782
3. Berger A, Czumaj A, Grigni M, Zhao H (2005) Approximation schemes for minimum 2-connected spanning subgraphs in weighted planar graphs. In: Proceedings of the 13th Annual European Symposium on Algorithms (ESA), Palma de Mallorca, pp 472–483
4. Borradaile G, Klein PN (2008) The two-edge connectivity survivable network problem in planar graphs. In: Proceedings of the 35th Annual International Colloquium on Automata Languages and Programming (ICALP), Reykjavik, pp 485–501
5. Cheriyan J, Vetta A (2005) Approximation algorithms for network design with metric costs. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC), Baltimore, pp 167–175
6. Czumaj A, Lingas A (1999) On approximability of the minimum-cost k-connected spanning subgraph problem. In: Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Baltimore, pp 281–290
7. Czumaj A, Lingas A (2000) Fast approximation schemes for Euclidean multi-connectivity problems. In: Proceedings of the 27th Annual International Colloquium on Automata, Languages and Programming (ICALP), Geneva, pp 856–868
8. Czumaj A, Lingas A, Zhao H (2002) Polynomial-time approximation schemes for the Euclidean survivable network design problem. In: Proceedings of the 29th Annual International Colloquium on Automata, Languages and Programming (ICALP), Málaga, pp 973–984
9. Frederickson GN, JáJá J (1982) On the relationship between the biconnectivity augmentation and

traveling salesman problem. Theor Comput Sci 19(2):189–201
10. Gabow HN, Goemans MX, Williamson DP (1998) An efficient approximation algorithm for the survivable network design problem. Math Progr Ser B 82(1–2):13–40
11. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, New York
12. Goemans MX, Williamson DP (1996) The primal-dual method for approximation algorithms and its application to network design problems. In: Hochbaum D (ed) Approximation algorithms for $\mathcal{NP}$-hard problems, chapter 4. PWS, Boston, pp 144–191
13. Grötschel M, Monma CL, Stoer M (1992) Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. Oper Res 40(2):309–330
14. Grötschel M, Monma CL, Stoer M (1995) Design of survivable networks. In: Handbooks in operations research and management science, vol 7: network models, chapter 10. North-Holland, pp 617–672
15. Hwang FK, Richards DS, Winter P (1992) The Steiner tree problem. North-Holland, Amsterdam
16. Khuller S (1996) Approximation algorithms for finding highly connected subgraphs. In: Hochbaum D (ed) Approximation algorithms for $\mathcal{NP}$-hard problems, chapter 6. PWS, Boston, pp 236–265
17. Mitchell JSB (1999) Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. SIAM J Comput 28(4):1298–1309
18. Monma CL, Shallcross DF (1989) Methods for designing communications networks with certain two-connected survivability constraints. Oper Res 37(4):531–541
19. Stoer M (1992) Design of survivable networks. Springer, Berlin

Minimum Spanning Trees

Seth Pettie
Electrical Engineering and Computer Science (EECS) Department, University of Michigan, Ann Arbor, MI, USA

Keywords

Minimal spanning tree; Minimum weight spanning tree; Shortest spanning tree

Years and Authors of Summarized Original Work

2002; Pettie, Ramachandran

Problem Definition

The *minimum spanning tree* (MST) problem is, given a connected, weighted, and undirected graph $G = (V, E, w)$, to find the tree with minimum total *weight* spanning all the vertices V. Here, $w : E \to \mathbb{R}$ is the weight function. The problem is frequently defined in geometric terms, where V is a set of points in d-dimensional space and w corresponds to Euclidean distance. The main distinction between these two settings is the form of the input. In the graph setting, the input has size $O(m+n)$ and consists of an enumeration of the $n = |V|$ vertices and $m = |E|$ edges and edge weights. In the geometric setting, the input consists of an enumeration of the coordinates of each point ($O(dn)$ space): all $\binom{V}{2}$ edges are implicitly present and their weights implicit in the point coordinates. See [16] for a discussion of the Euclidean minimum spanning tree problem.

History

The MST problem is generally recognized [7,12] as one of the first combinatorial problems studied specifically from an algorithmic perspective. It was formally defined by Borůvka in 1926 [1] (predating the fields of computability theory and combinatorial optimization and even much of graph theory), and since his initial algorithm, there has been a sustained interest in the problem. The MST problem has motivated research in matroid optimization [3] and the development of efficient data structures, particularly priority queues (aka heaps) and disjoint set structures [2,18].

Related Problems

The MST problem is frequently contrasted with the *traveling salesman* and *minimum Steiner tree* problems [6]. A Steiner tree is a tree that may span any *superset* of the given points; that is,

additional points may be introduced that reduce the weight of the minimum spanning tree. The traveling salesman problem asks for a tour (cycle) of the vertices with minimum total length. The generalization of the MST problem to directed graphs is sometimes called the *minimum branching* [5]. Whereas the undirected and directed versions of the MST problem are solvable in polynomial time, traveling salesman and minimum Steiner tree are NP-complete [6].

Optimality Conditions

A *cut* is a partition (V', V'') of the vertices V. An edge (u, v) *crosses* the cut (V', V'') if $u \in V'$ and $v \in V''$. A sequence $(v_0, v_1, \ldots, v_{k-1}, v_0)$ is a *cycle* if $(v_i, v_{i+1(\bmod k)}) \in E$ for $0 \leq i < k$.

The correctness of all MST algorithms is established by appealing to the dual *cut* and *cycle* properties, also known as the blue rule and red rule [18].

Cut Property An edge is in some minimum spanning tree if and only if it is the lightest edge crossing some cut.

Cycle Property An edge is not in any minimum spanning tree if and only if it is the sole heaviest edge on some cycle.

It follows from the cut and cycle properties that if the edge weights are unique, then there is a unique minimum spanning tree, denoted $\mathbf{MST}(G)$. Uniqueness can always be enforced by breaking ties in any consistent manner. MST algorithms frequently appeal to a useful corollary of the cut and cycle properties called the *contractibility* property. Let $G \backslash C$ denote the graph derived from G by contracting the subgraph C, that is, C is replaced by a single vertex c and all edges incident to exactly one vertex in C become incident to c; in general, $G \backslash C$ may have more than one edge between two vertices.

Contractibility Property If C is a subgraph such that for all pairs of edges e and f with exactly one endpoint in C, there exists a path $P \subseteq C$ connecting e f with each edge in P lighter than either e or f, then C is *contractible*. For any contractible C, it holds that $\mathbf{MST}(G) = \mathbf{MST}(C) \cup \mathbf{MST}(G \backslash C)$.

The Generic Greedy Algorithm

Until recently, all MST algorithms could be viewed as mere variations on the following generic greedy MST algorithm. Let $\mathcal{T}$ consist initially of n trivial trees, each containing a single vertex of G. Repeat the following step $n - 1$ times. Choose any $T \in \mathcal{T}$ and find the minimum weight edge (u, v) with $u \in T$ and v in a different tree, say $T' \in \mathcal{T}$. Replace T and T' in $\mathcal{T}$ with the single tree $T \cup \{(u, v)\} \cup T'$. After $n - 1$ iterations, $\mathcal{T} = \{\mathbf{MST}(G)\}$. By the cut property, every edge selected by this algorithm is in the MST.

Modeling MST Algorithms

Another corollary of the cut and cycle properties is that the set of minimum spanning trees of a graph is determined solely by the relative order of the edge weights – their specific numerical values are not relevant. Thus, it is natural to model MST algorithms as *binary decision trees*, where nodes of the decision tree are identified with edge weight comparisons and the children of a node correspond to the possible outcomes of the comparison. In this decision tree model, a trivial lower bound on the *time* of the optimal MST algorithm is the *depth* of the optimal decision tree.

Key Results

The primary result of [14] is an explicit MST algorithm that is *provably* optimal even though its asymptotic running time is currently unknown.

Theorem 1 *There is an explicit, deterministic minimum spanning tree algorithm whose running time is on the order of $D_{MST}(m, n)$, where m is the number of edges, n the number of vertices, and $D_{MST}(m, n)$ the maximum depth of an optimal decision tree for any m-edge n-node graph.*

It follows that the Pettie-Ramachandran algorithm [14] is asymptotically no worse than *any* MST algorithm that deduces the solution through edge weight comparisons. The best known upper bound on $D_{\mathrm{MST}}(m, n)$ is $O(m\alpha(m, n))$, due to Chazelle [2]. It is trivially $\Omega(m)$.

Let us briefly describe how the Pettie-Ramachandran algorithm works. An (m, n) instance is a graph with m edges and n vertices. Theorem 1 is proved by giving a linear time *decomposition* procedure that reduces any (m, n) instance of the MST problem to instances of size $(m^*, n^*), (m_1, n_1), \ldots, (m_s, n_s)$, where $m = m^* + \sum_i m_i$, $n = \sum_i n_i$, $n^* \leq n/\log\log\log n$, and each $n_i \leq \log\log\log n$. The (m^*, n^*)instance can be solved in $O(m+n)$ time with existing MST algorithms [2]. To solve the other instances, the Pettie-Ramachandran algorithm performs a brute-force search to find the minimum depth decision tree for *every* graph with at most log log log n vertices. Once these decision trees are found, the remaining instances are solved in $O(\sum_i D_{\text{MST}}(m_i, n_i)) = O(D_{\text{MST}}(m, n))$ time. Due to the restricted size of these instances ($n_i \leq \log\log\log n$), the time for a brute-force search is a negligible $o(n)$. The decomposition procedure makes use of Chazelle's *soft heap* [2] (an approximate priority queue) and an extension of the contractibility property.

Approximate Contractibility Let G' be derived from G by *increasing* the weight of some edges. If C is contractible w.r.t. G', then $\mathbf{MST}(G) = \mathbf{MST}(\mathbf{MST}(C) \cup \mathbf{MST}(G \backslash C) \cup E^*)$, where $E^*$is the set of edges with increased weights.

A secondary result of [14] is that the running time of the optimal algorithm is actually linear on nearly every graph topology, under any permutation of the edge weights.

Theorem 2 *Let G be selected uniformly at random from the set of all n-vertex, m-edge graphs. Then regardless of the edge weights, $\mathbf{MST}(G)$ can be found in $O(m+n)$ time with probability $1 - 2^{-\Omega(m/\alpha 2)}$, where $\alpha = \alpha(m, n)$ is the slowly growing inverse Ackermann function.*

Theorem 1 should be contrasted with the results of Karger, Klein, and Tarjan [9] and Chazelle [2] on the randomized and deterministic complexity of the MST problem.

Theorem 3 ([9]) *The minimum spanning forest of a graph with m edges can be computed by a randomized algorithm in $O(m)$ time with probability $1 - 2^{-\Omega(m)}$.*

Theorem 4 ([2]) *The minimum spanning tree of a graph can be computed in $O(m\alpha(m, n))$ time by a deterministic algorithm, where α is the inverse Ackermann function.*

Applications

Borůvka [1] invented the MST problem while considering the practical problem of electrifying rural Moravia (present-day Czech Republic) with the shortest electrical network. MSTs are used as a starting point for heuristic approximations to the optimal traveling salesman tour and optimal Steiner tree, as well as other network design problems. MSTs are a component in other graph optimization algorithms, notably the single-source shortest path algorithms of Thorup [19] and Pettie-Ramachandran [15]. MSTs are used as a tool for visualizing data that is presumed to have a tree structure; for example, if a matrix contains dissimilarity data for a set of species, the minimum spanning tree of the associated graph will presumably group closely related species; see [7]. Other modern uses of MSTs include modeling physical systems [17] and image segmentation [8]; see [4] for more applications.

Open Problems

The chief open problem is to determine the *deterministic* complexity of the minimum spanning tree problem. By Theorem 1, this is tantamount to determining the decision tree complexity of the MST problem.

Experimental Results

Moret and Shapiro [11] evaluated the performance of greedy MST algorithms using a variety

of priority queues. They concluded that the best MST algorithm is Jarník's [7] (also attributed to Prim and Dijkstra; see [3, 7, 12]) as implemented with a pairing heap [13]. Katriel, Sanders, and Träff [10] designed and implemented a non-greedy randomized MST algorithm based on that of Karger et al. [9]. They concluded that on moderately dense graphs, it runs substantially faster than the greedy algorithms tested by Moret and Shapiro.

Cross-References

▸ Randomized Minimum Spanning Tree

Recommended Reading

1. Borůvka O (1926) O jistém problému minimálním. Práce Moravské Přírodovědecké Společnosti 3:37–58. In Czech
2. Chazelle B (2000) A minimum spanning tree algorithm with inverse-Ackermann type complexity. J ACM 47(6):1028–1047
3. Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms. MIT, Cambridge
4. Eppstein D, Geometry in action: minimum spanning trees. http://www.ics.uci.edu/~eppstein/gina/mst.html. Last downloaded Nov 2014
5. Gabow HN, Galil Z, Spencer TH, Tarjan RE (1986) Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica 6:109–122
6. Garey MR, Johnson DS (1979) Computers and intractability: a guide to NP-completeness. Freeman, San Francisco
7. Graham RL, Hell P (1985) On the history of the minimum spanning tree problem. Ann Hist Comput 7(1):43–57
8. Ion A, Kropatsch WG, Haxhimusa Y (2006) Considerations regarding the minimum spanning tree pyramid segmentation method. In: Proceedings of the 11th workshop structural, syntactic, and statistical pattern recognition (SSPR), Hong Kong. LNCS, vol 4109. Springer, Berlin, pp 182–190
9. Karger DR, Klein PN, Tarjan RE (1995) A randomized linear-time algorithm for finding minimum spanning trees. J ACM 42:321–329
10. Katriel I, Sanders P, Träff JL (2003) A practical minimum spanning tree algorithm using the cycle property. In: Proceedings of the 11th annual European symposium on algorithms, Budapest. LNCS, vol 2832. Springer, Berlin, pp 679–690
11. Moret BME, Shapiro HD (1994) An empirical assessment of algorithms for constructing a minimum spanning tree. In: Computational support for discrete mathematics. DIMACS series in discrete mathematics and theoretical computer science, vol 15. American Mathematical Society, Providence
12. Pettie S (2003) On the shortest path and minimum spanning tree problems. Ph.D. thesis, The University of Texas, Austin
13. Pettie S (2005) Towards a final analysis of pairing heaps. In: Proceedings of the 46th annual symposium on foundations of computer science (FOCS), Pittsburgh, pp 174–183
14. Pettie S, Ramachandran V (2002) An optimal minimum spanning tree algorithm. J ACM 49(1):16–34
15. Pettie S, Ramachandran V (2005) A shortest path algorithm for real-weighted undirected graphs. SIAM J Comput 34(6):1398–1431
16. Preparata FP, Shamos MI (1985) Computational geometry. Springer, New York
17. Subramaniam S, Pope SB (1998) A mixing model for turbulent reactive flows based on euclidean minimum spanning trees. Combust Flame 115(4):487–514
18. Tarjan RE (1983) Data structures and network algorithms. CBMS-NSF regional conference series in applied mathematics, vol 44. SIAM, Philadelphia
19. Thorup M (1999) Undirected single-source shortest paths with positive integer weights in linear time. J ACM 46(3):362–394

M

Minimum Weight Triangulation

Christos Levcopoulos
Department of Computer Science, Lund University, Lund, Sweden

Keywords

Minimum length triangulation

Years and Authors of Summarized Original Work

1998; Levcopoulos, Krznaric

Problem Definition

Given a set S of n points in the Euclidean plane, a *triangulation* T of S is a maximal set of nonintersecting straight-line segments whose

endpoints are in S. The *weight* of T is defined as the total Euclidean length of all edges in T. A triangulation that achieves minimum weight is called a *minimum weight triangulation*, often abbreviated MWT, of S.

Key Results

Since there is a very large number of papers and results dealing with minimum weight triangulation, only relatively very few of them can be mentioned here.

Mulzer and Rote have shown that MWT is NP-hard [12]. Their proof of NP-completeness is not given explicitly; it relies on extensive calculations which they performed with a computer. Remy and Steger have shown a quasi-polynomial time approximation scheme for MWT [13]. These results are stated in the following theorem:

Theorem 1 *The problem of computing the MWT (minimum weight triangulation) of an input set S of n points in the plane is NP-hard. However, for any constant $\epsilon > 0$, a triangulation of S achieving the approximation ratio of $1 + \epsilon$, for an arbitrarily small positive constant ϵ, can be computed in time $n^{O(\log^8 n)}$.*

The complexity status of the symmetric problem of finding the *maximum* weight triangulation is still open, but there exists a quasi-polynomial time approximation scheme for it [10].

The Quasi-Greedy Triangulation Approximates the MWT

Levcopoulos and Krznaric showed that a triangulation of total length within a constant factor of MWT can be computed in polynomial time for arbitrary point sets [7]. The triangulation achieving this result is a modification of the so-called *greedy* triangulation. The greedy triangulation starts with the empty set of diagonals and keeps adding a shortest diagonal not intersecting the diagonals which have already been added, until a full triangulation is produced. The greedy triangulation has been shown to approximate the minimum weight triangulation within a constant factor, unless a special case arises where the greedy diagonals inserted are "climbing" in a special, very unbalanced way along a relatively long concave chain containing many vertices and with a large empty space in front of it, at the same time blocking visibility from another, opposite concave chain of many vertices. In such "bad" cases, the worst-case ratio between the length of the greedy and the length of the minimum weight triangulation is shown to be $\Theta\left(\sqrt{n}\right)$. To obtain a triangulation which always approximates the MWT within a constant factor, it suffices to take care of this special bad case in order to avoid the unbalanced "climbing," and replace it by a more balanced climbing along these two opposite chains. Each edge inserted in this modified method is still almost as short as the shortest diagonal, within a factor smaller than 1.2. Therefore, the modified triangulation which always approximates the MWT is named the *quasi-greedy* triangulation. In a similar way as the original greedy triangulation, the quasi-greedy triangulation can be computed in time $O(n \log n)$ [8]. Gudmundsson and Levcopoulos [5] showed later that a variant of this method can also be parallelized, thus achieving a constant factor approximation of MWT in $O(\log n)$ time, using $O(n)$ processors in the CRCW PRAM model. Another by-product of the quasi-greedy triangulation is that one can easily select in linear time a subset of its edges to obtain a convex partition which is within a constant factor of the minimum length convex partition of the input point set. This last property was crucial in the proof that the quasi-greedy triangulation approximates the MWT. The proof also uses an older result that the (original, unmodified) greedy triangulation of any convex polygon approximates the minimum weight triangulation [9]. Some of the results from [7] and from [8] can be summarized in the following theorem:

Theorem 2 *Let S be an input set of n points in the plane. The quasi-greedy triangulation of S, which is a slightly modified version of the greedy triangulation of S, has total length within a constant factor of the length of the MWT (minimum weight triangulation) of S and can be computed in time $O(n \log n)$. Moreover, the*

(unmodified) greedy triangulation of S has length within $O\left(\sqrt{n}\right)$ of the length of MWT of S, and this bound is asymptotically tight in the worst case.

Computing the Exact Minimum Weight Triangulation

Below, three approaches to compute the exact MWT are shortly discussed. These approaches assume that it is numerically possible to efficiently compare the total length of sets of line segments in order to select the set of smallest weight. This is a simplifying assumption, since this is an open problem per se. However, the problem of computing the exact MWT remains NP-hard even under this assumption [12]. The three approaches differ with respect to the creation and selection of subproblems, which are then solved by dynamic programming.

The first approach, sketched by Lingas [11], employs a general method for computing optimal subgraphs of the complete Euclidean graph. By developing this approach, it is possible to achieve subexponential time $2^{O(\sqrt{n}\log n)}$. The idea is to create the subproblems which are solved by dynamic programming. This is done by trying all (suitable) planar separators of length $O\left(\sqrt{n}\right)$, separating the input point set in a balanced way, and then to proceed recursively within the resulting subproblems.

The second approach uses fixed-parameter algorithms. So, for example, if there are only $O(\log n)$ points in the interior of the convex hull of S, then the MWT of S can be computed in polynomial time [4]. This approach extends also to compute the minimum weight triangulation under the constraint that the outer boundary is not necessarily the convex hull of the input vertices; it can be an arbitrary polygon. Some of these algorithms have been implemented; see Grantson et al. [2] for a comparison of some implementations. These dynamic programming approaches take typically cubic time with respect to the points of the boundary but exponential time with respect to the number of remaining points. So, for example, if k is the number of hole points inside the boundary polygon, then an algorithm, which has also been implemented, can compute the exact MWT in time $O(n^3 \cdot 2^k \cdot k)$ [2].

In an attempt to solve larger problems, a different approach uses properties of MWT which usually help to identify, for random point sets, many edges that *must* be, respectively *cannot* be, in MWT. One can then use dynamic programming to fill in the remaining MWT edges. For random sets consisting of tens of thousands of points from the uniform distribution, one can thus compute the exact MWT in minutes [1].

Applications

The problem of computing a triangulation arises, for example, in finite element analysis, terrain modeling, stock cutting, and numerical approximation [3, 6]. The *minimum weight* triangulation has attracted the attention of many researchers, mainly due to its natural definition of optimality, and because it has proved to be a challenging problem over the past 30 years, with unknown complexity status until the end of 2005.

Open Problems

All results mentioned leave open problems. For example, can one find a simpler proof of NP-completeness, which can be checked without running computer programs? It would be desirable to improve the approximation constant which can be achieved in polynomial time (to simplify the proof, the constant shown in [7] is not explicitly calculated and it would be relatively large, if the proof is not refined). The time bound for the approximation scheme could hopefully be improved. It could also be possible to refine the software which computes efficiently the exact MWT for large random point sets, so that it can handle efficiently a wider range of input, i.e., not only completely random point sets. This could perhaps be done by combining this software with implementations of fixed-parameter algorithms, as the ones reported in [2, 4], or with other approaches. It is also open whether or not the

subexponential exact method can be further improved.

Experimental Results

Please see the last paragraph under the section about key results.

URL to Code

Link to code used to compare some dynamic programming approaches in [2]: http://fuzzy.cs.unimagdeburg.de/~borgelt/pointgon.html

Cross-References

- Greedy Set-Cover Algorithms
- Minimum Geometric Spanning Trees
- Minimum k-Connected Geometric Networks

Recommended Reading

1. Beirouti R, Snoeyink J (1998) Implementations of the LMT heuristic for minimum weight triangulation. In: Symposium on computational geometry, Minneapolis, 7–10 June 1998, pp 96-105
2. Borgelt C, Grantson M, Levcopoulos C (2008) Fixed parameter algorithms for the minimum weight triangulation problem. Int J Comput Geom Appl 18(3):185–220
3. de Berg M, van Kreveld M, Overmars M, Schwarzkopf O (2000) Computational geometry – algorithms and applications, 2nd edn. Springer, Heidelberg
4. Grantson M, Borgelt C, Levcopoulos C (2005) Minimum weight triangulation by cutting out triangles. In: Proceedings of the 16th annual international symposium on algorithms and computation (ISAAC 2005), Sanya. Lecture notes in computer science, vol 3827. Springer, Heidelberg, pp 984–994
5. Gudmundsson J, Levcopoulos C (2000) A parallel approximation algorithm for minimum weight triangulation. Nord J Comput 7(1):32–57
6. Hjelle Ø, Dæhlen M (2006) Triangulations and applications. In: Mathematics and visualization, vol IX. Springer, Heidelberg. ISBN:978-3-540-33260-2
7. Levcopoulos C, Krznaric D (1998) Quasi-greedy triangulations approximating the minimum weight triangulation. J Algorithms 27(2):303–338
8. Levcopoulos C, Krznaric D (1999) The greedy triangulation can be computed from the Delaunay triangulation in linear time. Comput Geom 14(4):197–220
9. Levcopoulos C, Lingas A (1987) On approximation behavior of the greedy triangulation for convex polygons. Algorithmica 2:15–193
10. Levcopoulos C, Lingas A (2014) A note on a QPTAS for maximum weight triangulation of planar point sets. Inf Process Lett 114:414–416
11. Lingas A (1998) Subexponential-time algorithms for minimum weight triangulations and related problems. In: Proceedings 10th Canadian conference on computational geometry (CCCG), McGill University, Montreal, 10–12 Aug 1998
12. Mulzer W, Rote G (2006) Minimum-weight triangulation is NP-hard. In: Proceedings of the 22nd annual ACM symposium on computational geometry (SoCG'06), Sedona. ACM, New York. The journal version in J ACM 55(2):Article No. 11 (2008)
13. Remy J, Steger A (2006) A quasi-polynomial time approximation scheme for minimum weight triangulation. In: Proceedings of the 38th ACM symposium on theory of computing (STOC'06), Seattle. ACM, New York. The journal version in J ACM 56(3):Article No. 15 (2009)

Minimum Weighted Completion Time

V.S. Anil Kumar[1], Madha V. Marathe[2], Srinivasan Parthasarathy[2], and Aravind Srinivasan[3]

[1]Virginia Bioinformatics Institute, Virginia Tech, Blacksburg, VA, USA

[2]IBM T.J. Watson Research Center, Hawthorne, NY, USA

[3]Department of Computer Science, University of Maryland, College Park, MD, USA

Keywords

Average weighted completion time

Years and Authors of Summarized Original Work

1999; Afrati et al.

Problem Definition

The minimum weighted completion time problem involves (i) a set J of n jobs, a positive weight w_j for each job $j \in J$, and a release date r_j before which it cannot be scheduled; (ii) a set of m machines, each of which can process at most one job at any time; and (iii) an arbitrary set of positive values $\{p_{i,j}\}$, where $p_{i,j}$ denotes the time to process job j on machine i. A schedule involves assigning jobs to machines and choosing an order in which they are processed. Let C_j denote the completion time of job j for a given schedule. The *weighted completion time* of a schedule is defined as $\sum_{j \in J} w_j C_j$, and the goal is to compute a schedule that has the minimum weighted completion time.

In the scheduling notation introduced by Graham et al. [8], a scheduling problem is denoted by a 3-tuple $\alpha|\beta|\gamma$, where α denotes the machine environment, β denotes the additional constraints on jobs, and γ denotes the objective function. In this article, we will be concerned with the α-values 1, P, R, and Rm, which respectively denote one machine, identical parallel machines (i.e., for a fixed job j and for each machine i, $p_{i,j}$ equals a value p_j that is independent of i), unrelated machines (the $p_{i,j}$'s are dependent on both job i and machine j), and a fixed number m (not part of the input) of unrelated machines. The field β takes on the values r_j, which indicates that the jobs have release dates, and the value *pmtn*, which indicates that preemption of jobs is permitted. Further, the value *prec* in the field β indicates that the problem may involve precedence constraints between jobs, which poses further restrictions on the schedule. The field γ is either $\sum w_j C_j$ or $\sum C_j$, which denote total weighted and total (unweighted) completion times, respectively.

Some of the simpler classes of the weighted completion time scheduling problems admit optimal polynomial-time solutions. They include the problem $P||\sum C_j$, for which the *shortest-job-first* strategy is optimal, the problem $1\ ||\sum w_j C_j$, for which Smith's rule [14] (scheduling jobs in their nondecreasing order of p_j/w_j values) is optimal, and the problem $R||\sum C_j$, which can be solved via matching techniques [3, 10]. With the introduction of release dates, even the simplest classes of the weighted completion time minimization problem becomes strongly nondeterministic polynomial-time (NP)-hard. In this article, we focus on the work of Afrati et al. [1], whose main contribution is the design of polynomial-time approximation schemes (PTASs) for several classes of scheduling problems to minimize weighted completion time *with release dates*. Prior to this work, the best solutions for minimizing weighted completion time with release dates were all $O(1)$-approximation algorithms (e.g., [5, 6, 12]); the only known PTAS for a strongly NP-hard problem involving weighted completion time was due to Skutella and Woeginger [13], who developed a PTAS for the problem $P||\sum w_j C_j$. For an excellent survey on the minimum weighted completion time problem, we refer the reader to Chekuri and Khanna [4]. Another important objective is the flow time, which is a generalization of completion time; a recent breakthrough of Bansal and Kulkarni shows how to approximate the total flow time and maximum flow time to within polylgarithmic factors [2].

Key Results

Afrati et al. [1] were the first to develop PTASs for weighted completion time problems involving release dates. We summarize the running times of these PTASs in Table 1.

The results presented in Table 1 were obtained through a careful sequence of input transformations followed by dynamic programming. The input transformations ensure that the input becomes *well structured* at a slight loss in optimality, while dynamic programming allows efficient enumeration of all the near-optimal solutions to the well-structured instance.

The first step in the input transformation is *geometric rounding*, in which the processing times and release dates are converted to powers of $1+\epsilon$ with at most $1+\epsilon$ loss in the overall performance. More significantly, the step (i)

Minimum Weighted Completion Time, Table 1 Summary of results of Afrati et al. [1]

Problem	Running time of polynomial-time approximation schemes
$1 \mid r_j \mid \sum w_j C_j$	$O\left(2^{poly(\frac{1}{\epsilon})} n + n \log n\right)$
$P \mid r_j \mid \sum w_j C_j$	$O\left((m+1)^{poly(\frac{1}{\epsilon})} n + n \log n\right)$
$P \mid r_j, pmtn \mid \sum w_j C_j$	$O\left(2^{poly(\frac{1}{\epsilon})} n + n \log n\right)$
$Rm \mid r_j \mid \sum w_j C_j$	$O\left(f\left(m, \frac{1}{\epsilon}\right) poly\,(n)\right)$
$Rm \mid r_j, pmtn \mid \sum w_j C_j$	$O\left(f\left(m, \frac{1}{\epsilon}\right) n + n \log n\right)$
$Rm \mid\mid \sum w_j C_j$	$O\left(f\left(m, \frac{1}{\epsilon}\right) n + \log n\right)$

ensures that there are only a small number of distinct processing times and release dates to deal with, (ii) allows time to be broken into geometrically increasing intervals, and (iii) aligns release dates with start and end times of intervals. These are useful properties that can be exploited by dynamic programming.

The second step in the input transformation is *time stretching*, in which small amounts of idle time are added throughout the schedule. This step also changes completion times by a factor of at most $1 + O(\epsilon)$ but is useful for *cleaning up* the scheduling. Specifically, if a job is *large* (i.e., occupies a large portion of the interval where it executes), it can be pushed into the idle time of a later interval where it is small. This ensures that most jobs have small sizes compared with the length of the intervals where they execute, which greatly simplifies schedule computation. The next step is *job shifting*. Consider a partition of the time interval $[0,\infty)$ into intervals of the form $I_x = [(1+\epsilon)^x, (1+\epsilon)^{x+1})$ for integral values of x. The job-shifting step ensures that there is a slightly suboptimal schedule in which every job j gets completed within $O(\log_{1+\epsilon}(1+\frac{1}{\epsilon}))$ intervals after r_j. This has the following nice property: If we consider blocks of intervals $\mathcal{B}_0, \mathcal{B}_1, \ldots$, with each block $\mathcal{B}_i$ containing $O(\log_{1+\epsilon}(1+\frac{1}{\epsilon}))$ consecutive intervals, then a job j starting in block $\mathcal{B}_i$ completes within the next block. Further, the other steps in the job-shifting phase ensure that there are not too many *large* jobs which spill over to the next block; this allows the dynamic programming to be done efficiently.

The precise steps in the algorithms and their analysis are subtle, and the above description is clearly an oversimplification. We refer the reader to [1] or [4] for further details.

Applications

A number of optimization problems in parallel computing and operations research can be formulated as machine scheduling problems. When precedence constraints are introduced between jobs, the weighted completion time objective can generalize the more commonly studied makespan objective and hence is important.

Open Problems

Some of the major open problems in this area are to improve the approximation ratios for scheduling on unrelated or related machines for jobs with precedence constraints. The following problems in particular merit special mention. The best known solution for the $1 \mid prec \mid \sum w_j C_j$ problem is the 2-approximation algorithm due to Hall et al. [9]; improving upon this factor is a major open problem in scheduling theory. The problem $R \mid prec \mid \sum_j w_j C_j$ in which the precedence constraints form an arbitrary acyclic graph is especially open – the only known results in this direction are when the precedence constraints form chains [7] or trees [11].

The other open direction is inapproximability – there are significant gaps between the known approximation guarantees and hardness factors for various problem classes. For instance, the

$R||\sum w_j C_j$ and $R|r_j|\sum w_j C_j$ are both known to be approximable-hard, but the best known algorithms for these problems (due to Skutella [12]) have approximation ratios of 3/2 and 2, respectively. Closing these gaps remains a significant challenge.

Cross-References

- Approximation Schemes for Makespan Minimization
- Flow Time Minimization
- List Scheduling
- Minimum Flow Time

Acknowledgments The Research of V.S. Anil Kumar and M.V. Marathe was supported in part by NSF Award CNS-0626964. A. Srinivasan's research was supported in part by NSF Award CNS-0626636.

Recommended Reading

1. Afrati FN, Bampis E, Chekuri C, Karger DR, Kenyon C, Khanna S, Milis I, Queyranne M, Skutella M, Stein C, Sviridenko M (1999) Approximation schemes for minimizing average weighted completion time with release dates. In: Proceedings of the foundations of computer science, pp 32–44
2. Bansal N, Kulkarni J (2014) Minimizing flow-time on unrelated machines. arXiv:1401.7284
3. Bruno JL, Coffman EG, Sethi R (1974) Scheduling independent tasks to reduce mean finishing time. Commun ACM 17:382–387
4. Chekuri C, Khanna S (2004) Approximation algorithms for minimizing weighted completion time. In: Leung JY-T (eds) Handbook of scheduling: algorithms, models, and performance analysis. CRC, Boca Raton
5. Chekuri C, Motwani R, Natarajan B, Stein C (2001) Approximation techniques for average completion time scheduling. SIAM J Comput 31(1):146–166
6. Goemans M, Queyranne M, Schulz A, Skutella M, Wang Y (2002) Single machine scheduling with release dates. SIAM J Discret Math 15:165–192
7. Goldberg LA, Paterson M, Srinivasan A, Sweedyk E (2001) Better approximation guarantees for job-shop scheduling. SIAM J Discret Math 14:67–92
8. Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discret Math 5:287–326
9. Hall LA, Schulz AS, Shmoys DB, Wein J (1997) Scheduling to minimize average completion time: off-line and on-line approximation algorithms. Math Oper Res 22(3):513–544
10. Horn W (1973) Minimizing average flow time with parallel machines. Oper Res 21:846–847
11. Kumar VSA, Marathe MV, Parthasarathy S, Srinivasan A (2005) Scheduling on unrelated machines under tree-like precedence constraints. In: APPROX-RANDOM, pp 146–157
12. Skutella M (2001) Convex quadratic and semidefinite relaxations in scheduling. J ACM 46(2):206–242
13. Skutella M, Woeginger GJ (1999) A PTAS for minimizing the weighted sum of job completion times on parallel machines. In: Proceedings of the 31st annual ACM symposium on theory of computing (STOC'99), pp 400–407
14. Smith WE (1956) Various optimizers for single-stage production. Nav Res Log Q 3:59–66

Min-Sum Set Cover and Its Generalizations

Sungjin Im
Electrical Engineering and Computer Sciences (EECS), University of California, Merced, CA, USA

Keywords

Approximation; Covering problems; Greedy algorithm; Latency; Randomized rounding; Set cover; Submodular

Years and Authors of Summarized Original Work

2004; Feige, Lovász, Tetali
2010; Bansal, Gupta, Krishnaswamy
2011; Azar, Gamzu

Problem Definition

The min sum set cover (MSSC) problem is a latency version of the set cover problem. The input to MSSC consists of a collection of sets $\{S_i\}_{i\in[m]}$ over a universe of elements $[n] := \{1, 2, 3, \ldots, n\}$. The goal is to schedule elements,

one at a time, to hit all sets as early on average as possible. Formally, we would like to find a permutation $\pi : [n] \to [n]$ of the elements $[n]$ ($\pi(i)$ is the ith element in the ordering) such that the average (or equivalently total) cover time of the sets $\{S_i\}_{i\in[m]}$ is minimized. The cover time of a set S_i is defined as the earliest time t such that $\pi(t) \in S_i$. For convenience, we will say that we schedule/process element $\pi(i)$ at time i.

Since MSSC was introduced in [4], several generalizations have been studied. Here we discuss two of them. In the generalized min sum set cover (GMSSC) problem [2], each set S_i has a requirement κ_i. In this generalization, a set S_i is covered at the first time t when κ_i elements are scheduled from S_i, i.e., $|\{\pi(1), \pi(2), \ldots, \pi(t)\} \cap S_i| \geq \kappa_i$. Note that MSSC is a special case of GMSSC when $\kappa_i = 1$ for all $i \in [n]$.

Another interesting generalization is submodular ranking (SR) [1]. In SR, each set S_i is replaced with a nonnegative and monotone submodular function $f_i : 2^{[n]} \to [0, 1]$ with $f_i([n]) = 1$; function f is said to be submodular if $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$ for all $A, B \subseteq [n]$ and monotone if $f(A) \leq f(B)$ for all $A \subseteq B$. The cover time of each function f_i is now defined as the earliest time t such that $f_i(\{\pi(1), \pi(2), \ldots, \pi(t)\}) = 1$. Note that GMSSC is a special case of SR when $f_i(A) = \min\{|S_i \cap A|/\kappa_i, 1\}$. Also it is worth noting that SR generalizes set cover.

Key Results

We summarize main results known for MSSC, GMSSC, and SR.

Theorem 1 ([4]) *There is a 4-approximation for MSSC, and there is a matching lower bound* $4-\epsilon$ *unless* $P = NP$.

Interestingly, the tight 4-approximation was achieved by a very simple greedy algorithm that schedules an element at each time that covers the largest number of uncovered sets. The analysis in [4] introduced the notion of "histograms"; see below for more detail.

Theorem 2 ([3, 6, 7]) *There is an* $O(1)$*-approximation for GMSSC.*

Azar and Gamzu gave the first nontrivial approximation for GMSSC whose guarantee was $O(\log \max_i \kappa_i)$ [2]. The analysis was also based on histograms and was inspired by the work in [4]. Bansal et al. [3] showed that the analysis of the greedy algorithm in [2] is essentially tight and used a linear programming relaxation and randomized rounding to give the first $O(1)$-approximation for GMSSC; the precise approximation factor obtained was 485. The LP used in [3] was a time-indexed LP strengthened with knapsack covering inequalities. The rounding procedure combined threshold rounding and randomized "boosted-up" independent rounding. The approximation was later improved to 28 by [7], subsequently to 12.4 by [6]. The key idea for these improvements was to use α-point rounding to resolve conflicts between elements, which is popular in the scheduling literature.

Theorem 3 ([1, 5]) *There is an* $O(\log(1/\epsilon))$*-approximation for SR where* ϵ *is the minimum marginal positive increase of any function* f_i.

Note that this result immediately implies an $O(\log \max_i \kappa_i)$-approximation for GMSSC. The algorithm in [1] is an elegant greedy algorithm which schedules an element e at time t with the maximum $\sum_i (f_i(A \cup \{e\}) - f_i(A))/(1 - f_i(A))$ – here A denotes all elements scheduled by time $t-1$, and if $f_i(A) = 1$, then f_i is excluded from the summation. Note that this algorithm becomes the greedy algorithm in [4] for the special case of MSSC. The analysis was also based on histograms. Later, Im et al. [5] gave an alternative analysis of this greedy algorithm which was inspired by the analysis of other latency problems. We note that the algorithm that schedules element e that gives the maximum total marginal increase of $\{f_i\}$ has a very poor approximation guarantee, as observed in [1].

As we discussed above, there are largely three analysis techniques used in this line of work: histogram-based analysis, latency argument-based analysis, and LP rounding. We will sketch these techniques following [3–5] closely – we chose these papers since

they present the techniques in a relatively simpler way, though they do not necessarily give the best approximation guarantees or most general results. We begin with the analysis tools developed for greedy algorithms. To present key ideas more transparently, we will focus on MSSC.

Histogram-Based Analysis

We sketch the analysis of the 4-approximation in [4]. Let R_t denote the uncovered sets at time t and N_t the sets that are first covered at time t. Observe that $\sum_{t\in[n]} |R_t|$ is the algorithm's total cover time. In the analysis, we represent the optimal and the algorithm's solutions using histograms. First, in the optimal solution's histogram sets are ordered in increasing order of their cover times, and set S_i has width 1 and height equal to its cover time. In the algorithm's solution, as before, sets are ordered in increasing order of their cover times, but set S_i has height equal to $|R_t|/|N_t|$ where t is S_i's cover time. Here, the increase in the algorithm's objective at time t is uniformly distributed to sets N_t that are newly covered at time t. Note that the areas of both histograms are equal to the optimal cost and the algorithm's cost, respectively. Then we can show that after shrinking the algorithm's histogram by a factor of 2, both horizontally and vertically, one can place it completely inside the optimal solution's histogram. This analysis is very simple and is based on a clever observation on the greedy solution's structure. This type of analysis was also used in [1,2].

Latency Argument-Based Analysis

This analysis does not seem to yield tight approximation guarantees, but could be more flexible since it does not compare two histograms directly. The key idea is to show that if we can't charge the number of uncovered sets in our algorithm's schedule at time t to the analogous number in the optimal schedule, then our algorithm must have covered a lot of sets during the time interval $[t/2, t]$. In other words, if our algorithm didn't make enough progress recently, then our algorithm's current status can be shown to be comparable to the optimal solution's status. Intuitively, if our algorithm is not comparable to the optimal solution, then the algorithm can nearly catch up with the optimal solution by following the choices the optimal solution has made. For technical reasons, we may have to compare our algorithm's status to the optimal solution's earlier status. This analysis is easily generalized to GMSSC, SR, and more general metric settings [5].

We now discuss the linear programming-based approach. Bansal et al. discussed why greedy algorithms are unlikely to yield an $O(1)$-approximation for GMSSC [3].

LP and Randomized Rounding

Consider the following time-indexed integer program (IP) used in [3]: variable x_{et} is an indicator variable that is 1 if element e is scheduled at time t, otherwise 0. Variable y_{it} is 1 if S_i is covered by time t, otherwise 0. The IP is relaxed into an LP by allowing x, y to be fractional.

$$
\begin{aligned}
\min \quad & \sum_{t\in[n]} \sum_{i\in[m]} (1 - y_{it}) \\
\text{s.t.} \quad & \sum_{t\in[n]} x_{et} = 1 && \forall e \in [n] \\
& \sum_{e\in[n]} x_{et} = 1 && \forall t \in [n] \\
& \sum_{e\in S_i\setminus A} \sum_{1\le t'<t} x_{et'} \ge (\kappa_i - |A|) \cdot y_{it} && \forall i \in [m], A \subseteq S_i, t \in [n] \\
& 0 \le y \le 1 \\
& x \ge 0.
\end{aligned}
$$

Note that for *integral* solutions, the objective is exactly the total cover time since each set S_i uncovered at time t adds 1 to the objective. The first two constraints say that every element must be scheduled and exactly one element must be scheduled at a time. If we use the most natural constraint $\sum_{e \in S_i} \sum_{1 \le t' < t} x_{et'} \ge \kappa_i y_{it}$ (it says that if set S_i is covered by time t, then there must be at least κ_i elements scheduled from S_i by time t), the LP has a large integrality gap [3]. Hence, [3] strengthened the LP with the above knapsack covering inequalities. There is an easy separation oracle for the last constraint; hence we can solve the LP in polynomial time. The analysis is done by showing that the expected cover time of each set S_i is at most $O(1)$ factor larger than the earliest time τ when the set S_i is covered by the LP by at least a half, i.e., $y_{i\tau} \ge 1/2$. This is sufficient to give an $O(1)$-approximation since the LP pays at least $\tau/2$ for set S_i.

Applications

The MSSC problem and its closely related problems have various applications in adaptive query processing and distributed resource allocation problems. Also GMSSC has applications in Web page ranking and broadcast scheduling. For details, see [1, 4]. Perhaps it would be no stretch to say that min sum set cover problems are at least loosely connected to all problems whose goal is to satisfy multiple demands with the overall minimum latency.

Open Problems

An outstanding open problem is to settle the approximability of GMSSC. As mentioned before, GMSSC captures MSSC (all $\kappa_i = 1$), for which there is a tight 4-approximation known [4]. The other extreme case is when $\kappa_i = |S_i|$ for all i. This problem is essentially equivalent to a classic precedence constrained scheduling problem $1|\text{prec}|\sum_j w_j C_j$ for which there are several 2-approximations known; see [3] for pointers. However, the current best approximation factor known for GMSSC is 12.4. Im et al. conjectured that GMSSC admit a 4-approximation [6].

Recommended Reading

1. Azar Y, Gamzu I (2011) Ranking with submodular valuations. In: SODA, San Francisco, pp 1070–1079
2. Azar Y, Gamzu I, Yin X (2009) Multiple intents re-ranking. In: STOC, Bethesda, pp 669–678
3. Bansal N, Gupta A, Krishnaswamy R (2010) A constant factor approximation algorithm for generalized min-sum set cover. In: SODA, Austin, pp 1539–1545
4. Feige U, Lovász L, Tetali P (2004) Approximating min sum set cover. Algorithmica 40(4):219–234
5. Im S, Nagarajan V, van der Zwaan R (2012) Minimum latency submodular cover. In: ICALP (1), Warwick, pp 485–497
6. Im S, Sviridenko M, Zwaan R (2014) Preemptive and non-preemptive generalized min sum set cover. Math Program 145(1–2):377–401
7. Skutella M, Williamson DP (2011) A note on the generalized min-sum set cover problem. Oper Res Lett 39(6):433–436

Misra-Gries Summaries

Graham Cormode
Department of Computer Science, University of Warwick, Coventry, UK

Keywords

Approximate counting; Frequent items; Streaming algorithms

Years and Authors of Summarized Original Work

1982; Misra, Gries

Problem Definition

The frequent items problem is to process a stream of items and find all items occurring more than a given fraction of the time. It is one of the most heavily studied problems in data stream algorithms, dating back to the 1980s. Many applications rely directly or indirectly on finding the frequent items, and implementations are in use in large-scale industrial systems. Informally, given a sequence of items, the problem is simply to find those items which occur most frequently.

Typically, this is formalized as finding all items whose frequency exceeds a specified fraction of the total number of items. Variations arise when the items have weights and further when these weights can also be negative.

Definition 1 Given a stream $\mathcal{S}$ of n items $t_1 \ldots t_n$, the frequency of an item i is $f_i = |\{j \,|\, t_j = i\}|$. The exact ϕ-frequent items comprise the set $\{i \,|\, f_i > \phi n\}$.

Example The stream $\mathcal{S} = (a, b, a, c, c, a, b, d)$ has $f_a = 3$, $f_b = 2$, $f_c = 2$, $f_d = 1$. For $\phi = 0.2$, the frequent items are a, b, and c.

A streaming algorithm which solves this problem must use a linear amount of space, even for large values of ϕ: given an algorithm that claims to solve this problem, we could insert a set S of N items, where every item has frequency 1. Then, we could also insert N copies of item i. If i is then reported as a frequent item (occurring more than 50 % of the time), then $i \in S$, else $i \notin S$. Consequently, since set membership requires $\Omega(N)$ space, $\Omega(N)$ space is also required to solve the frequent items problem. Instead, an approximate version is defined based on a tolerance for error ϵ.

Definition 2 Given a stream $\mathcal{S}$ of n items, the ϵ-approximate frequent items problem is to return a set of items F so that for all items $i \in F$, $f_i > (\phi - \epsilon)n$, and there is no $i \notin F$ such that $f_i > \phi n$.

Since the exact ($\epsilon = 0$) frequent items problem is hard in general, we will use "frequent items" or "the frequent items problem" to refer to the ϵ-approximate frequent items problem. A related problem is to estimate the frequency of items on demand.

Definition 3 Given a stream $\mathcal{S}$ of n items defining frequencies f_i as above, the frequency estimation problem is to process a stream so that, given any i, an $\hat{f}_i$ is returned satisfying $\hat{f}_i \leq f_i \leq \hat{f}_i + \epsilon n$.

Key Results

The problem of frequent items dates back at least to a problem first studied by Moore in 1980 [5]. It was published as a "problem" in the Journal of Algorithms in the June 1981 issue [17], to determine if there was a majority choice in a list of n votes.

Preliminaries: The Majority Algorithm

In addition to posing the majority question as a problem, Moore also invented the MAJORITY algorithm along with Boyer in 1980, described in a technical report from early 1981 [4]. A similar solution with proof of the optimal number of comparisons was provided by Fischer and Salzburg [9]. MAJORITY can be stated as follows: store the first item and a counter, initialized to 1. For each subsequent item, if it is the same as the currently stored item, increment the counter. If it differs and the counter is zero, then store the new item and set the counter to 1; else, decrement the counter. After processing all items, the algorithm guarantees that if there is a majority vote, then it must be the item stored by the algorithm. The correctness of this algorithm is based on a pairing argument: if every non-majority item is paired with a majority item, then there should still remain an excess of majority items. Although not posed as a streaming problem, the algorithm has a streaming flavor: it takes only one pass through the input (which can be ordered arbitrarily) to find a majority item. To verify that the stored item really is a majority, a second pass is needed to simply count the true number of occurrences of the stored item.

Algorithm 1: MISRA-GRIES(k)

$n \leftarrow 0$; $T \leftarrow \emptyset$;
for each i :
do
- $n \leftarrow n + 1$;
- **if** $i \in T$
  - **then** $c_i \leftarrow c_i + 1$;
  - **else if** $|T| < k - 1$
    - **then** $T \leftarrow T \cup \{i\}$; $c_i \leftarrow 1$;
  - **else for all** $j \in T$
    - **do** $c_j \leftarrow c_j - 1$;
    - **if** $c_j = 0$
      - **then** $T \leftarrow T \backslash \{j\}$;

Misra-Gries Summary

The Misra-Gries summary is a simple algorithm that solves the frequent items problem. It can be viewed as a generalization of MAJORITY to track multiple frequent items.

Instead of keeping a single counter and item from the input, the MISRA-GRIES summary stores $k-1$ (item, counter) pairs. The natural generalization of MAJORITY is to compare each new item against the stored items T and increment the corresponding counter if it is among them. Else, if there is some counter with count zero, it is allocated to the new item and the counter set to 1. If all $k-1$ counters are allocated to distinct items, then all are decremented by 1. A grouping argument is used to argue that any item which occurs more than n/k times must be stored by the algorithm when it terminates. Example pseudocode to illustrate this algorithm is given in Algorithm 1, making use of set notation to represent the operations on the set of stored items T: items are added and removed from this set using set union and set subtraction, respectively, and we allow ranging over the members of this set (thus, implementations will have to choose appropriate data structures which allow the efficient realization of these operations). We also assume that each item j stored in T has an associated counter c_j. For items not stored in T, then c_j is defined as 0 and does not need to be explicitly stored.

This n/k generalization was first proposed by Misra and Gries [16]. The time cost of the algorithm is dominated by the $O(1)$ dictionary operations per update and the cost of decrementing counts. Misra and Gries use a balanced search tree and argue that the decrement cost is amortized $O(1)$; Karp et al. propose a hash table to implement the dictionary [11]; and Demaine et al. show how the cost of decrementing can be made worst case $O(1)$ by representing the counts using offsets and maintaining multiple linked lists [8].

Bose et al. [3] observed that executing this algorithm with $k = 1/\epsilon$ ensures that the count associated with each item on termination is at most ϵn below the true value. The bounds on the accuracy of the structure were tightened by Berinde et al. to show that the error depends only on the "tail": the total weight of items outside the top-k most frequent, rather than the total weight of all items [2]. This gives a stronger accuracy guarantee when the input distribution is skewed, for example, if the frequencies follow a Zipfian distribution. They also show that the algorithm can be altered to tolerate updates with weights, rather than assuming that each item has equal unit weight.

A similar data structure called SPACESAVING was introduced by Metwally et al. [15]. This structure also maintains a set of items and counters, but follows a different set of update rules. Recently, it was shown that the SPACESAVING structure is isomorphic to MISRA-GRIES: the state of both structures can be placed in correspondence as each update arrives [1]. The different representations reflect that SPACESAVING maintains an upper bound on the count of stored items, while MISRA-GRIES keeps a lower bound. In studies, the upper bound tends to be closer to the true count, but it is straightforward to switch between the two representations.

Moreover, Agarwal et al. [1] showed that the MISRA-GRIES summary is *mergeable*. That is, two summaries of different inputs of size k can be combined together to obtain a new summary of size k that summarizes the union of the two inputs. This merging can be done repeatedly, to summarize arbitrarily many inputs in arbitrary configurations. This allows the summary to be used in distributed and parallel environments.

Lastly, the concept behind the algorithm of tracking information on k representative elements has inspired work in other settings. Liberty [12] showed how this can be used to track an approximation to the best k-rank summary of a matrix, using k rows. This was extended by Ghashami and Phillips [10] to offer better accuracy by keeping more rows.

Applications

The question of tracking approximate counts for a large number of possible objects arises in

a number of settings. Many applications have arisen in the context of the Internet, such as tracking the most popular source, destinations, or source-destination pairs (those with the highest amount of traffic) or tracking the most popular objects, such as the most popular queries to a search engine, or the most popular pieces of content in a large content host. It forms the basis of other problems, such as finding the frequent itemsets within a stream of transactions: those subsets of items which occur as a subset of many transactions. Solutions to this problem have used ideas similar to the count and prune strategy of the Misra-Gries summary to find approximate frequent itemsets [14]. Finding approximate counts of items is also needed within other stream algorithms, such as approximating the entropy of a stream [6].

Experimental Results

There have been a number of experimental studies of Misra-Gries and related algorithms, for a variety of computing models. These have shown that the algorithm is accurate and fast to execute [7, 13].

URLs to Code and Data Sets

Code for this algorithm is widely available:

http://www.cs.rutgers.edu/~muthu/massdal-code-index.html
http://hadjieleftheriou.com/sketches/index.html
https://github.com/cpnielsen/twittertrends

Cross-References

▸ Count-Min Sketch

Recommended Reading

1. Agarwal P, Cormode G, Huang Z, Phillips J, Wei Z, Yi K (2012) Mergeable summaries. In: ACM principles of database systems, Scottsdale
2. Berinde R, Cormode G, Indyk P, Strauss M (2009) Space-optimal heavy hitters with strong error bounds. In: ACM principles of database systems, Providence
3. Bose P, Kranakis E, Morin P, Tang Y (2003) Bounds for frequency estimation of packet streams. In: SIROCCO, Umeå
4. Boyer B, Moore J (1981) A fast majority vote algorithm. Technical report ICSCA-CMP-32, Institute for Computer Science, University of Texas
5. Boyer RS, Moore JS (1991) MJRTY – a fast majority vote algorithm. In: Bledsoe WW, Boyer RS (eds) Automated reasoning: essays in honor of Woody Bledsoe. Automated reasoning series. Kluwer Academic, Dordrecht/Boston, pp 105–117
6. Chakrabarti A, Cormode G, McGregor A (2007) A near-optimal algorithm for computing the entropy of a stream. In: ACM-SIAM symposium on discrete algorithms, New Orleans
7. Cormode G, Hadjieleftheriou M (2009) Finding the frequent items in streams of data. Commun ACM 52(10):97–105
8. Demaine E, López-Ortiz A, Munro JI (2002) Frequency estimation of internet packet streams with limited space. In: European symposium on algorithms (ESA), Rome
9. Fischer M, Salzburg S (1982) Finding a majority among n votes: solution to problem 81-5. J Algorithms 3(4):376–379
10. Ghashami M, Phillips JM (2014) Relative errors for deterministic low-rank matrix approximations. In: ACM-SIAM symposium on discrete algorithms, Portland, pp 707–717
11. Karp R, Papadimitriou C, Shenker S (2003) A simple algorithm for finding frequent elements in sets and bags. ACM Trans Database Syst 28: 51–55
12. Liberty E (2013) Simple and deterministic matrix sketching. In: ACM SIGKDD, Chicago, pp 581–588
13. Manerikar N, Palpanas T (2009) Frequent items in streaming data: an experimental evaluation of the state-of-the-art. Data Knowl Eng 68(4): 415–430
14. Manku G, Motwani R (2002) Approximate frequency counts over data streams. In: International conference on very large data bases, Hong Kong, pp 346–357
15. Metwally A, Agrawal D, Abbadi AE (2005) Efficient computation of frequent and top-k elements in data streams. In: International conference on database theory, Edinburgh
16. Misra J, Gries D (1982) Finding repeated elements. Sci Comput Program 2:143–152
17. Moore J (1981) Problem 81-5. J Algorithms 2:208–209

Mobile Agents and Exploration

Evangelos Kranakis[1] and Danny Krizanc[2]
[1]Department of Computer Science, Carleton, Ottawa, ON, Canada
[2]Department of Computer Science, Wesleyan University, Middletown, CT, USA

Keywords

Distributed algorithms; Graph exploration; Mobile agent; Navigation; Rendezvous; Routing; Time/Memory tradeoffs

Years and Authors of Summarized Original Work

1952; Shannon

Problem Definition

How can a network be explored efficiently with the help of mobile agents? This is a very broad question and to answer it adequately it will be necessary to understand more precisely what mobile agents are, what kind of networked environment they need to probe, and what complexity measures are interesting to analyze.

Mobile Agents

Mobile agents are autonomous, intelligent computer software that can move within a network. They are modeled as automata with limited memory and computation capability and are usually employed by another entity (to which they must report their findings) for the purpose of collecting information. The actions executed by the mobile agents can be discrete or continuous and transitions from one state to the next can be either deterministic or non-deterministic, thus giving rise to various natural complexity measures depending on the assumptions being considered.

Network Model

The network model is inherited directly from the theory of distributed computing. It is a connected graph whose vertices comprise the computing nodes and edges correspond to communication links. It may be static or dynamic and its resources may have various levels of accessibility. Depending on the model being considered, nodes and links of the network may have distinct labels. A particularly useful abstraction is an anonymous network whereby the nodes have no identities, which means that an agent cannot distinguish two nodes except perhaps by their degree. The outgoing edges of a node are usually thought of as distinguishable but an important distinction can be made between a globally consistent edge-labeling versus a locally independent edge-labeling.

Efficiency Measures for Exploration

Efficiency measures being adopted involve the time required for completing the exploration task, usually measured either by the number of edge traversals or nodes visited by the mobile agent. The interplay between time required for exploration and memory used by the mobile agent (*time/memory tradeoffs*) are key parameters considered for evaluating algorithms. Several researchers impose no restrictions on the memory but rather seek algorithms minimizing exploration time. Others, investigate the minimum size of memory which allows for exploration of a given type of network (e.g., tree) of given (known or unknown) size, regardless of the exploration time. Finally, several researchers consider time/memory tradeoffs.

Main Problems

Given a model for both the agents and the network, the graph exploration problem is that of designing an algorithm for the agent that allows it to visit all of the nodes and/or edges of the network. A closely related problem is where the domain to be explored is presented as a region of the plane with obstacles and exploration becomes visiting all unobstructed portions of the region in the sense of visibility. Another related problem is that of rendezvous where two or more agents are required to gather at a single node of a network.

Key Results

Claude Shannon [17] is credited with the first finite automaton algorithm capable of exploring an arbitrary maze (which has a range of 5×5 squares) by trial and error means. Exploration problems for mobile agents have been extensively studied in the scientific literature and the reader will find a useful historical introduction in Fraigniaud et al. [11].

Exploration in General Graphs

The network is modeled as a graph and the agent can move from node to node only along the edges. The graph setting can be further specified in two different ways. In Deng and Papadimitriou [8] the agent explores strongly connected directed graphs and it can move only in the direction from head to tail of an edge, but not vice-versa. At each point, the agent has a map of all nodes and edges visited and can recognize if it sees them again. They minimize the ratio of the total number of edges traversed divided by the optimum number of traversals, had the agent known the graph. In Panaite and Pelc [15] the explored graph is undirected and the agent can traverse edges in both directions. In the graph setting it is often required that apart from completing exploration the agent has to draw a map of the graph, i.e., output an isomorphic copy of it. Exploration of directed graphs assuming the existence of labels is investigated in Albers and Henzinger [1] and Deng and Papadimitriou [8]. Also in Panaite and Pelc [15], an exploration algorithm is proposed working in time $e + O(n)$, where is n the number of nodes and e the number of links. Fraigniaud et al. [11] investigate memory requirements for exploring unknown graphs (of unknown size) with unlabeled nodes and locally labeled edges at each node. In order to explore all graphs of diameter D and max degree d a mobile agent needs $\Omega(D \log d)$ memory bits even when exploration is restricted to planar graphs. Several researchers also investigate exploration of anonymous graphs in which agents are allowed to drop and remove pebbles. For example in Bender et al. [4] it is shown that one pebble is enough for exploration, if the agent knows an upper bound on the size of the graph, and $\Theta(\log \log n)$ pebbles are necessary and sufficient otherwise.

Exploration in Trees

In this setting it is assumed the agent can distinguish ports at a node (locally), but there is no global orientation of the edges and no markers available. *Exploration with stop* is when the mobile agent has to traverse all edges and stop at some node. For *exploration with return* the mobile agent has to traverse all edges and stop at the starting node. In *perpetual exploration* the mobile agent has to traverse all edges of the tree but is not required to stop. The upper and lower bounds on memory for the exploration algorithms analyzed in Diks et al. [9] are summarized in the table, depending on the knowledge that the mobile agent has. Here, n is the number of nodes of the tree, $N \geq n$ is an upper bound known to the mobile agent, and d is the maximum degree of a node of the tree.

Exploration	Knowledge	Lower bounds	Upper bounds
Perpetual	$\emptyset$	None	$O(\log d)$
w/Stop	$n \leq \mathrm{N}$	$\Omega(\log \log \log n)$	$O(\log N)$
w/Return	$\emptyset$	$\Omega(\log n)$	$O(\log^2 n)$

Exploration in a Geometric Setting

Exploration in a geometric setting with unknown terrain and convex obstacles is considered by Blum et al. [5]. They compare the distance walked by the agent (or robot) to the length of the shortest (obstacle-free) path in the scene and describe and analyze robot strategies that minimize this ratio for different kinds of scenes. There is also related literature for exploration in more general settings with polygonal and rectangular obstacles by Deng et al. [7] and Bar-Eli et al. [3], respectively. A setting that is important in wireless networking is when nodes are aware of their location. In this case, Kranakis et al. [12] give efficient algorithms for navigation, namely compass routing and face routing that guarantee delivery in Delaunay and arbitrary planar geometric graphs, respectively, using only local information.

Rendezvous

The rendezvous search problem differs from the exploration problem in that it concerns two searchers placed at different nodes of a graph that want to minimize the time required to rendezvous (usually) at the same node. At any given time the mobile agents may occupy a vertex of the graph and can either stay still or move from vertex to vertex. It is of interest to minimize the time required to rendezvous. A natural extension of this problem is to study multi-agent mobile systems. More generally, given a particular agent model and network model, a set of agents distributed arbitrarily over the nodes of the network are said to rendezvous if executing their programs after some finite time they all occupy the same node of the network at the same time. Of special interest is the highly symmetric case of anonymous agents on an anonymous network and the simplest interesting case is that of two agents attempting to rendezvous on a ring network. In particular, in the model studied by Sawchuk [16] the agents cannot distinguish between the nodes, the computation proceeds in synchronous steps, and the edges of each node are oriented consistently. The table summarizes time/memory tradeoffs known for six algorithms (see Kranakis et al. [13] and Flocchini et al. [10]) when the k mobile agents use indistinguishable pebbles (one per mobile agent) to mark their position in an n node ring.

Memory	Time	Memory	Time
$O(k \log n)$	$O(n)$	$O(\log n)$	$O(n)$
$O(\log n)$	$O(kn)$	$O(\log k)$	$O(n)$
$O(k \log\log n)$	$O\left(\frac{n \log n}{\log\log n}\right)$	$O(\log k)$	$O(n \log k)$

Kranakis et al. [14] show a striking computational difference for rendezvous in an oriented, synchronous, $n \times n$ torus when the mobile agents may have more indistinguishable tokens. It is shown that two agents with a constant number of unmovable tokens, or with one movable token each cannot rendezvous if they have $o(\log n)$ memory, while they can perform rendezvous with detection as long as they have one unmovable token and $O(\log n)$ memory. In contrast, when two agents have two movable tokens each then rendezvous (respectively, rendezvous with detection) is possible with constant memory in a torus. Finally, two agents with three movable tokens each and constant memory can perform rendezvous with detection in a torus. If the condition on synchrony is dropped the rendezvous problem becomes very challenging. For a given initial location of agents in a graph, De Marco et al. [6] measure the performance of a rendezvous algorithm as the number of edge traversals of both agents until rendezvous is achieved. If the agents are initially situated at a distance D in an infinite line, they give a rendezvous algorithm with cost $O(D|L_{\min}|^2)$ when D is known and $O((D + |L_{\max}|)^3)$ if D is unknown, where $|L_{\min}|$ and $|L_{\max}|$ are the lengths of the shorter and longer label of the agents, respectively. These results still hold for the case of the ring of unknown size but then they also give an optimal algorithm of cost $O(n|L_{\min}|)$, if the size n of the ring is known, and of cost $O(n|L_{\max}|)$, if it is unknown. For arbitrary graphs, they show that rendezvous is feasible if an upper bound on the size of the graph is known and they give an optimal algorithm of cost $O(D|L_{\min}|)$ if the topology of the graph and the initial positions are known to the agents.

Applications

Interest in mobile agents has been fueled by two overriding concerns. First, to simplify the complexities of distributed computing, and second to overcome the limitations of user interface approaches. Today they find numerous applications in diverse fields such as distributed problem solving and planning (e.g., task sharing and coordination), network maintenance (e.g., daemons in networking systems for carrying out tasks like monitoring and surveillance), electronic commerce and intelligence search (e.g., data mining and surfing crawlers to find products and services from multiple sources), robotic exploration (e.g., rovers, and other mobile platforms that can explore potentially dangerous environments or even enhance planetary extravehicular activity), and distributed rational decision making (e.g., auction

protocols, bargaining, decision making). The interested reader can find useful information in several articles in the volume edited by Weiss [18].

Open Problems

Specific directions for further research would include the study of time/memory tradeoffs in search game models (see Alpern and Gal [2]). Multi-agent systems are particularly useful for content-based searches and exploration, and further investigations in this area would be fruitful. Memory restricted mobile agents provide a rich model with applications in sensor systems. In the geometric setting, navigation and routing in a three dimensional environment using only local information is an area with many open problems.

Cross-References

- ▸ Deterministic Searching on the Line
- ▸ Robotics
- ▸ Routing

Recommended Reading

1. Albers S, Henzinger MR (2000) Exploring unknown environments. SIAM J Comput 29:1164–1188
2. Alpern S, Gal S (2003) The theory of search games and rendezvous. Kluwer, Norwell
3. Bar-Eli E, Berman P, Fiat A, Yan R (1994) On-line navigation in a room. J Algorithms 17:319–341
4. Bender MA, Fernandez A, Ron D, Sahai A, Vadhan S (1998) The power of a pebble: exploring and mapping directed graphs. In: Proceedings of the 30th annual symposium on theory of computing, Dallas, 23–26 May 1998, pp 269–278
5. Blum A, Raghavan P, Schieber B (1997) Navigating in unfamiliar geometric terrain. SIAM J Comput 26:110–137
6. De Marco G, Gargano L, Kranakis E, Krizanc D, Pelc A, Vaccaro U (2006) Asynchronous deterministic rendezvous in graphs. Theor Comput Sci 355: 315–326
7. Deng X, Kameda T, Papadimitriou CH (1998) How to learn an unknown environment I: the rectilinear case. J ACM 45:215–245
8. Deng X, Papadimitriou CH (1999) Exploring an unknown graph. J Graph Theory 32:265–297
9. Diks K, Fraigniaud P, Kranakis E, Pelc A (2004) Tree exploration with little memory. J Algorithms 51: 38–63
10. Flocchini P, Kranakis E, Krizanc D, Santoro N, Sawchuk C (2004) Multiple mobile agent rendezvous in the ring. In: Proceedings of the LATIN 2004, Bueons Aires, 5–8 Apr 2004. LNCS, vol 2976, pp 599–608
11. Fraigniaud P, Ilcinkas D, Peer G, Pelc A, Peleg D (2005) Graph exploration by a finite automaton. Theor Comput Sci 345:331–344
12. Kranakis E, Singh H, Urrutia J (1999) Compass routing in geometric graphs. In: Proceedings of the 11th Canadian conference on computational geometry (CCCG-99), Vancouver, 15–18 Aug 1999, pp 51–54
13. Kranakis E, Krizanc D, Santoro N, Sawchuk C (2003) Mobile agent rendezvous search problem in the ring. In: Proceedings of the international conference on distributed computing systems (ICDCS), Providence, 19–22 May 2003, pp 592–599
14. Kranakis E, Krizanc D, Markou E (2006) Mobile agent rendezvous in a synchronous torus. In: Correa J, Hevia A, Kiwi M (eds) Proceedings of LATIN 2006, 7th Latin American symposium, Valdivia, 20–24 March 2006. SVLNCS, vol 3887, pp 653–664
15. Panaite P, Pelc A (1999) Exploring unknown undirected graphs. J Algorithms 33:281–295
16. Sawchuk C (2004) Mobile agent rendezvous in the ring. PhD thesis, Carleton University, Ottawa
17. Shannon C (1951) Presentation of a maze solving machine, in cybernetics, circular, causal and feedback machines in biological and social systems. In: von Feerster H, Mead M, Teuber HL (eds) Trans. 8th Conf, New York, March 15–16, 1951, pp 169–181. Josiah Mary Jr. Foundation, New York (1952)
18. Weiss G (ed) (1999) Multiagent systems: a modern approach to distributed artificial intelligence. MIT, Cambridge, MA

Model Checking with Fly-Automata

Bruno Courcelle and Iréne Durand
Laboratoire Bordelais de Recherche en Informatique (LaBRI), CNRS, Bordeaux University, Talence, France

Keywords

Automaton on terms; Clique-width; Fly-automaton; Fixed-parameter tractable algorithm; Model checking; Monadic second-order logic; Tree-width

Years and Authors of Summarized Original Work

2012, 2013; Courcelle, Durand

Problem Definition

The verification of monadic second-order (MSO) graph properties, equivalently, the model-checking problem for MSO logic over finite binary relational structures, is fixed-parameter tractable (FPT) where the parameter consists of the formula that expresses the property and the tree-width or the clique-width of the input graph or structure. How to build usable algorithms for this problem? The proof of the general theorem (an *algorithmic meta-theorem*, cf. [12]) is based on the description of the input by algebraic terms and the construction of finite automata that accept the terms describing the satisfying inputs. But these automata are in practice much too large to be constructed [11, 14]. A typical number of states is $2^{2^{10}}$, and lower bounds match this number. Can one use automata and overcome this difficulty?

Key Results

We propose to use *fly-automata* (*FA*) [3]. They are automata whose states are *described* and not *listed* and whose transitions are *computed on the fly* and not *tabulated*. When running on a term of size 1,000, a fly-automaton with $2^{2^{10}}$ states computes only 1,000 transitions if it is deterministic. FA can have infinitely many states. For example, a state can record, among other things, the (unbounded) number of occurrences of a particular symbol in the input term. FA can thus check certain graph properties that are *not monadic second-order expressible*. An example is *regularity*, the fact that all vertices have the same degree. Furthermore, an FA equipped with an *output function* that maps the set of accepting states to an effectively given domain $\mathcal{D}$ can compute a value, for example, the number of k-colorings of the given graph G or the minimum cardinality of one of the k color classes if G is k-colorable (this number measures how close this graph is to be $(k-1)$-colorable). We have implemented and tested an FA that computes the number of 3-colorings of a graph.

Tree-width and *clique-width* are graph complexity measures that serve as parameters in many FPT algorithms [7, 8, 10]. Both are based on hierarchical decompositions of graphs that can be expressed by terms written with the operation symbols of appropriate *graph algebras* [6]. The model-checking automata take such terms as inputs. We will present results concerning graphs of bounded clique-width. The similar results for graphs of bounded tree-width reduce to them as we will explain at the end of this section.

Graph Algebras and Monadic Second-Order Logic

Graphs are finite, undirected, and without loops and multiple edges. The extension to directed graphs, possibly with loops and/or labels, is straightforward. A graph G is identified with the relational structure $\langle V_G, edg_G \rangle$ where edg_G is a binary symmetric relation representing adjacency.

Rather than giving a formal definition of *monadic second-order* (MSO) logic, we present the closed formula expressing 3-colorability (an NP-complete property). It is $\exists X, Y.Col(X,Y)$ where $Col(X,Y)$ is the formula

$$X \cap Y = \emptyset \wedge \forall u, v.\{edg(u,v) \Longrightarrow [\neg(u \in X \wedge v \in X) \wedge \neg(u \in Y \wedge v \in Y) \wedge \neg(u \notin X \cup Y \wedge v \notin X \cup Y)]\}.$$

This formula expresses that X, Y and $V_G - (X \cup Y)$ are the three color classes of a 3-coloring. The corresponding colors are respectively 1, 2, and 3.

Definition 1 (The graph algebra $\mathcal{G}$)

(a) We will use $\mathbb{N}_+$ as a set of labels called *port labels*. A *p-graph* is a triple $G = \langle V_G, edg_G, \pi_G \rangle$ where π_G is a mapping: $V_G \to \mathbb{N}_+$. If $\pi_G(x) = a$, we say that x is an *a-port*. The set $\pi(G)$ of port labels of G is its *type*. By using a default label, say 1, we make every nonempty graph into a p-graph of type $\{1\}$.

(b) We let F_k be the following finite set of *operations* on p-graphs of type included in $C := \{1, \ldots, k\} \subseteq \mathbb{N}_+$:
- The binary symbol $\oplus$ denotes the union of two *disjoint* p-graphs,
- The unary symbol $redb_{a\to b}$ denotes the *relabelling* that changes every port label a into b (where $a, b \in C$),
- The unary symbol $add_{a,b}$, for $a < b$, $a, b \in C$, denotes the *edge addition* that adds an edge between every a-port x and every b-port y (unless there is already an edge between them, our graphs have no multiple edges),
- For each $a \in C$, the nullary symbol $\mathbf{a}$ denotes an isolated a-port.

(c) Every term t in $T(F_k)$ (the set of finite terms written with F_k) is called a *k-expression*. Its *value* is a p-graph, $val(t)$, that we now define. For each position u of t (equivalently, each node u of the syntax tree of t), we define a p-graph $val(t)/u$, whose vertex set is the set of leaves of t below u. The definition of $val(t)/u$ is, for fixed t, by bottom-up induction on u:
- If u is an occurrence of $\mathbf{a}$, then $val(t)/u$ has vertex u as an a-port and no edge,
- If u is an occurrence of $\oplus$ with sons u_1 and u_2, then $val(t)/u := val(t)/u_1 \oplus val(t)/u_2$ (note that $val(t)/u_1$ and $val(t)/u_2$ are disjoint),
- If u is an occurrence of $relab_{a\to b}$ with son u_1, then $val(t)/u := relab_{a\to b}(val(t)/u_1)$,
- If u is an occurrence of $add_{a,b}$ with son u_1, then $val(t)/u := add_{a,b}(val(t)/u_1)$.

Finally, $val(t) := val(t)/root_t$. Its vertex set is the set of all leaves (occurrences of nullary symbols). For an example, let

$$t := add^1_{b,c}(add^2_{a,b}(\mathbf{a}^3 \oplus^4 \mathbf{b}^5) \oplus^6 relab^7_{b\to c}(add^8_{a,b}(\mathbf{a}^9 \oplus^{10} \mathbf{b}^{11})))$$

where the superscripts 1–11 number the positions of t. The p-graph $val(t)$ is $3_a - 5_b - 11_c - 9_a$ where the subscripts a, b, c indicate the port labels. (For clarity, port labels are letters in examples.) If $u := 2$ and $w := 8$, then $t/u = t/w = add_{a,b}(\mathbf{a} \oplus \mathbf{b})$; however, $val(t)/u$ is the p-graph $3_a - 5_b$ and $val(t)/w$ is $9_a - 11_b$, isomorphic to $val(t)/u$.

(d) The *clique-width* of a graph G, denoted by $cwd(G)$, is the least integer k such that G is isomorphic to $val(t)$ for some t in $T(F_k)$. We denote by $\mathcal{G}_k$ the set $val(T(F_k))$ of p-graphs that are the value of a term over F_k. We let F be the union of the sets F_k and $\mathcal{G}$ be the union of the sets $\mathcal{G}_k$. Every p-graph is isomorphic to a graph in $\mathcal{G}$, hence, has a clique-width.

(e) An *F-congruence* is an equivalence relation $\approx$ on p-graphs such that:
- Two isomorphic p-graphs are equivalent, and
- If $G \approx G'$ and $H \approx H'$, then $\pi(G) = \pi(G')$, $G \oplus H \approx G' \oplus H'$, $add_{a,b}(G) \approx add_{a,b}(G')$ and $relab_{a\to b}(G) \approx relab_{a\to b}(G')$.

(f) A set of graphs L is *recognizable* if it is a union of classes of an F-congruence such that, for each finite type $C \subseteq \mathbb{N}_+$, the number of equivalence classes of p-graphs of type C is finite.

Definition 2 (Fly-automata)

(a) Let H be a finite or countable, effectively given, signature. A *fly-automaton over H* (in short, an *FA over H*) is a 4-tuple $\mathcal{A} = \langle H, Q_\mathcal{A}, \delta_\mathcal{A}, Acc_\mathcal{A} \rangle$ such that $Q_\mathcal{A}$ is the finite or countable, effectively given, set of *states*; $Acc_\mathcal{A}$ is the set of *accepting states,* a decidable subset of $Q_\mathcal{A}$; and $\delta_\mathcal{A}$ is a computable function that defines the *transition rules*: for each tuple $(f, q_1, \ldots, q_m)$ with $q_1, \ldots, q_m \in Q_\mathcal{A}$, $f \in H$, $\rho(f) = m \geq 0$, $\delta_\mathcal{A}(f, q_1, \ldots, q_m)$ is a finite set of states. We write $f[q_1, \ldots, q_m] \to q$ (and $f \to q$ if f is

nullary) to mean that $q \in \delta_{\mathcal{A}}(f, q_1, \ldots, q_m)$. We say that $\mathcal{A}$ is *finite* if H and $Q_{\mathcal{A}}$ are finite.

(b) *Runs* and *recognized languages* are defined as usual; see [1]. A *deterministic* FA $\mathcal{A}$ (by "deterministic" we mean "deterministic and complete") has a unique run on each term t, and $q_{\mathcal{A}}(t)$ is the state reached at the root of t. The mapping $q_{\mathcal{A}}$ is computable, and the membership in $L(\mathcal{A})$ of a term $t \in T(H)$ is decidable.

(c) Every FA $\mathcal{A}$ that is not deterministic can be *determinized* by an easy extension of the usual construction, see [3]; it is important that the sets $\delta_{\mathcal{A}}(f, q_1, \ldots, q_m)$ be finite.

(d) A deterministic FA over H *with output function* is a 4-tuple $\mathcal{A} = \langle H, Q_{\mathcal{A}}, \delta_{\mathcal{A}}, Out_{\mathcal{A}} \rangle$ that is a deterministic FA where $Acc_{\mathcal{A}}$ is replaced by a total and computable *output function* $Out_{\mathcal{A}}$: $Q_{\mathcal{A}} \to \mathcal{D}$ such that $\mathcal{D}$ is an effectively given domain. The *function computed by* $\mathcal{A}$ is $Comp(\mathcal{A}) : T(H) \to \mathcal{D}$ such that $Comp(\mathcal{A})(t) := Out_{\mathcal{A}}(q_{\mathcal{A}}(t))$.

Example 1 The number of accepting runs of an automaton.

Let $\mathcal{A} = \langle H, Q_{\mathcal{A}}, \delta_{\mathcal{A}}, Acc_{\mathcal{A}} \rangle$ be a nondeterministic FA. We construct a deterministic FA $\mathcal{B}$ that computes the number of accepting runs of $\mathcal{A}$ on any term in $T(H)$. As set of states $Q_{\mathcal{B}}$, we take the set of finite subsets of $Q_{\mathcal{A}} \times \mathbb{N}_+$. The transitions are defined so that $\mathcal{B}$ reaches state α at the root of $t \in T(H)$ if and only if α is the finite set of pairs $(q, n) \in Q_{\mathcal{A}} \times \mathbb{N}_+$ such that n is the number of runs of $\mathcal{A}$ that reach state q at its root. This number is finite and α can be seen as a partial function: $Q_{\mathcal{A}} \to \mathbb{N}_+$ having a finite domain. For a symbol f of arity 2, $\mathcal{B}$ has the transition: $f[\alpha, \beta] \to \gamma$ where γ is the set of pairs (q, n) such that n is the sum of the integers $n_p.n_r$ over all pairs $(p, r) \in Q_{\mathcal{A}} \times Q_{\mathcal{A}}$ such that $(p, n_p) \in \alpha$, $(r, n_r) \in \beta$ and $q \in \delta_{\mathcal{A}}(f, p, r)$. The transitions for other symbols are defined similarly. The function $Out_{\mathcal{A}}$ maps a state α to the sum of the integers n such that $(q, n) \in \alpha \cap (Acc_{\mathcal{A}} \times \mathbb{N}_+)$.□

Example 2 An FA for checking 3-colorability.

In order to construct an FA that accepts the terms $t \in T(F)$ such that $val(t)$ is 3-colorable, we first construct an FA $\mathcal{A}$ for the property $Col(X, Y)$. For this purpose, we transform F into $F^{(2)}$ by replacing each nullary symbol $\mathbf{a}$ by the four nullary symbols $(\mathbf{a}, ij)$, $i, j \in \{0, 1\}$. A term $t \in T(F^{(2)})$ defines, first, the graph $val(t')$ where t' is obtained from t by removing the Booleans i, j from the nullary symbols and, second, the pair (V_X, V_Y) such that V_X is the set of vertices u (leaves of t) that are occurrences of $(\mathbf{a}, 1j)$ for some $\mathbf{a}$ and j and V_Y is the set of those that are occurrences of $(\mathbf{a}, i1)$ for some $\mathbf{a}$ and i. The set of terms $t \in T(F^{(2)})$ such that $Col(V_X, V_Y)$ holds in $val(t')$ is defined by a deterministic FA $\mathcal{A}$ that we now specify. Its states are *Error* and the finite subsets of $\mathbb{N}_+ \times \{1, 2, 3\}$. Their meanings are as follows:

- At position u of t, the automaton reaches state *Error* if and only if $val(t')/u$ has a vertex in $V_X \cap V_Y$ or an edge between two vertices, either both in V_X or both in V_Y or both in $V_G - (V_X \cup V_Y)$, hence of the same color, respectively 1, 2, or 3;
- It reaches state $\alpha \subseteq \mathbb{N}_+ \times \{1, 2, 3\}$ if and only if these conditions do not hold and α is the set of pairs (a, i) such that $val(t')/u$ has an a-port of color i.

All states except *Error* are accepting. Here are the transitions of $\mathcal{A}$:

$$(\mathbf{a}, 00) \to \{(a, 3)\}, (\mathbf{a}, 10) \to \{(a, 1)\},$$
$$(\mathbf{a}, 01) \to \{(a, 2)\}, (\mathbf{a}, 11) \to \mathit{Error}.$$

For $\alpha, \beta \subseteq \mathbb{N}_+ \times \{1, 2, 3\}$, $\mathcal{A}$ has transitions:

$$\oplus\,[\alpha, \beta] \to \alpha \cup \beta,$$
$$add_{a,b}[\alpha] \to \mathit{Error}, \text{ if } (a, i) \text{ and } (b, i) \text{ belong to } \alpha \text{ for some } i = 1, 2, 3,$$
$$add_{a,b}[\alpha] \to \alpha, \text{ otherwise },$$
$$relab_{a \to b}[\alpha] \to \beta, \text{ obtained by replacing } a \text{ by } b \text{ in each pair of } \alpha.$$

Its other transitions are $\oplus[\alpha, \beta] \rightarrow Error$ if α or β is $Error$, $add_{a,b}[Error] \rightarrow Error$, and $relab_{a\rightarrow b}[Error] \rightarrow Error$.

This FA checks $Col(X, Y)$. To check, $\exists X, Y.Col(X, Y)$, we build a nondeterministic FA $\mathcal{B}$ by deleting the state $Error$, by replacing the first three rules of $\mathcal{A}$ by $\mathbf{a} \rightarrow \{(a, 3)\}, \mathbf{a} \rightarrow \{(a, 1)\}, \mathbf{a} \rightarrow \{(a, 2)\}$, and by deleting those that yield $Error$. All states are accepting, but on some terms, no run can reach the root, and these terms are rejected. Furthermore, the construction of Example 1 shows how to make $\mathcal{B}$ into a deterministic FA that computes the number of 3-colorings, because the 3-colorings of $val(t)$ are in bijection with the accepting runs of $\mathcal{B}$ on t. □

Recognizability Theorem: The set of graphs that satisfy a closed MSO formula φ is F-recognizable.

Weak Recognizability Theorem: For every closed MSO formula φ, for every k, the set of graphs in $\mathcal{G}_k$ that satisfy φ is F_k-recognizable.

About proofs: The Recognizability Theorem is Theorem 5.68 of [6]. Its proof shows that the equivalence defined by the fact that the two considered p-graphs have the same type and satisfy the same closed MSO formulas of quantifier height at most that of φ satisfies the conditions of Definition 1(f). (These formulas have unary predicates for expressing port labels.) The Weak Recognizability Theorem follows from the former one. It can be proved directly by constructing an FA over F [3]. (We construct a single FA, not a particular FA for each subsignature F_k as in Theorem 6.35 of [6].) This construction can be implemented, at least in a number of nontrivial cases. The proof of the strong theorem does not provide any usable automaton.

Counting and Optimizing Automata

Let $P(X_1, \ldots, X_s)$ be an MSO property of vertex sets $X_1, \ldots, X_s$. We denote $(X_1, \ldots, X_s)$ by $\overline{X}$ and $t \models P(\overline{X})$ means that $\overline{X}$ satisfies P in the graph $val(t)$ defined by a term t. We are interested not only to check the validity of $\exists \overline{X}.P(\overline{X})$ but also to compute from a term t the following values:

$\#\overline{X}.P(\overline{X})$, defined as the number of assignments $\overline{X}$ such that $t \models P(\overline{X})$,
$\mathrm{Sp}\overline{X}.P(\overline{X})$, the *spectrum* of $P(\overline{X})$, defined as the set of tuples of the form $(|X_1|, \ldots, |X_s|)$ such that $t \models P(\overline{X})$,
$\mathrm{MSp}\overline{X}.P(\overline{X})$, the *multispectrum* of $P(\overline{X})$, defined as the multiset of tuples $(|X_1|, \ldots, |X_s|)$ such that $t \models P(\overline{X})$.

These computations can be done by FA. The construction for $\#\overline{X}.P(\overline{X})$ is similar to that of Example 1. We obtain in this way FPT or XP algorithms [8, 10].

Edge Set Quantifications and Tree-Width

The two recognizability theorems and the corresponding constructions of FA yielding FPT and XP algorithms hold and can be done for graphs of bounded tree-width and MSO formulas with edge set quantifications: it suffices to replace a graph G by its incidence graph $Inc(G)$, a bipartite graph whose vertices are those of G and its edges, to observe that the clique-width of $Inc(G)$ is bounded in terms of the tree-width of G and that an MSO formula with edge set quantifications over G can be translated into an MSO formula over $Inc(G)$. Another approach is in [2].

Beyond MS Logic

The property that the considered graph is the union of two disjoint regular graphs with possibly some edges between these two subgraphs is not MSO expressible but can be checked by an FA. An FA can also compute the minimal number of edges between X and $V_G - X$ such that $G[X]$ and $G[V_G - X]$ are connected, when such a set X exists.

Open Problems

The parsing problem for graphs of clique-width at most k is NP-complete (with k in the input) [9]. Good heuristics remain to be developed.

Experimental Results

These constructions have been implemented and tested [3–5]. We have computed the number of optimal colorings of graphs of clique-width at most 8 for which the chromatic polynomial is known, which allows to verify the correctness of the automaton. We can verify in, respectively, 35 and 105 min that the 20×20 and the 6×60 grids are 3-colorable. In 29 min, we can verify that the McGee graph (24 vertices) given by a term over F_{10} is acyclically 3-colorable.

A different approach using games is presented in [13].

Recommended Reading

1. Comon H. et al (2007) Tree automata techniques and applications. http://tata.gforge.inria.fr/
2. Courcelle B. (2012) On the model-checking of monadic second-order formulas with edge set quantifications. Discrete Appl. Math. 160:866–887
3. Courcelle B., Durand I (2012) Automata for the verification of monadic second-order graph properties. J. Appl. Logic 10:368–409
4. Courcelle B., Durand I (submitted for publication, 2013) Computations by fly-automata beyond monadic second-order logic. http://arxiv.org/abs/1305.7120
5. Courcelle B., Durand I. (2013) Model-checking by infinite fly-automata. In: Proceedings of the 5th conference on algebraic informatics, Porquerolles, France. Lecture Notes in Computer Science, Vol. 8080, pp 211–222
6. Courcelle B., Engelfriet J. (2012) Graph structure and monadic second-order logic, a language theoretic approach. Vol. 138 of Encyclopedia of Mathematics and its Application. Cambridge University Press, Cambridge, UK
7. Courcelle B, Makowsky J, Rotics U Linear-time solvable optimization problems on graphs of bounded clique-width. Theory Comput. Syst. 33:125–150 (2000)
8. Downey R., Fellows M. (1999) Parameterized complexity. Springer, New York
9. Fellows M., Rosamond F., Rotics U., Szeider S. (2009) Clique-width is NP-complete. SIAM J. Discret. Math. 23:909–939
10. Flum J., Grohe M. (2006) Parametrized complexity theory. Springer, Berlin, Heidelberg
11. Frick M., Grohe M. (2004) The complexity of first-order and monadic second-order logic revisited. Ann Pure Appl. Logic 130:3–31
12. Grohe M., Kreutzer S. (2011) Model theoretic methods in finite combinatorics. In: Grohe M, Makowsky J (eds) Contemporary mathematics, Vol. 558. American Mathematical Society, Providence, Rhode Island, pp 181–206
13. Kneis J., Langer A., Rossmanith P. (2011) Courcelle's theorem – a game-theoretic approach. Discret Optim 8:568–594
14. Reinhardt K. (2002) The complexity of translating logic to finite automata. In: Graedel E. *et al*. (eds) Automata, logics, and infinite games: a guide to current research. Lecture Notes in Computer Science, Vol. 2500. Springer, Berlin, Heidelberg, pp 231–238

Modularity Maximization in Complex Networks

My T. Thai
Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, USA

Keywords

Approximation algorithms; Community structure; Complexity; Modularity clustering

Years and Authors of Summarized Original Work

2013; Dinh, Thai
2013; Dinh, Nguyen, Alim, Thai
2013; DasGupta, Desai
2014; Dinh, Thai

Problem Definition

Many complex networks of interests such as the Internet, social, and biological networks exhibit the community structure where nodes are naturally clustered into tightly connected communities, with only sparser connections between them. The modularity maximization is concerned with finding such community structures in a given complex network.

Consider a network represented as an undirected graph $G = (V, E)$ consisting of $n = |V|$ vertices and $m = |E|$ edges. The *adjacency*

matrix of G is denoted by $\boldsymbol{A} = (A_{ij})$, where A_{ij} is the weight of edge (i, j) and $A_{ij} = 0$ if $(i, j) \notin E$. We also denote the (weighted) degree of vertex i, the total weights of edges incident at i, by $\mathsf{deg}(i)$ or, in short, d_i.

Community structure (CS) is a division of the vertices in V into a collection of disjoint subsets of vertices $\mathcal{C} = \{C_1, C_2, \ldots, C_l\}$ (with **unspecified** l) where $\bigcup_{i=1}^{l} C_i = V$. Each subset $C_i \subseteq V$ is called a *community*, and we wish to have more edges connecting vertices in the same communities than edges that connect vertices in different communities. The *modularity* [7] of $\mathcal{C}$ is the fraction of the edges that fall within the given communities minus the expected number of such fraction if edges were distributed at random. The randomization of the edges is done so as to preserve the degree of each vertex. If vertices i and j have degrees d_i and d_j, then the expected number of edges between i and j is $\frac{d_i d_j}{2M}$. Thus, the modularity, denoted by Q, is then

$$Q(\mathcal{C}) = \frac{1}{2M} \sum_{ij} \left(A_{ij} - \frac{d_i d_j}{2M} \right) \delta_{ij} \qquad (1)$$

where M is the total edge weights and the element δ_{ij} of the membership matrix $\boldsymbol{\delta}$ is defined as

$$\delta_{ij} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ are in the same community} \\ 0, & \text{otherwise} \end{cases}$$

The modularity values can be either positive or negative, and the higher (positive) modularity values indicate stronger community structure. Therefore, the *maximizing modularity problem* asks us to find a division $\mathcal{C}$ which maximizes the modularity value $Q(\mathcal{C})$.

Key Results

Computational Complexity

This problem is different from the partition problem as we do not know the total number of partitions beforehand. That being said, l is unspecified. Somewhat surprisingly, modularity maximization is still NP-complete on trees, one of the simplest graph classes.

Theorem 1 *Modularity maximization on trees is NP-complete.*

The proof has been presented in [3], reducing from the subset-sum problem. Furthermore, for dense graphs, namely, for the complements of 3-regular graphs, DasGupta and Desai have provided a $(1+\epsilon)$-inapproximability of the modularity maximization problem [1], stated as follows.

Theorem 2 *It is NP-hard to approximate the modularity maximization problem on* $(n-4)$*–regular graphs within a factor of* $1+\epsilon$ *for some constant* $\epsilon > 0$.

The proof has been presented in [1], reducing from the maximum-cardinality-independent set problem for 3-regular graphs (3-MIS). The basic intuition behind this proof is that large-size cliques must be properly contained within the communities.

Exact Solutions

Although the problem is in NP class, efficient algorithms to obtain optimal solutions for small size networks are still of interest. Dinh and Thai have presented an exact algorithm with a running time of $O(n^5)$ to the problem on uniform-weighted trees [3]. The algorithm is based on the dynamic programming, which exploits the relationship between maximizing modularity and minimizing the sum of squares of component volumes, where volume of a component S is defined as $\mathsf{vol}(S) = \sum_{v \in S} d_v$.

When the input graph is not a tree, an exact solution based on integer linear programming (ILP) is provided by Dinh and Thai [3]. Note that in the ILP for modularity maximization, there is a triangle inequality $x_{ij} + xjk - x_{ik} \geq 0$ to guarantee the values of x_{ij} be consistent to each other. Here $x_{ij} = 0$ if i and j are in the same community; otherwise $x_{ij} = 0$. Therefore, the ILP has $3\binom{n}{3} = \theta(n^3)$ constraints, which is about half a million constraints for a network of 100 vertices. As a consequence, the sizes of solved instances were limited to few hundred nodes. Along this direction, Dinh and Thai have pre-

sented a sparse metric, which reduces the number of constraints to $O(n^2)$ in sparse networks where $m = O(n)$ [3].

Approximation Algorithms

When G is a tree, the problem can be solved by a polynomial time approximation scheme (PTAS) with a running time of $O(n^{\epsilon+1})$ for $\epsilon > 0$ [3]. The PTAS is solely based on the following observation. Removing $k-1$ edges in G will yield k connected communities and $Q_k \geq (1 - \frac{1}{k})Q_{\text{opt}}$ where Q_k is the maximum modularity of a community structure with k communities and Q_{opt} is the optimal solution.

When G having the degree distribution follows the power law, i.e., the fraction of nodes in the network having k degrees is proportional to $k^{-\gamma}$, where $1 < \gamma \leq 4$, the problem can be approximated to a constant factor for $\gamma > 2$ and up to an $O(1/\log n)$ when $1 < \gamma \leq 2$ [2]. The details of this algorithm, namely, low-degree following (LDF), is presented in Algorithm 1.

Algorithm 1 Low-degree following algorithm (parameter $d_0 \in \mathbb{N}^+$)

```
1.  L := ∅, M := ∅, O := ∅, p_i = 0 ∀i = 1..n
2.  for each vertex i ∈ V do
3.    if (k_i ≤ d_0)&(i ∉ L ∪ M) then
4.      if N(i) \ M ≠ ∅ then
5.        Select a vertex j ∈ N(i) \ M
6.        Let
      M = M ∪ {i}, L = L ∪ {j}, p_i = j
7.      else
8.        Select a vertex t ∈ N(i)
9.        O = O ∪ {i}, p_i = t
10. C = ∅
11. for each vertex i ∈ V \ (M ∪ O) do
12.   C_i = {i} ∪ {j ∈ M | p_j = i} ∪ {t ∈ O | p_{p_t} = i}
13.   C = C ∪ {C_i}
14. Return C
```

The selection of d_0 is important to derive the approximation factor as d_0 needs to be a sufficient large constant that is still relative small to n when n tends to infinity. In an actual implementation of the algorithm, Dinh and Thai have designed an automatic selection of d_0 to maximize Q. LDF can be extended to solve the problem in directed graphs [2].

Theorem 3 ([1]) *There exists an $O(\log d)$-approximation for d-regular graphs with $d < \frac{n}{2\ln n}$ and $O(\log d_{\max})$-approximation for weighted graphs $d_{\max} < \frac{\sqrt{n}}{16\ln n}$.*

Modularity in Dynamic Networks

Networks in real life are very dynamic, which requires us to design an adaptive approximation algorithm to solve the problem. In this approach, the community structure (CS) at time t is detected based on the community structure at time $t-1$ and the changes in the network, instead of recomputing it from scratch. Indeed, the above LDF algorithm can be enhanced to cope with this situation [4]. At first LDF is run to find the base CS at time 0. Then at each time step, we adaptively follow and unfollow the nodes that violate the condition 3 in Algorithm 1.

Applications

Finding community structures has applications in vast domains. For example, a community in biology networks often consists of proteins, genes, or subunits with functional similarity. Thus, finding communities can help to predict unknown proteins. Likewise, in online social networks, a community can be a group of users having common interests; therefore, obtaining CS can help to predict user interests. Furthermore, detecting CS finds itself extremely useful in deriving social-based solutions for many network problems, such as forwarding and routing strategies in communication networks, Sybil defense, worm containment on cellular networks, and sensor reprogramming. In the network visualization perspective, finding CS helps display core network components and their mutual interactions, hence presents a more compact and understandable description of the network as a whole.

Recommended Reading

1. DasGupta B Desai D (2013) On the complexity of Newman's community finding approach for biological and social networks. J Comput Syst Sci 79:50–67
2. Dinh TN, Thai MT (2013) Community detection in scale-free networks: approximation algorithms for

maximizing modularity. IEEE J Sel Areas Commun Spec Issue Netw Sci (JSAC) 31(6):997–1006
3. Dinh TN, Thai MT (2014) Towards optimal community detection: from trees to general weighted networks. Internet Math. doi:10.1080/15427951.2014.950875
4. Dinh TN, Nguyen NP, Alim MA, Thai MT (2013) A near-optimal adaptive algorithm for maximizing modularity in dynamic scale-free networks. J Comb Optim (JOCO). doi:10.1007/s10878-013-9665-1
5. Fortunato S (2010) Community detection in graphs. Phys Rep 486(3-5):75-174
6. Fortunato S Barthelemy M (2007) Resolution limit in community detection. Proc Natl Acad Sci 104(1):36-41
7. Newman MEJ (2006) Modularity and community structure in networks. Proc Natl Acad Sci 103:8577–8582
8. Nguyen NP, Dinh TN, Tokala S, Thai MT (2011) Overlapping communities in dynamic networks: their detection and mobile applications. In: Proceedings of ACM international conference on mobile computing and networking (MobiCom), Las Vegas

Monotone Minimal Perfect Hash Functions

Paolo Boldi and Sebastiano Vigna
Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy

Keywords

Minimal perfect hash functions; Succinct data structures

Years and Authors of Summarized Original Work

2009; Belazzougui, Boldi, Pagh, Vigna
2011; Belazzougui, Boldi, Pagh, Vigna

Problem Definition

A minimal perfect hash function is a (data structure providing a) bijective map from a set S of n keys to the set of the first n natural numbers. In the static case (i.e., when the set S is known in advance), there is a wide spectrum of solutions available, offering different trade-offs in terms of construction time, access time, and size of the data structure.

An important distinction is whether any bijection will be suitable or whether one wants it to respect some specific property. A *monotone minimal perfect hash function (MMPHF)* is one where the keys are bit vectors and the function is required to preserve their lexicographic order.

Sometimes in the literature, this situation is identified with the one in which the set S has some predefined linear order and the bijection is required to respect the order: in this case, one should more precisely speak of *order-preserving* minimal perfect hash functions; for this scenario, a ready-made $\Omega(n \log n)$ space lower bound is trivially available (since all the $n!$ possible key orderings must be representable). This is not true in the monotone case, so the distinction between monotone and order preserving is not moot.

Problem Formulation

Let $[x]$ denote the set of the first x natural numbers. Given a positive integer $u = 2^w$, and a set $S \subseteq [u]$ with $|S| = n$, a function $h : S \to [m]$ is *perfect* iff it is injective, *minimal* iff $m = n$, and *monotone* iff $x \leq y$ implies $h(x) \leq h(y)$. In the following, we assume to work in the RAM model with words of length w. For simplicity, we will describe only the case in which the keys have fixed length w; the results can be extended to the general case [2].

Key Results

One key building block that is needed to describe the possible approaches to the problem is that of storing an *arbitrary* r-bit function $h : S \to [2^r]$ in a succinct way (i.e., using $rn + o(n)$ bits and guaranteeing constant access time). A practical (although slightly less efficient) method for storing a succinct function $f : S \to [2^r]$ was presented in [8]. The construction draws three hash functions $h_0, h_1, h_2 : S \to [\gamma n]$ (with $\gamma \approx 1.23$) and builds a 3-hypergraph with one hyperedge $(h_0(x), h_1(x), h_2(x))$ for every $x \in$

B_0		B_1		B_2		B_3	
s_0	0001001000000	s_3	0010011000000	s_6	0010011010100	s_9	0010011110110
s_1	0010010101100	s_4	0010011001000	s_7	0010011010101	s_{10}	0100100010000
s_2	0010010101110	s_5	0010011010010	s_8	0010011010110		

Monotone Minimal Perfect Hash Functions, Fig. 1 A toy example: $S = \{s_0, \ldots, s_{10}\}$ is divided into three buckets of size three (except for the last one that contains just two elements), whose delimiters $D = \{s_2, s_5, s_8\}$ appear in boldface

d_0 :	
s_0	2
s_1	2
s_2	2
s_3	8
s_4	8
s_5	8
s_6	11
s_7	11
s_8	11
s_9	1
s_{10}	1

d_1 :	
00	0
00100110	1
001001101011	2
0	3

Monotone Minimal Perfect Hash Functions, Fig. 2 Bucketing with longest common prefix for the set S of Fig. 1: d_0 maps each element x of S to the length of the longest common prefix of the bucket to which x belongs; d_1 maps each longest common prefix to the bucket index

S. With positive probability, this hypergraph does not have a nonempty 2-core; or, equivalently, the set of equations (in the variables a_i)

$$f(x) = \left(a_{h_0(x)} + a_{h_1(x)} + a_{h_2(x)}\right) \bmod 2^r$$

has a solution that can be found by a hypergraph-peeling process in time $O(n)$. Storing the function amounts to storing γn integers of r bits each (the array a_i), so γrn bits are needed (excluding the bits required to store the hash functions), and it is possible to improve this amount to $\gamma n + rn$ using standard techniques [2]; function evaluation takes constant time (one essentially just needs to evaluate three hash functions). We shall refer to this data structure as an *MWHC function* (from the name of the authors). Alternative, asymptotically more efficient data structures for the same purpose are described in [3, 4].

Trivial Solution

MWHC functions can themselves be used to store a MMPHF (setting $r = \lceil \log n \rceil$ and using the appropriate function f). This idea (requiring $\gamma n \lceil \log n \rceil$ bits) is also implied in the original paper [8], where the authors actually present their construction as a solution to the order-preserving minimal perfect hash function problem.

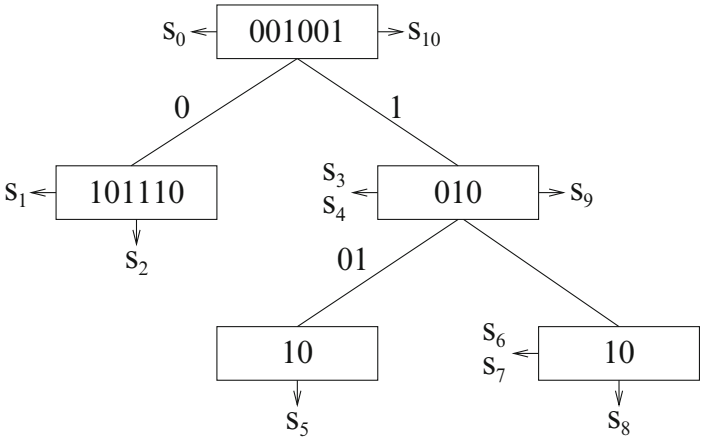
A Constant-Time, $O(n \log w)$-Space Solution

More sophisticated approaches are based on the general technique of *bucketing*: a *distributor* function $f : S \to [m]$ is stored, which divides the set of keys into m buckets respecting the lexicographic order; then, for every bucket $i \in [m]$, a MMPHF g_i on $f^{-1}(i)$ is stored as a succinct function. Different choices for the distributor and the bucket sizes provide different space/time trade-offs.

A constant-time solution can be obtained using buckets of equal size $b = O(\log n)$. Let $B_0, \ldots, B_{m-1}$ be the unique order-preserving partition of S into buckets of size b and p_i be the longest common prefix of the keys in B_i; Fig. 1 shows an example with $b = 3$. It is easy to see that the p_i's are all distinct, which allows us to build a succinct function d_1 mapping $p_i \mapsto i$. Moreover, we can store a succinct function $d_0 : S \to [w]$ mapping x to the length of the longest common prefix p_i of the bucket B_i containing x. The two functions together form the distributor (given x one applies d_1 to the prefix of x of length $d_0(x)$); see Fig. 2. The space required by d_0 and d_1 is $O(n \log w)$ and $O((n/b) \log(n/b)) = O(n)$, respectively. The functions g_i's require $\sum_i O(|B_i| \log b) = O(n \log \log n) = O(n \log w)$ bits. The overall space requirement is thus $O(n \log w)$; optimal values for b when using MWHC functions are computed in [2].

A $O(\log w)$-Time, $O(n \log \log w)$-Space Solution

In search for a better space bound, we note that an obvious alternative approach to the bucketing



Monotone Minimal Perfect Hash Functions, Fig. 3 The standard compacted trie built from the set D of Fig. 1. This data structure can be used to rank arbitrary elements of the universe with respect to D: when the trie is visited with an element not in D, the visit will terminate at an *exit node*, determining that the given element is to the left (i.e., smaller than) or to the right (i.e., larger than) all the leaves that descend from that node. The picture shows, for each element of S, the node where the visit would end

problem is by *ranking*. Given a set of strings X, a ranking data structure provides, for each string $s \in [u]$, the number of strings in X that are smaller than s, that is, $|\{x \in X \mid x < s\}|$. If you consider the set D of *delimiters* (i.e., the set containing the last string of each bucket), a distributor can be built using any data structure that provides the rank of an arbitrary string with respect to D.

For instance, a trivial way to obtain such rank information is to build a compacted trie [7] containing the strings in D (see Fig. 3). Much more sophisticated data structures are obviously available today (e.g., [6]), but they all fail to meet their purpose in our case. They occupy too much space, and we do not really need to rank *all* possible strings: we just need to rank strings in S. We call this problem the *relative ranking problem*: given sets $D \subseteq S$, we want to rank a string s w.r.t. D under the condition that s belongs to S.

The relative ranking problem can be approached in different ways: in particular, a static *probabilistic z-fast trie* [1] provides a $O(n)$-space, $O(\log w)$-time solution for buckets of size $O(\log w)$ (the trie can actually provide a wrong output for a small set of keys; their correct output needs to be stored separately – see [1] for details). The functions g_i's require $O(n \log \log w)$ bits, which dominates the space bound.

Different Approaches

Other theoretical and practical solutions, corroborated with experimental data, were described in [2].

Open Problems

Currently, the main open problem about monotone minimal perfect hashing is that all known lower bounds are trivial; in particular, the lower bound $n \log e + \log w - O(\log n)$ by Fredman and Komlós (provided that $u = 2^w \geq n^{2+\epsilon}$ for some fixed $\epsilon > 0$) for minimal perfect hash functions [5] is of little help – it is essentially independent from the size of the universe. We already know that the dependence on the size of the universe can be really small (as small as $O(n \log \log w)$ bits), but it is currently conjectured that there is no monotone minimal perfect hashing scheme whose number of bits per key is constant in the size of the universe.

URLs to Code and Datasets

The most comprehensive implementation of MMPHFs is contained in the Sux4J Java free library (http://sux4j.di.unimi.it/). A good collection of datasets is available by the LAW (http://law.di.unimi.it/) under the form of list of URLs from web crawls and list of ids from social networks (e.g., Wikipedia): these are typical use cases of a MMPHF.

Cross-References

▸ Minimal Perfect Hash Functions

Recommended Reading

1. Belazzougui D, Boldi P, Pagh R, Vigna S (2009) Monotone minimal perfect hashing: searching a sorted table with $O(1)$ accesses. In: Proceedings of the 20th annual ACM-SIAM symposium on discrete mathematics (SODA). ACM, New York, pp 785–794
2. Belazzougui D, Boldi P, Pagh R, Vigna S (2011) Theory and practice of monotone minimal perfect hashing. ACM J Exp Algorithmics 16(3):3.2:1–3.2:26
3. Charles DX, Chellapilla K (2008) Bloomier filters: a second look. In: Halperin D, Mehlhorn K (eds) Algorithms – ESA 2008, Proceedings of the 16th annual European symposium, Karlsruhe, 15–17 Sept 2008. Lecture notes in computer science, vol 5193. Springer, pp 259–270
4. Dietzfelbinger M, Pagh R (2008) Succinct data structures for retrieval and approximate membership (extended abstract). In: Aceto L, Damgård I, Goldberg LA, Halldórsson MM, Ingólfsdóttir A, Walukiewicz I (eds) Proceedings of the 35th international colloquium on automata, languages and programming, ICALP 2008, Part I: Track A: algorithms, automata, complexity, and games. Lecture notes in computer science, vol 5125. Springer, pp 385–396
5. Fredman ML, Komlós J (1984) On the size of separating systems and families of perfect hash functions. SIAM J Algebraic Discret Methods 5(1):61–68
6. Gupta A, Hon WK, Shah R, Vitter JS (2007) Compressed data structures: dictionaries and data-aware measures. Theor Comput Sci 387(3):313–331
7. Knuth DE (1997) The art of computer programming. Sorting and searching, vol 3, 2nd edn. Addison-Wesley, Reading
8. Majewski BS, Wormald NC, Havas G, Czech ZJ (1996) A family of perfect hashing methods. Comput J 39(6):547–554

Monotonicity Testing

Deeparnab Chakrabarty
Microsoft Research, Bangalore, Karnataka, India

Keywords

Boolean functions; Monotonicity; Property testing

Years and Authors of Summarized Original Work

1999; Dodis, Goldreich, Lehman, Raskhodnikova, Ron, Samorodnitsky
2000; Goldreich, Goldwasser, Lehman, Ron, Samorodnitsky
2013; Chakrabarty, Seshadhri

Problem Definition

A real-valued function $f : D \to \mathbb{R}$ defined over a partially ordered set (poset) D is *monotone* if $f(x) \leq f(y)$ for any two points $x \prec y$. In this article, we focus on the poset induced by the coordinates of a d-dimensional, n-hypergrid, $[n]^d$, where $x \prec y$ iff $x_i \leq y_i$ for all integers $1 \leq i \leq d$. Here, we have used $[n]$ as a shorthand for $\{1, \ldots, n\}$. The hypercube, $\{0, 1\}^d$, and the n-line, $[n]$, are two special cases of this.

Monotonicity testing is the algorithmic problem of deciding whether a given function is monotone. The algorithm has query access to the function, which means that it can query f at any domain point x and obtain the value of $f(x)$. The performance of the algorithm is measured by the number of queries it makes. Although the running time of the algorithm is also important, we ignore this parameter in this article because in most relevant cases it is of the same order of magnitude as the query complexity. We desire algorithms whose query complexity is bounded polynomially in d and $\log n$.

Monotonicity is a fundamental property. In one dimension, monotone functions correspond

to sorted arrays. In many applications it may be useful to know if an outcome of a procedure monotonically changes with some or all of its attributes. In learning theory, for instance, monotone concepts are known to require fewer samples to learn; it is useful to test beforehand if a concept is monotone or not.

Property Testing Framework

It is not too hard to see that, without any extra assumptions, monotonicity testing is infeasible unless almost the entire input is accessed. Therefore, the problem is studied under the property testing framework where the goal is to distinguish monotone functions from those which are "far" from monotone. A function f is said to be ϵ-far from being monotone if it needs to be modified on at least ϵ-fraction of the domain points to make it monotone.

Formally, the monotonicity testing problem is defined as follows. Given an input distance parameter ϵ, the goal is to design a $q(d, n, \epsilon)$-query tester which is a randomized algorithm that queries the function on at most $q(d, n, \epsilon)$ points and satisfies the following requirements:

(a) If the function is monotone, the algorithm **accepts**.
(b) If the function is ϵ-far from being monotone, the algorithm **rejects**.

The tester can err in each of the above cases but the error probability should be at most $1/3$. If a tester never rejects a monotone function, then it is called a *one-sided error* tester. If the queries made by the tester do not depend on the function values returned by the previous queries, then the tester is called a *nonadaptive* tester. The function q is called the *query complexity* of the monotonicity tester, and one is interested in (ideally matching) upper and lower bounds for it.

Key Results

Monotonicity testing was first studied by Ergun et al. [7] for functions defined on the n-line, that is, the case when $d = 1$. The authors designed an $O(\epsilon^{-1} \log n)$-query tester.

Goldreich et al. [9] studied monotonicity testing over the hypercube $\{0, 1\}^n$ and designed an $O(\epsilon^{-1} d)$-query tester for *Boolean* functions. These are functions whose range is $\{0, 1\}$. This tester repeats the following *edge test* $O(\epsilon^{-1} d)$ times: sample an edge of the hypercube uniformly at random, query the function values at its endpoints, and check if these two points violate monotonicity. That is, if $x \prec y$ and $f(x) > f(y)$, where x, y are the queried points. If at any time a violation is found, the tester rejects; otherwise it accepts. Clearly, this tester is nonadaptive and with one-sided error. Goldreich et al. [9] also demonstrated an $O(\epsilon^{-1} d \log n)$-query tester for Boolean functions defined over the d-dimensional n-hypergrid. This tester also defines a distribution over comparable pairs (not necessarily adjacent) of points, that is, points x, y with $x \prec y$. In each iteration, it samples a pair from the distribution, queries the function value on these points, and checks for a violation of monotonicity. Such testers are called *pair testers.*

Goldreich et al. [9] showed a *range reduction theorem* which states that if there exists a pair tester for Boolean functions over the hypergrid whose query complexity has linear dependence on ϵ, that is $q(d, n, \epsilon) = \epsilon^{-1} q(d, n)$, then such a tester could be extended to give an $O(\epsilon^{-1} q(d, n)|\mathbf{R}|)$-query tester for real-valued functions, where $\mathbf{R}$ is the range of the real-valued function. Note that $|\mathbf{R}|$ could be as large as 2^n. Dodis et al. [6] obtained a stronger range reduction theorem and showed an $O(\epsilon^{-1} q(d, n) \log |\mathbf{R}|)$-query tester under the same premise. This implied an $O(\epsilon^{-1} d \log n \log |\mathbf{R}|)$-query monotonicity tester for the hypergrid.

Recently, Chakrabarty and Seshadhri [3] removed the dependence on the range size exhibiting an $O(\epsilon^{-1} d \log n)$-query monotonicity tester for any real-valued function. In fact, the tester in their paper is the same as the tester defined in [9] alluded to above. In a separate paper, the same authors [4] complemented the above result by showing that any tester (adaptive and with

two-sided error) for monotonicity with distance parameter ϵ must make $\Omega(\epsilon^{-1} d \log n)$-queries.

Sketch of the Techniques

In this section, we sketch some techniques used in the results mentioned above. For simplicity, we restrict ourselves to functions defined on the hypercube.

To analyze their tester, Goldreich et al. [9] used the following *shifting* technique from combinatorics to convert a Boolean function f to a monotone function. Pick any dimension i and look at all the violated edges whose endpoints differ precisely in this dimension. If $(x \prec y)$ is a violation, then since f is Boolean, it must be that $f(x) = 1$ and $f(y) = 0$. For every such violation, redefine $f(x) = 0$ and $f(y) = 1$. Note that once dimension i is treated so, there is no violation across dimension i. The crux of the argument is that for any other dimension j, the number of violated edges across that dimension can only decrease. Therefore, once all dimensions are treated, the function becomes monotone, and the number of points at which it has been modified is at most twice the number of violated edges in the beginning. In other words, if f is ϵ-far, then the number of violated edges is at least $\epsilon 2^{d-1}$. Therefore, the edge test succeeds with probability at least ϵ/d since the total number of edges is $d 2^{d-1}$.

The fact that treating one dimension does not increase the number of violated edges across any other dimension crucially uses the fact that the function is Boolean and is, in general, not true for all real-valued functions. The range reduction techniques of [6,9] convert a real-valued function to a collection of Boolean ones with certain properties, and the size of the collection, which is $|\mathbf{R}|$ in [9] and $O(\log |\mathbf{R}|)$ in [6], appears as the extra multiplicative term in the query complexity.

The technique of Chakrabarty and Seshadhri [3] to handle general real-valued functions departs from the above in that it does not directly fix a function to make it monotone. Rather, they use the connection between the distance to monotonicity and matchings in the *violation graph* G_f which was defined by [6]. The vertices of G_f are the domain points, and (x, y) is an edge in G_f if and only if (x, y) is a violation to monotonicity of f. A folklore result, which appears in print in [8], is that the distance to monotonicity of f is precisely the cardinality of the *minimum vertex cover* of G_f divided by the domain size. This, in turn, implies that if f is ϵ-far, then *any* maximal matching in G_f must have cardinality at least $\epsilon 2^{d-1}$.

Chakrabarty and Seshadhri [3] introduce the notion of the *weighted* violation graph G_f where the weight of $(x \prec y)$ is defined as $(f(x) - f(y))$. They prove that the number of violated edges of the hypercube is at least the size of the *maximum weight* matching M^* in G_f. Since M^* is also maximal, this shows that the number of violated edges is at least $\epsilon 2^{d-1}$. The proof of [3] follows by charging each matched pair of points in M^* differing in the ith dimension to a *distinct* violated edge across the ith dimension.

Boolean Monotonicity Testing

Let us go back to Boolean functions defined over the hypercube. Recall that Goldreich et al. [9] designed an $O(\epsilon^{-1} d)$-query tester for such functions. Furthermore, their analysis is tight: there are functions for which the edge tester's success probability is $\Theta(\epsilon/d)$. The best known lower bound [8], however, is that any nonadaptive tester with one-sided error with respect to a specific constant distance parameter requires $\Omega(\sqrt{d})$ queries, and any adaptive, one-sided error tester required $\Omega(\log d)$-queries.

Chakrabarty and Seshadhri [2] obtained the first $o(d)$-query tester: they describe an $O(d^{7/8}\epsilon^{-3/2} \ln(1/\epsilon))$ query tester for Boolean functions defined over the hypercube. The tester is a combination of the edge test and what the authors call the *path test* – in each step one of the two is performed with probability $1/2$. The path test is the following. Orient all edges in the hypercube to go from the point with fewer 1s to the one with more 1s. Sample a random *directed* path from the all-0s point to the all-1s point. Sample two points x, y on this path which (a) have "close" to $d/2$ ones, and (b) are "sufficiently" far away from each other. Query $f(x), f(y)$ and check for a violation to monotonicity.

Chakrabarty and Seshadhri [2] obtain the following result for the path tester. If in the hypercube (and not the violation graph), there exists a matching of violated edges whose cardinality equals $\sigma 2^d$, then the path test catches a violation with probability roughly $\Omega(\sigma^3/\sqrt{d})$. Although this is good if σ is large, say a constant, the analysis above does not give better testers for functions with small σ. The authors circumvent this via the following *dichotomy* theorem. Given a function f with distance parameter ϵ, let the *number* of violated edges be $\delta 2^n$; from the result of Goldreich et al. [9], we know that $\delta \geq \epsilon$. Chakrabarty and Seshadhri [2] prove that if for any function f the quantity σ is small, then δ must be large. In particular, they prove that $\delta\sigma \geq \epsilon^2/32$. Therefore, for *any* function, either the edge test or the path test succeeds with probability $\omega(1/d)$.

Very recent work settles the question of nonadaptive, Boolean monotonicity testing. Chen, De, Servedio, and Tan [5] prove that any nonadaptive, monotonicity tester for Boolean functions needs to make $\Omega(d^{\frac{1}{2}-c})$-queries for any constant $c > 0$, even if it is allowed to have *two-sided error*. Khot, Minzer, and Safra [10] generalize the dichotomy theorem of Chakrabarty and Seshadhri [2] to obtain a $O(d^{1/2}\epsilon^{-2})$-query, nonadaptive monotonicity tester with one-sided error.

Open Problems

The query complexity of *adaptive* monotonicity testers for Boolean valued functions is not well understood. The best upper bounds are that of nonadaptive testers, while the best lower bound is only $\Omega(\log d)$. Understanding whether adaptivity helps in Boolean monotonicity testing or not is an interesting open problem. Recent work of Berman, Raskhodnikova, and Yaroslavtsev [1] sheds some light: they show that any nonadaptive, one-sided error monotonicity tester for functions $f : [n] \times [n] \to \{0, 1\}$, that is, functions defined over the two-dimensional grid, requires $\Omega(\frac{1}{\epsilon}\log(\frac{1}{\epsilon}))$ queries where ϵ is the distance parameter; on the other hand, they also demonstrate an *adaptive*, one-sided error monotonicity tester for such functions which makes only $O(\frac{1}{\epsilon})$ queries.

In this article, we only discussed the poset defined by the n-hypergrid. The best known tester for functions defined over a general N-element poset is an $O(\sqrt{N/\epsilon})$ tester, while the best lower bound is $\Omega(N^{\frac{1}{\log\log N}})$ for nonadaptive testers. Both results are due to Fischer et al. [8], and closing this gap is a challenging problem.

Cross-References

- Linearity Testing/Testing Hadamard Codes
- Testing if an Array Is Sorted
- Testing Juntas and Related Properties of Boolean Functions

Recommended Reading

1. Berman P, Raskhodnikova S, Yaroslavtsev G (2014) L_p Testing. In: Proceedings, ACM symposium on theory of computing (STOC), New York
2. Chakrabarty D, Seshadhri C (2013) A $o(n)$ monotonicity tester for Boolean functions over the hypercube. In: Proceedings, ACM symposium on theory of computing (STOC), Palo Alto
3. Chakrabarty D, Seshadhri C (2013) Optimal bounds for monotonicity and Lipschitz testing over hypercubes and hypergrids. In: Proceedings, ACM symposium on theory of computing (STOC), Palo Alto
4. Chakrabarty D, Seshadhri C (2013) An optimal lower bound for monotonicity testing over hypergrids. In: Proceedings, international workshop on randomization and computation (RANDOM), Berkeley
5. Chen X, De A, Servedio R, Tan LY (2015) Boolean function monotonicity testing requires (almost) $n^{1/2}$ non-adaptive queries. In: Proceedings, ACM symposium on theory of computing (STOC), Portland
6. Dodis Y, Goldreich O, Lehman E, Raskhodnikova S, Ron D, Samorodnitsky A (1999) Improved testing algorithms for monotonicity. In: Proceedings, international workshop on randomization and computation (RANDOM), Berkeley
7. Ergun F, Kannan S, Kumar R, Rubinfeld R, Viswanathan M (2000) Spot-checkers. J Comput Syst Sci 60(3):717–751
8. Fischer E, Lehman E, Newman I, Raskhodnikova S, Rubinfeld R, Samorodnitsky A (2002) Monotonicity testing over general poset domains. In: Proceedings, ACM symposium on theory of computing (STOC), Montreal

9. Goldreich O, Goldwasser S, Lehman E, Ron D, Samorodnitsky A (2000) Testing monotonicity. Combinatorica 20:301–337
10. Khot S, Minzer D, Safra S (2015) On monotonicity testing and boolean isoperimetric type theorems. In: Electornic colloquium on computational complexity (ECCC), TR15-011

Multi-armed Bandit Problem

Nicolò Cesa-Bianchi
Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy

Keywords

Adaptive allocation; Regret minimization; Repeated games; Sequential experiment design

Years and Authors of Summarized Original Work

2002; Auer, Cesa-Bianchi, Freund, Schapire
2002; Auer, Cesa-Bianchi, Fischer

Problem Definition

A multi-armed bandit is a sequential decision problem defined on a set of actions. At each time step, the decision maker selects an action from the set and obtains an observable payoff. The goal is to maximize the total payoff obtained in a sequence of decisions. The name *bandit* refers to the colloquial term for a slot machine ("one-armed bandit" in American slang) and to the decision problem, faced by a casino gambler, of choosing which slot machine to play next. Bandit problems naturally address the fundamental trade-off between exploration and exploitation in sequential experiments. Indeed, the decision maker must use a strategy (called allocation policy) able to balance the exploitation of actions that did well in the past with the exploration of actions that might give higher payoffs in the future. Although the original motivation came from clinical trials [14] (when different treatments are available for a certain disease and one must decide which treatment to use on the next patient), bandits have often been used in industrial applications, for example, to model the sequential allocation of a unit resource to a set of competing tasks.

Definitions and Notation

A bandit problem with $K \geq 2$ actions is specified by the processes $\langle X_{i,t} \rangle$ that, at each time step $t = 1, 2, \ldots$, assign a payoff $X_{i,t}$ to each action $i = 1, \ldots, K$. An allocation policy selects at time t an action $I_t \in \{1, \ldots, K\}$, possibly using randomization, and receives the associated payoff $X_{I_t,t}$. Note that the index I_t of the action selected by the allocation policy at time t can only depend on the set $X_{I_1,1}, \ldots, X_{I_{t-1},t-1}$ of previously observed payoffs (and on the policy's internal randomization). It is this information constraint that creates the exploration vs. exploitation dilemma at the core of bandit problems.

The performance of an allocation policy over a horizon of T steps is typically measured against that of the policy that consistently plays the optimal action for this horizon. This notion of performance, called regret, is formally defined by

$$R_T = \max_{i=1,\ldots,K} \mathbb{E}\left[\sum_{t=1}^{T} X_{i,t} - \sum_{t=1}^{T} X_{I_t,t}\right] . \quad (1)$$

The expectation in (1) is taken with respect to both the policy's internal randomization and the potentially stochastic nature of the payoff processes. Whereas we focus here on payoff processes that are either deterministic or stochastic i.i.d., other choices have been also considered. Notable examples are the Markovian payoff processes [8] or the more general Markov decision processes studied in reinforcement learning.

If the processes $\langle X_{i,t} \rangle$ are stochastic i.i.d. with unknown expectations $\mu_1, \ldots, \mu_K$, as in Robbins' original formalization of the bandit problem [13], then

$$\max_{i=1,\dots,K} \mathbb{E}\left[\sum_{t=1}^{T} X_{i,t}\right] = T\left(\max_{i=1,\dots,K} \mu_i\right) = T\mu^*$$

where $\mu^* = \max_i \mu_i$ is the highest expected payoff. In this case, the regret (1) becomes the stochastic regret

$$R_T^{\text{IID}} = T\mu^* - \sum_{t=1}^{T} \mathbb{E}\left[\mu_{I_t}\right]. \tag{2}$$

On the other hand, if the payoff processes $\langle X_{i,t}\rangle$ are fixed, deterministic sequences of unknown numbers $x_{i,t}$, then (1) becomes the nonstochastic regret

$$R_T^{\text{DET}} = \max_{i=1,\dots,K} \sum_{t=1}^{T} x_{i,t} - \sum_{t=1}^{T} \mathbb{E}\left[x_{I_t,t}\right] \tag{3}$$

where the expectation is only with respect to the internal randomization used by the allocation policy. This nonstochastic version of the bandit problem is directly inspired by the problem of playing repeatedly an unknown game – see the pioneering work of Hannan [9] and Blackwell [5] on repeated games and also the recent literature on online learning.

The analyses in [2, 3] assume bounded payoffs, $X_{i,t} \in [0,1]$ for all i and t. Under this assumption, $R_T = \mathcal{O}(T)$ irrespective of the allocation policy being used. The main problem is to determine the optimal allocation policies (the ones achieving the slowest regret growth) for the stochastic and the deterministic case. The parameters that are typically used to express the regret bounds are the horizon T and the number K of actions.

Key Results

Consider first the stochastic i.i.d. bandits with $K \geq 2$ actions and expected payoffs $\mathbb{E}[X_{i,t}] = \mu_i$ for $i = 1,\dots,K$ and $t \geq 1$. Also, let $\Delta_i = \mu^* - \mu_i$ (where, as before, $\mu^* = \max_{i=1,\dots,K} \mu_i$). A simple instance of stochastic bandits are the Bernoulli bandits, where payoffs $X_{i,1}, X_{i,2}, \dots$ for each action are i.i.d. Bernoulli random variables. Lai and Robbins [11] prove the following asymptotic lower bound on R_T for Bernoulli bandits. Let $N_{i,T} = \left|\{t = 1,\dots,T\}\, I_t = i\right|$ be the number of times the allocation policy selected action i within horizon T, and let $\text{KL}(\mu,\mu')$ be the Kullback-Leibler divergence between two Bernoulli random variables of parameter μ and μ'.

Theorem 1 ([11]) *Consider an allocation policy that, for any Bernoulli bandit with $K \geq 2$ actions, for any action i with $\mu_i < \mu^*$, and for any $a > 0$, satisfies $\mathbb{E}[N_{i,T}] = o\left(T^a\right)$. Then, for any choice of $\mu_1,\dots,\mu_K$,*

$$\liminf_{T\to\infty} \frac{R_T^{\text{IID}}}{\ln T} \geq \sum_{i\,:\,\Delta_i>0} \frac{\Delta_i}{\text{KL}(\mu_i,\mu^*)}.$$

This shows that when $\mu_1,\dots,\mu_K$ are fixed and $T \to \infty$, no policy can have a stochastic regret growing slower than $\Theta(K \ln T)$ in a Bernoulli bandit. For any fixed horizon T, the following stronger lower bound holds.

Theorem 2 ([3]) *For all $K \geq 2$ and any horizon T, there exist $\mu_1,\dots,\mu_K$ such that any allocation policy for the Bernoulli bandit with K actions suffers a stochastic regret of at least*

$$R_T^{\text{IID}} \geq \frac{1}{20}\sqrt{KT}. \tag{4}$$

Note that Yao's minimax principle immediately implies the lower bound $\Omega\left(\sqrt{KT}\right)$ on the nonstochatic regret R_T^{DET}.

Starting with [11], several allocation policies have been proposed that achieve a stochastic regret of optimal order $K \ln T$. The UCB algorithm of [2] is a simple policy achieving this goal nonasymptotically. At each time step t, UCB selects the action i, maximizing $\overline{X}_{i,t} + C_{i,t}$. Here $\overline{X}_{i,t}$ is the sample average of payoffs in previous selections of i, and $C_{i,t}$ is an upper bound on the length of the confidence interval for the estimate of μ_i at confidence level $1 - \frac{1}{t}$.

Theorem 3 ([2]) *There exists an allocation policy that for any Bernoulli bandit with $K \geq 2$ actions satisfies*

$$R_T^{\mathrm{IID}} \leq \sum_{i\,:\,\Delta_i > 0} \left(\frac{8}{\Delta_i} \ln T + 2 \right) \qquad \textit{for all } T.$$

The same result also applies to any stochastic i.i.d. bandit with payoffs bounded in $[0, 1]$. A comparison with Theorem 2 can be made by removing the dependence on Δ_i in the upper bound of Theorem 3. Once rewritten in this way, the bound becomes $R_T^{\mathrm{IID}} \leq 2\sqrt{K T \ln T}$, showing optimality (up to log factors) of UCB even when the values $\mu_1, \ldots, \mu_K$ can be chosen as a function of a target horizon T.

A bound on the nonstochastic regret matching the lower bound of Theorem 2 up to log factors is obtained via the randomized Exp3 policy introduced in [3]. At each time step t, Exp3 selects each action i with probability proportional to $\exp\bigl(\eta_t \widehat{X}_{i,t}\bigr)$, where $\widehat{X}_{i,t}$ is an importance sampling estimate of the cumulative payoff $x_{i,1} + \cdots + x_{i,t-1}$ (recall that $x_{i,t}$ is only observed if $I_t = i$) and the parameter η_t is set to $\sqrt{(\ln K)/(tK)}$.

Theorem 4 ([3]) *For any bandit problem with $K \geq 2$ actions and deterministic payoffs $\langle x_{i,t} \rangle$, the regret of the Exp3 algorithm satisfies $R_T^{\mathrm{DET}} \leq 2\sqrt{K T \ln K}$ for all T.*

Variants

The bandit problem has been extended in several directions. For example, in the pure exploration variant of stochastic bandits [6], a different notion of regret (called simple regret) is used. In this setting, at the end of each step t, the policy has to output a recommendation J_t for the index of the optimal action. The simple regret of the policy for horizon T is then defined by $r_T = \mu^* - \mathbb{E}\,\mu_{J_T}$.

The term adaptive adversary is used to denote a generalized nonstochastic bandit problem where the payoff processes of all actions are represented by a deterministic sequence $f_1, f_2, \ldots$ of functions. The payoff at time t of action i is then defined by $f_t(I_1, \ldots, I_{t-1}, i)$, where $I_1, \ldots, I_{t-1}$ is the sequence of actions selected by the policy up to time $t-1$. In the presence of an adaptive adversary, the appropriate performance measure is policy regret [1].

In the setting of contextual bandits [15], at each time step the allocation policy has access to side information (e.g., in the form of a feature vector). Here regret is not measured with respect to the best action, but rather with respect to the best mapping from side information to actions in a given class of such mappings.

If the set of action in a regular bandit is very large, possibly infinite, then the regret can be made small by imposing dependencies on the payoffs. For instance, the payoff at time t of each action a is defined by $f_t(a)$, where f_t is an unknown function. Control on the regret is then achieved by making specific assumptions on the space of actions a and on the payoff functions f_t (e.g., linear, Lipschitz, smooth, convex, etc.) – see, e.g., [4, 7] for early works in this direction.

Applications

Bandit problems have an increasing number of industrial applications particularly in the area of online services, where one can benefit from adapting the service to the individual sequence of requests. A prominent example of bandit application is online advertising. This is the problem of deciding which advertisement to display on the web page delivered to the next visitor of a website. A related problem is website optimization, which deals with the task of sequentially choosing design elements (font, images, layout) of the web page to be displayed to the next visitor. Another important application area is that of personalized recommendation systems, where the goal is to choose what to show from multiple content feeds in order to match the user's interest. In these applications, the payoff is associated with the users's actions, e.g., click-throughs or other desired behaviors – see, e.g., [12]. Bandits have been also applied to the problem of source routing, where a sequence of packets must be routed from a source host to a destination host in a given network, and the network protocol allows to choose a specific source-destination path for

each packet to be sent. The (negative) payoff is the time it takes to deliver a packet and depends additively on the congestion of the edges in the chosen path – see, e.g., [4]. A further application area is computer game playing, where each move is chosen by simulating and evaluating many possible game continuations after the move. Algorithms for bandits (more specifically, for a tree-based version of the bandit problem) can be used to explore more efficiently the huge tree of game continuations by focusing on the most promising subtrees. This idea has been successfully implemented in the MoGo player, which plays Go at the world-class level. MoGo is based on the UCT strategy for hierarchical bandits [10], which in turn builds on the UCB allocation policy.

Cross-References

- ▸ Online Learning and Optimization
- ▸ Reinforcement Learning

Recommended Reading

1. Arora R, Dekel O, Tewari A (2009) Online bandit learning against an adaptive adversary: from regret to policy regret. In: Proceedings of the 29th international conference on machine learning, Montreal
2. Auer P, Cesa-Bianchi N, Fischer P (2002) Finite-time analysis of the multiarmed bandit problem. Mach Learn J 47(2–3):235–256
3. Auer P, Cesa-Bianchi N, Freund Y, Schapire R (2002) The nonstochastic multiarmed bandit problem. SIAM J Comput 32(1):48–77
4. Awerbuch B, Kleinberg R (2004) Adaptive routing with end-to-end feedback: distributed learning and geometric approaches. In: Proceedings of the 36th annual ACM symposium on theory of computing, Chicago. ACM, pp 45–53
5. Blackwell D (1956) An analog of the minimax theorem for vector payoffs. Pac J Math 6:1–8
6. Bubeck S, Munos R, Stoltz G (2009) Pure exploration in multi-armed bandits problems. In: Proceedings of the 20th international conference on algorithmic learning theory, Porto
7. Flaxman AD, Kalai AT, McMahan HB (2005) Online convex optimization in the bandit setting: gradient descent without a gradient. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms, Philadelphia. Society for Industrial and Applied Mathematics, pp 385–394
8. Gittins J, Glazebrook K, Weber R (2011) Multi-armed bandit allocation indices, 2nd edn. Wiley, Hoboken
9. Hannan J (1957) Approximation to Bayes risk in repeated play. Contrib. Theory Games 3:97–139
10. Kocsis L, Szepesvari C (2006) Bandit based Monte-Carlo planning. In: Proceedings of the 15th European conference on machine learning, Vienna, pp 282–293
11. Lai TL, Robbins H (1985) Asymptotically efficient adaptive allocation rules. Adv Appl Math 6: 4–22
12. Li L, Chu W, Langford J, Schapire R (2010) A contextual-bandit approach to personalized news article recommendation. In: Proceedings of the 19th international conference on world wide web, Raleigh
13. Robbins H (1952) Some aspects of the sequential design of experiments. Bull Am Math Soc 58:527–535
14. Thompson W (1933) On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Bull Am Math Soc 25: 285–294
15. Wang CC, Kulkarni S, Poor H (2005) Bandit problems with side observations. IEEE Trans Autom Control 50(3):338–355

Multicommodity Flow, Well-linked Terminals and Routing Problems

Chandra Chekuri
Department of Computer Science, University of Illinois, Urbana-Champaign, Urbana, IL, USA
Department of Mathematics and Computer Science, The Open University of Israel, Raanana, Israel

Keywords

All-or-nothing multicommodity flow problem; Edge disjoint paths problem; Maximum edge disjoint paths problem; Node disjoint paths problem

Years and Authors of Summarized Original Work

2005; Chekuri, Khanna, Shepherd

Problem Definition

Three related optimization problems derived from the classical edge disjoint paths problem (EDP) are described. An instance of EDP consists of an undirected graph $G = (V, E)$ and a multiset $\mathcal{T} = \{s_1t_1, s_2t_2, \ldots, s_kt_k\}$ of k node pairs. EDP is a decision problem: can the pairs in $\mathcal{T}$ be connected (alternatively routed) via edge-disjoint paths? In other words, are there paths $P_1, P_2, \ldots, P_k$ such that for $1 \leq i \leq k$, P_i is path from s_i to t_i, and no edge $e \in E$ is in more than one of these paths? EDP is known to be NP-Complete. This article considers there maximization problems related to EDP.

- **Maximum Edge-Disjoint Paths Problem (MEDP).** Input to MEDP is the same as for EDP. The objective is to *maximize* the number of pairs in $\mathcal{T}$ that can be routed via edge-disjoint paths. The output consists of a subset $S \subseteq \{1, 2, \ldots, k\}$ and for each $i \in S$ a path P_i connecting s_i to t_i such that the paths are edge-disjoint. The goal is to maximize $|S|$.
- **Maximum Edge-Disjoint Paths Problem with Congestion (MEDPwC).** MEDPwC is a relaxation of MEDP. The input, in addition to G and the node pairs, contains an integer congestion parameter c. The output is the same for MEDP; a subset $S \subseteq \{1, 2, \ldots, k\}$ and for each $i \in S$ a path P_i connecting s_i to t_i. However, the paths $P_i, 1 \leq i \leq k$ are not required to be edge-disjoint. The relaxed requirement is that for each edge $e \in E$, the number of paths for the routed pairs that contain e is at most c. Note that MEDPwC with $c = 1$ is the same as MEDP.
- **All-or-Nothing Multicommodity Flow Problem (ANF).** ANF is a different relaxation of MEDP obtained by relaxing the notion of routing. A pair s_it_i is now said to be routed if a unit flow is sent from s_i to t_i (potentially on multiple paths). The input is the same as for MEDP. The output consists of a subset $S \subseteq \{1, 2, \ldots, k\}$ such that there is a feasible multicommodity flow in G that routes one unit of flow for each pair in S. The goal is to maximize $|S|$.

In the rest of the article, graphs are assumed to be undirected multigraphs. Given a graph $G = (V, E)$ and $S \subset V$, let $\delta_G(S)$ denote the set of edges with exactly one end point in S. Let n denote the number of vertices in the input graph.

Key Results

A few results in the broader literature are reviewed in addition to the results from [5]. EDP is NP-Complete when k is part of the input. A highly non-trivial result of Robertson and Seymour yields a polynomial time algorithm when k is a fixed constant.

Theorem 1 ([16]) *There is a polynomial time algorithm for EDP when k is a fixed constant independent of the input size.*

Using Theorem 1 it is easy to see that MEDP and MEDPwC have polynomial time algorithms for fixed k. The same holds for ANF by simple enumeration since the decision version is polynomial-time solvable via linear programming.

The focus of this article is on the case when k is part of the input, and in this setting, all three problems considered are NP-hard. The starting point for most approximation algorithms is the natural multicommodity flow relaxation given below. This relaxation is valid for both MEDP and ANF. The end points of the input pairs are referred to as *terminals* and let X denote the set of terminals. To describe the relaxation as well as simplify further discussion, the following simple assumption is made without loss of generality; each node in the graph participates in at most one of the input pairs. This assumption implies that the input pairs induce a matching M on the terminal set X. Thus the input for the problem can alternatively given as a triple (G, X, M).

For the given instance (G, X, M), let $\mathcal{P}_i$ denote the set of paths joining s_i and t_i in G and let $\mathcal{P} = \cup_i \mathcal{P}_i$. The LP relaxation has the following variables. For each path $P \in \mathcal{P}$ there is a variable $f(P)$ which is the amount of flow sent on P. For

each pair $s_i t_i$ there is a variable x_i to indicate the total flow that is routed for the pair.

$$\text{(MCF} - \text{LP)} \max \sum_{i=1}^{k} x_i \quad \text{s.t}$$

$$x_i - \sum_{P \in \mathcal{P}_i} f(P) = 0 \quad 1 \leq i \leq k$$

$$\sum_{P: e \in P} f(P) \leq 1 \quad \forall e \in E$$

$$x_i, f(P) \in [0,1] 1 \leq i \leq k, P \in \mathcal{P}$$

The above path formulation has an exponential (in n) number of variables, however it can still be solved in polynomial time. There is also an equivalent compact formulation with a polynomial number of variables and constraints. Let OPT denote the value of an optimum solution to a given instance. Similarly, let OPT-LP denote the value of an optimum solution the LP relaxation for the given instance. It can be seen that OPT-LP $\geq$ OPT. It is known that the integrality gap of (MCF-LP) is $\Omega(\sqrt{n})$ [10]; that is, there is an infinite family of instances such that $OPT - LP/OPT = \Omega(\sqrt{n}$. The current best approximation algorithm for MEDP is given by the following theorem.

Theorem 2 ([7]) *The integrality gap of (MCF-LP) for MEDP is $\Theta(\sqrt{n})$ and there is an $O(\sqrt{n})$ approximation for MEDP.*

For MEDPwC the approximation ratio improves with the congestion parameter c.

Theorem 3 ([18]) *There is an $O(n^{1/c})$ approximation for MEDPwC with congestion parameter c. In particular there is a polynomial time algorithm that routes $\Omega(\text{OPT} - \text{LP}/n^{1/c})$ pairs with congestion at most c.*

The above theorem is established via randomized rounding of a solution to (MCF-LP). Similar results, but via simpler combinatorial algorithms, are obtained in [2, 15].

In [5] a new framework was introduced to obtain approximation algorithm for routing problems in undirected graphs via (MCF-LP). A key part of the framework is the so-called well-linked decomposition that allows a reduction of an arbitrary instance to an instance in which the terminals satisfy a strong property.

Definition 1 Let $G = (V, E)$ be a graph. A subset $X \subseteq V$ is *cut-well-linked* in G if for every $S \subset V$, $|\delta_G(S)| \geq \min\{|S \cap X|, |(V \setminus S) \cap X|\}$. X is *flow-well-linked* if there exists a feasible fractional multicommodity flow in G for the instance in which there is a demand of $1/|X|$ for each unordered pair $uv, u, v \in X$.

The main result in [5] is the following.

Theorem 4 ([5]) *Let (G, X, M) be an instance of MEDP or ANF and let* OPT-LP *be the value of an optimum solution to (MCF-LP) on (G, X, M). There there is a polynomial time algorithm that obtains a collection of instances $(G_1, X_1, M_1), (G_2, X_2, M_2), \ldots, (G_h, X_h, M_h)$ with the following properties:*

- *The graphs $G_1, G_2, \ldots, G_h$ are node-disjoint induced subgraphs of G. For $1 \leq i \leq h$, $X_i \subseteq X$ and $M_i \subseteq M$.*
- *For $1 \leq i \leq h$, X_i is flow-well-linked in G_i.*
- *$\sum_{i=1}^{h} |X_i| = \Omega(\text{OPT} - \text{LP}/\log^2 n)$.*

For planar graphs and graphs that exclude a fixed minor, the above theorem gives a stronger guarantee: $\sum_{i=1}^{h} |X_i| = \Omega(\text{OPT} - \text{LP}/\log n)$. A well-linked instance satisfies a strong symmetry property based on the following observation. If A is flow-well-linked in G then for *any* matching J on X, OPT-LP on the instance (G, A, J) is $\Omega(|A|)$. Thus the particular matching M of a given well-linked instance (G, X, M) is essentially irrelevant. The second part of the framework in [5] consists of exploiting the well-linked property of the instances produced by the decomposition procedure. At a high level this is done by showing that if G has a well-linked set X, then it contains a "crossbar" (a routing structure) of size $\Omega(|X|/\text{poly}(\log n))$. See [5] for more precise definitions. Techniques for the second part vary based on the problem as well as

the family of graphs in question. The following results are obtained using Theorem 4 and other non-trivial ideas for the second part [3–5, 8].

Theorem 5 ([5]) *There is an $O(\log^2 n)$ approximation for ANF. This improves to an $O(\log n)$ approximation in planar graphs.*

Theorem 6 ([5]) *There is an $O(\log n)$ approximation for MEDPwC in planar graphs for $c \geq 2$. There is an $O(\log n)$ approximation for ANF in planar graphs.*

Theorem 7 ([8]) *There is an $O(r \log n \log r)$ approximation for MEDP in graphs of treewidth at most r.*

Generalizations and Variants

Some natural variants and generalizations of the problems mentioned in this article are obtained by considering three orthogonal aspects: (i) node disjointness instead of edge-disjointness, (ii) capacities on the edges and/or nodes, and (iii) demand values on the pairs (each pair $s_i t_i$ has an integer demand d_i and the objective is to route d_i units of flow between s_i and t_i). Results similar to those mentioned in the article are shown to hold for these generalizations and variants [5]. Capacities and demand values on pairs are somewhat easier to handle while node-disjoint problems often require additional non-trivial ideas. The reader is referred to [5] for more details.

For some special classes of graphs (trees, expanders and grids to name a few), constant factor or poly-logarithmic approximation ratios are known for MEDP.

Applications

Flow problems are at the core of combinatorial optimization and have numerous applications in optimization, computer science and operations research. Very special cases of EDP and MEDP include classical problems such as single-commodity flows, and matchings in general graphs, both of which have many applications. EDP and variants arise most directly in telecommunication networks and VLSI design. Since EDP captures difficult problems as special cases, there are only a few algorithmic tools that can address the numerous applications in a unified fashion. Consequently, empirical research tends to focus on application specific approaches to obtain satisfactory solutions. The flip side of the difficulty of EDP is that it offers a rich source of problems, the study of which has led to important algorithmic advances of broad applicability, as well as fundamental insights in graph theory, combinatorial optimization, and related fields.

Open Problems

A number of very interesting open problems remain regarding the approximability of the problems discussed in this article. Table 1 gives the best known upper and lower bounds on the approximation ratio as well as integrality gap of (MCF-LP). All the inapproximability results in Table 1, and the integrality gap lower bounds for MEDPwC and ANF, are from [1]. The inapproximability results are based on the assumption that NP $\not\subseteq$ ZTIME($n^{\text{poly}(\log n)}$). Closing the gaps between the lower and upper bounds are the major open problems.

Multicommodity Flow, Well-linked Terminals and Routing Problems, Table 1 Known bounds for MEDP, ANF and MEDPwC in general undirected graphs. The best upper bound on the approximation ratio is the same as the upper bound on the integrality gap of (MCF-LP)

	Integrality gap of (MCF-LP)		Approximation ratio
	Upper bound	Lower bound	Lower bound
MEDP	$O(\sqrt{n})$	$\Omega(\sqrt{n})$	$\Omega(\log^{1/2-\epsilon} n)$
MEDPwC	$O(n^{1/c})$	$\Omega(\log^{(1-\epsilon)/(c+1)} n)$	$\Omega(\log^{(1-\epsilon)/(c+1)} n)$
ANF	$O(\log^2 n)$	$\Omega(\log^{1/2-\epsilon} n)$	$\Omega(\log^{1/2-\epsilon} n)$

Cross-References

- ▸ Randomized Rounding
- ▸ Separators in Graphs
- ▸ Treewidth of Graphs

Recommended Reading

The limited scope of this article does not do justice to the large literature on EDP and related problems. In addition to the articles cited in the main body of the article, the reader is referred to [6, 9, 11–14, 17] for further reading and pointers to existing literature.

1. Andrews M, Chuzhoy J, Khanna S, Zhang L (2005) Hardness of the undirected edge-disjoint paths problem with congestion. In: Proceedings of the IEEE FOCS, pp 226–244
2. Azar Y, Regev O (2006) Combinatorial algorithms for the unsplittablae flow problem. Algorithmica 44(1):49–66, Preliminary version in Proceedings of the IPCO 2001
3. Chekuri C, Khanna S, Shepherd FB (2004) Edge-disjoint paths in Planar graphs. In: Proceedings of the IEEE FOCS, pp 71–80
4. Chekuri C, Khanna S, Shepherd FB (2004) The all-or-nothing multicommodity flow problem. In: Proceedings of the ACM STOC, pp 156–165
5. Chekuri C, Khanna S, Shepherd FB (2005) Multicommodity flow, well-linked terminals, and routing problems. In: Proceedings of the ACM STOC, pp 183–192
6. Chekuri C, Khanna S, Shepherd FB (2006) Edge-disjoint paths in planar graphs with constant congestion. In: Proceedings of the ACM STOC, pp 757–766
7. Chekuri C, Khanna S, Shepherd FB (2006) An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and UFP. Theory Comput 2:137–146
8. Chekuri C, Khanna S, Shepherd FB (2007) A note on multiflows and treewidth. Algorithmica. Published online
9. Frank A (1990) Packing paths, cuts, and circuits – a survey. In: Korte B, Lovász L, Prömel HJ, Schrijver A (eds) Paths, flows and VLSI-layout. Springer, Berlin, pp 49–100
10. Garg N, Vazirani V, Yannakakis M (1997) Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica 18(1):3–20, Preliminary version appeared in Proceedings of the ICALP 1993
11. Guruswami V, Khanna S, Rajaraman R, Shepherd FB, Yannakakis M (2003) Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. J Comput Syst Sci 67:473–496, Preliminary version in Proceedings of the ACM STOC 1999
12. Kleinberg JM (1996) Approximation algorithms for disjoint paths problems. PhD thesis, MIT, Cambridge, MA
13. Kleinberg JM (2005) An approximation algorithm for the disjoint paths problem in even-degree planar graphs. In: Proceedings of the IEEE FOCS, pp 627–636
14. Kolliopoulos SG (2007) Edge disjoint paths and unsplittable flow. In: Handbook on approximation algorithms and metaheuristics. Chapman & Hall/CRC Press computer & science series, vol 13. Chapman Hall/CRC Press
15. Kolliopoulos SG, Stein C (2004) Approximating disjoint-path problems using greedy algorithms and packing integer programs. Math Program A 99:63–87, Preliminary version in Proceedings of the IPCO 1998
16. Robertson N, Seymour PD (1995) Graph minors XIII. The disjoint paths problem. J Comb Theory B 63(1):65–110
17. Schrijver A (2003) Combinatorial optimization: polyhedra and efficiency. Springer, Berlin
18. Srinivasan A (1997) Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. In: Proceedings of the IEEE FOCS, pp 416–425

M

Multicut

Shuchi Chawla
Department of Computer Science, University of Wisconsin–Madison, Madison, WI, USA

Years and Authors of Summarized Original Work

1993; Garg, Vazirani, Yannakakis
1996; Garg, Vazirani, Yannakakis

Problem Definition

The Multicut problem is a natural generalization of the *s-t* mincut problem – given an undirected capacitated graph $G = (V, E)$ with k pairs of vertices $\{s_i, t_i\}$; the goal is to find a subset of edges of the smallest total capacity whose removal from G disconnects s_i from t_i for every

$i \in \{1, \cdots, k\}$. However, unlike the Mincut problem which is polynomial-time solvable, the Multicut problem is known to be NP-hard and APX-hard for $k \geq 3$ [6].

This problem is closely related to the Multi-Commodity Flow problem. The input to the latter is a capacitated network with k commodities (source-sink pairs); the goal is to route as much total flow between these source-sink pairs as possible while satisfying capacity constraints. The maximum multi-commodity flow in a graph can be found in polynomial time via linear programming, and there are also several combinatorial FPTASes known for this problem [7, 9, 11].

It is immediate from the definition of Multicut that the multicommodity flow in a graph is bounded above by the capacity of a minimum multicut in the graph. When there is a single commodity to be routed, the **max-flow min-cut** theorem of Ford and Fulkerson [8] states that the converse also holds: the maximum s-t flow in a graph is exactly equal to the minimum s-t cut in the graph. This duality between flows and cuts in a graph has many applications and, in particular, leads to a simple algorithm for finding the minimum cut in a graph.

Given its simplicity and elegance, several attempts have been made to extend this duality to other classes of flow and partitioning problems. Hu showed, for example, that the min-multicut equals the maximum multi-commodity flow when there are only two commodities in the graph [12]. Unfortunately, this property does not extend to graphs with more than two commodities. The focus has therefore been on obtaining approximate max-multicommodity flow min-multicut theorems. Such theorems would also imply a polynomial-time algorithm for approximately computing the capacity of the minimum multicut in a graph.

Key Results

Garg, Vazirani and Yannakakis [10] were the first to obtain an approximate max-multicommodity flow min-multicut theorem. They showed that the maximum multicommodity flow in a graph is always at least an $O(\log k)$ fraction of the minimum multicut in the graph. Moreover, their proof of this result is constructive. That is, they also provide an algorithm for computing a multicut for a given graph with capacity at most $O(\log k)$ times the maximum multicommodity flow in the graph. This is the best approximation algorithm known to date for the Multicut problem.

Theorem 1 *Let M denote the minimum multicut in a graph with k commodities and f denote the maximum multicommodity flow in the graph. Then*

$$\frac{M}{O(\log k)} \leq f \leq M .$$

Moreover, there is a polynomial time algorithm for finding an $O(\log k)$-approximate multicut in a graph.

Furthermore, they show that this theorem is tight to within constant factors. That is, there are families of graphs in which the gap between the maximum multicommodity flow and minimum multicut is $\Theta(\log k)$.

Theorem 2 *There exists a infinite family of multicut instances $\{(G_k, P_k)\}$ such that for all k, the graph $G_k = (V_k, E_k)$ contains k vertices and $P_k \subseteq V_k \times V_k$ is a set of $\Omega(k^2)$ source-sink pairs. Furthermore, the maximum multicommodity flow in the instance (G_k, P_k) is $O(k/\log k)$ and the minimum multicut is $\Omega(k)$.*

Garg et al. also consider the Sparsest Cut problem which is another partitioning problem closely related to Multicut, and provided an approximation algorithm for this problem. Their results for Sparsest Cut have subsequently been improved upon [3, 15]. The reader is referred to the entry on ▸ Sparsest Cut for more details.

Applications

A key application of the Multicut problem is to the 2CNF $\equiv$ Deletion problem. The latter is

a constraint satisfaction problem in which given a weighted set of clauses of the form $P \equiv Q$, where P and Q are literals, the goal is to delete a minimum weight set of clauses so that the remaining set is satisfiable. The 2CNF $\equiv$ Deletion problem models a number of partitioning problems, for example the Minimum Edge-Deletion Graph Bipartization problem – finding the minimum weight set of edges whose deletion makes a graph bipartite. Klein et al. [14] showed that the 2CNF $\equiv$ Deletion problem reduces in an approximation preserving way to Multicut. Therefore, a ρ-approximation to Multicut implies a ρ-approximation to 2CNF $\equiv$ Deletion. (See the survey by Shmoys [16] for more applications.)

Open Problems

There is a big gap between the best-known algorithm for Multicut and the best hardness result (APX-hardness) known for the problem. Improvements in either direction may be possible, although there are indications that the $O(\log k)$ approximation is the best possible. In particular, Theorem 2 implies that the integrality gap of the natural linear programming relaxation for Multicut is $\Theta(\log k)$. Although improved approximations have been obtained for other partitioning problems using semi-definite programming instead of linear programming, Agarwal et al. [1] showed that similar improvements cannot be achieved for Multicut – the integrality gap of the natural SDP-relaxation for Multicut is also $\Theta(\log k)$. On the other hand, there are indications that the APX-hardness is not tight. In particular, assuming the so-called Unique Games conjecture, it has been shown that Multicut cannot be approximated to within any constant factor [4, 13]. In light of these negative results, the main open problem related to this work is to obtain a super-constant hardness for the Multicut problem under a standard assumption such as $P \neq NP$.

The Multicut problem has also been studied in directed graphs. The best known approximation algorithm for this problem is an $O(n^{11/23} \log^{O(1)} n)$-approximation due to Aggarwal, Alon and Charikar [2], while on the hardness side, Chuzhoy and Khanna [5] show that there is no $2^{\Omega(\log^{1-\epsilon} n)}$ approximation, for any $\epsilon > 0$, unless NP$\subseteq$ZPP. Chuzhoy and Khanna also exhibit a family of instances for which the integrality gap of the natural LP relaxation of this problem (which is also the gap between the maximum directed multicommodity flow and the minimum directed multicut) is $\Omega(n^{1/7})$.

Cross-References

▸ Sparsest Cut

Recommended Reading

1. Agarwal A, Charikar M, Makarychev K, Makarychev Y (2005) O($\sqrt{\log n}$) approximation algorithms for Min UnCut, Min 2CNF deletion, and directed cut problems. In: Proceedings of the 37th ACM symposium on theory of computing (STOC), Baltimore, pp 573–581
2. Aggarwal A, Alon N, Charikar M (2007) Improved approximations for directed cut problems. In: Proceedings of the 39th ACM symposium on theory of computing (STOC), San Diego, pp 671–680
3. Arora S, Satish R, Vazirani U (2004) Expander flows, geometric embeddings, and graph partitionings. In: Proceedings of the 36th ACM symposium on theory of computing (STOC), Chicago, pp 222–231
4. Chawla S, Krauthgamer R, Kumar R, Rabani Y, Sivakumar D (2005) On the hardness of approximating sparsest cut and multicut. In: Proceedings of the 20th IEEE conference on computational complexity (CCC), San Jose, pp 144–153
5. Chuzhoy J, Khanna S (2007) Polynomial flow-cut gaps and hardness of directed cut problems. In: Proceedings of the 39th ACM symposium on theory of computing (STOC), San Diego, pp 179–188
6. Dahlhaus E, Johnson DS, Papadimitriou CH, Seymour PD, Yannakakis M (1994) The complexity of multiterminal cuts. SIAM Comput J 23(4):864–894
7. Fleischer L (1999) Approximating fractional multicommodity flow independent of the number of commodities. In: Proceedings of the 40th IEEE symposium on foundations of computer science (FOCS), New York, pp 24–31
8. Ford LR, Fulkerson DR (1956) Maximal flow through a network. Can J Math 8:399–404
9. Garg N, Könemann J (1998) Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: Proceedings of the 39th IEEE symposium on foundations of computer science (FOCS), pp 300–309

10. Garg N, Vazirani VV, Yannakakis M (1996) Approximate max-flow min-(multi)cut theorems and their applications. SIAM Comput J 25(2):235–251
11. Grigoriadis MD, Khachiyan LG (1996) Coordination complexity of parallel price-directive decomposition. Math Oper Res 21:321–340
12. Hu TC (1963) Multi-commodity network flows. Oper Res 11(3):344–360
13. Khot S, Vishnoi N (2005) The unique games conjecture, integrality gap for cut problems and the embeddability of negative-type metrics into l_1 In: Proceedings of the 46th IEEE symposium on foundations of computer science (FOCS), pp 53–62
14. Klein P, Agrawal A, Ravi R, Rao S (1990) Approximation through multicommodity flow. In: Proceedings of the 31st IEEE symposium on foundations of computer science (FOCS), pp 726–737
15. Linial N, London E, Rabinovich Y (1995) The geometry of graphs and some of its algorithmic applications. Combinatorica 15(2):215–245, Also in Proceedings of 35th FOCS, pp 577–591 (1994)
16. Shmoys DB (1997) Cut problems and their application to divide-and-conquer. In: Hochbaum DS (ed) Approximation algorithms for NP-hard problems. PWS, Boston, pp 192–235

Multidimensional Compressed Pattern Matching

Amihood Amir
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel
Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

Keywords

Multidimensional compressed search; Pattern matching in compressed images; Two-dimensional compressed matching

Years and Authors of Summarized Original Work

2003; Amir, Landau, Sokol

Problem Definition

Let c be a given compression algorithm, and let $c(D)$ be the result of c compressing data D. The *compressed search problem with compression algorithm c* is defined as follows.

INPUT: Compressed text $c(T)$ and pattern P.
OUTPUT: All locations in T where pattern P occurs.

A compressed matching algorithm is *optimal* if its time complexity is $O(|c(T)|)$.

Although optimality in terms of time is always important, when dealing with compression, the criterion of **extra space** is perhaps more important (Ziv, Personal communication, 1995). Applications employ compression techniques specifically because there is a limited amount of available space. Thus, it is not sufficient for a compressed matching algorithm to be optimal in terms of time, it must also satisfy the given space constraints. Space constraints may be due to limited amount of disk space (e.g., on a server), or they may be related to the size of the memory or cache. Note that if an algorithm uses as little extra space as the size of the cache, the runtime of the algorithm is also greatly reduced as no cache misses will occur [13]. It is also important to remember that in many applications, e.g., LZ compression on strings, the *compression ratio* – $|S|/|c(S)|$ – is a small constant. In a case where the compression ratio of the given text is a constant, an optimal compressed matching performs no better than the naive algorithm of decompressing the text. However, if the constants hidden in the *"big O"* are smaller than the compression ratio, then the compressed matching does offer a practical benefit. If those constants are larger than the optimal the compressed search algorithm may, in fact, be using more space than the uncompressed text.

Definition 1 (inplace) A compressed matching is said to be *inplace* if the extra space used is proportional to the input size of the pattern.

Note that this definition encompasses the compressed matching model (e.g., [2]) where the pattern is input in uncompressed form, as well as the *fully compressed* model [10], where the pattern is input in compressed form. The *inplace* requirement allows the extra space to be the input

size of the pattern, whatever that size may be. However, in many applications the compression ratio is a constant; therefore, a stronger space constraint is defined.

Definition 2 Let $\mathcal{AP}$ be the set of all patterns of size m, and let $c(\mathcal{AP})$ be the set of all compressed images of $\mathcal{AP}$. Let m' be the length of the smallest pattern in $c(\mathcal{AP})$. A compressed matching algorithm with input pattern P of length m is called *strongly inplace* if the amount of extra space used is proportional to m'.

The problem as defined above is equally applicable to textual (one-dimensional), image (two-dimensional), or any type of data, such as bitmaps, concordances, tables, XML data, or any possible data structure.

The compressed matching problem is considered crucial in image databases, since they are highly compressible. The initial definition of the compressed matching paradigm was motivated by the *two dimensional run-length compression*. This is the compression used for fax transmissions. The run-length compression is defined as follows.

Let $S = s_1 s_2 \cdots s_n$ be a string over some alphabet Σ. The *run-length compression* of string S is the string $S' = \sigma_1^{r_1}\sigma_2^{r_2}\cdots\sigma_k^{r_k}$ such that (1) $\sigma_i \neq \sigma_{i+1}$ for $1 \leq i < k$ and (2) S can be described as the concatenation of k *segments*, the symbol σ_1 repeated r_1 times, the symbol σ_2 repeated r_2 times, ..., and the symbol σ_k repeated r_k times. The *two-dimensional run-length compression* is the concatenation of the run-length compression of all the matrix rows (or columns).

The *two-dimensional run-length compressed matching problem* is defined as follows:
INPUT: Text array T of size $n \times n$, and pattern array P of size $m \times m$ *both* in two-dimensional run-length compressed form.
OUTPUT: All locations in T of occurrences of P. Formally, the output is the set of locations (i, j) such that $T[i+k, j+l] = P[k+1, l+1]k, l = 0 \ldots m-1$.

Another ubiquitous lossless two-dimensional compression is CompuServe's GIF standard, widely used on the World Wide Web. It uses LZW [19] (a variation of LZ78) on the image linearized row by row.

The *two-dimensional LZ compression* is formally defined as follows. Given an image $T[1 \ldots n, 1 \ldots n]$, create a string T_{lin} $[1 \ldots n^2]$ by concatenating all rows of T. Compressing T_{lin} with one-dimensional LZ78 yields the two-dimensional LZ compression of the image T.

The *two-dimensional LZ compressed matching problem* is defined as follows:
INPUT: Text array T of size $n \times n$, and pattern array P of size $m \times m$ *both* in two-dimensional LZ compressed form.
OUTPUT: All locations in T of occurrences of P. Formally, the output is the set of locations (i, j) such that $T[i+k, j+l] = P[k+1, l+1]$ $k, l = 0 \ldots m-1$.

Key Results

The definition of compressed search first appeared in the context of searching for two dimensional run-length compression [1, 2]. The following result was achieved there.

Theorem 1 (Amir and Benson [3]) *There exists an $O(|c(T)| \log |c(T)|)$ worst-case time solution to the compressed search problem with the two dimensional run-length compression algorithm.*

The above mentioned paper did not succeed in achieving either an optimal or an inplace algorithm. Nevertheless, it introduced the notion of *two-dimensional periodicity*. As in strings, periodicity plays a crucial rôle in two-dimensional string matching, and its advent has provided solutions to many longstanding open problems of two-dimensional string matching. In [5], it was used to achieve the first linear-time, alphabet-independent, two-dimensional text scanning. Later, in [4, 16] it was used in two different ways for a linear-time witness table construction. In [7] it was used to achieve the first parallel, time and work optimal, CREW algorithm for text scanning. A simpler variant of periodicity was used by [11] to obtain a constant-time CRCW algorithm for text scanning. A recent

further attempt has been made [17] to generalize periodicity analysis to higher dimensions.

The first optimal two-dimensional compressed search algorithm was the following.

Theorem 2 (Amir et al. [6]) *There exists an $O(|c(T)|)$ worst-case time solution to the compressed search problem with the two-dimensional run-length compression algorithm.*

Optimality was achieved by a concept the authors called *witness-free dueling*. The paper proved new properties of two-dimensional periodicity. This enables duels to be performed in which no witness is required. At the heart of the dueling idea lies the concept that two overlapping occurrences of a pattern in a text can use the content of a predetermined text position or witness in the overlap to eliminate one of them. Finding witnesses is a costly operation in a compressed text; thus, the importance of witness-free dueling.

The original algorithm of Amir et al. [6] takes time $O(|c(T)| + |P| \log \sigma)$, where σ is $\min(|P|, |\Sigma|)$, and Σ is the alphabet. However with the witness table construction of Galil and Park [12] the time is reduced to $O(|c(T)| + |P|)$. Using known techniques, one can modify their algorithm so that its extra space is $O(|P|)$. This creates an optimal algorithm that is also inplace, provided the pattern is input in uncompressed form. With use of the run-length compression, the difference between $|P|$ and $|c(P)|$ can be quadratic. Therefore it is important to seek an inplace algorithm.

Theorem 3 (Amir et al. [9]) *There exists an $O(|c(T)| + |P| \log \sigma)$ worst-case time solution to the compressed search problem with the two-dimensional run-length compression algorithm, where σ is $\min(|P|, |\Sigma|)$, and Σ is the alphabet, for all patterns that have no trivial rows (rows consisting of a single repeating symbol). The amount of space used is $O(|c(P)|)$.*

This algorithm uses the framework of the non-compressed two dimensional pattern matching algorithm of [6]. The idea is to use the **dueling** mechanism defined by Vishkin [18]. Applying the dueling paradigm directly to run-length compressed matching has previously been considered impossible since the location of a witness in the compressed text cannot be accessed in constant time. In [9], a way was shown in which a witness *can* be accessed in (amortized) constant time, enabling a relatively straightforward application of the dueling paradigm to compressed matching.

A strongly inplace compressed matching algorithm exists for the two-dimensional LZ compression, but its preprocessing is not optimal.

Theorem 4 (Amir et al. [8]) *There exists an $O(|c(T)| + |P|^3 \log \sigma)$ worst-case time solution to the compressed search problem with the two-dimensional LZ compression algorithm, where σ is $\min(|P|, |\Sigma|)$, and Σ is the alphabet. The amount of space used is $O(m)$, for an $n \times m$ size pattern. $O(m)$ is the best compression achievable for any $m \times m$ sized pattern under the two-dimensional LZ compression.*

The algorithm of [8] can be applied to any two-dimensional compressed text, in which the compression technique allows sequential decompression in small space.

Applications

The problem has many applications since two-dimensional data appears in many different types of compression. The two compressions discussed here are the run-length compression, used by fax transmissions, and the LZ compression, used by GIF.

Open Problems

Any lossless two-dimensional compression used, especially one with a large compression ratio, presents the problem of enabling the search without uncompressing the data for saving of both time and space.

Searching in two-dimensional lossy compressions will be a major challenge. Initial steps in this direction can be found in [14, 15], where JPEG compression is considered.

Cross-References

- Multidimensional Compressed Pattern Matching
- Multidimensional String Matching

Recommended Reading

1. Amir A, Benson G (1992) Efficient two dimensional compressed matching. In: Proceeding of data compression conference, Snow Bird, pp 279–288
2. Amir A, Benson G (1992) Two-dimensional periodicity and its application. In: Proceeding of 3rd symposium on discrete algorithms, Orlando, pp 440–452
3. Amir A, Benson G (1998) Two-dimensional periodicity and its application. SIAM J Comput 27(1):90–106
4. Amir A, Benson G, Farach M (1992) The truth, the whole truth, and nothing but the truth: alphabet independent two dimensional witness table construction. Technical Report GITCC-92/52, Georgia Institute of Technology
5. Amir A, Benson G, Farach M (1994) An alphabet independent approach to two dimensional pattern-matching. SIAM J Comput 23(2):313–323
6. Amir A, Benson G, Farach M (1997) Optimal two-dimensional compressed matching. J Algorithms 24(2):354–379
7. Amir A, Benson G, Farach M (1998) Optimal parallel two dimensional text searching on a crew pram. Inf Comput 144(1):1–17
8. Amir A, Landau G, Sokol D (2003) Inplace 2D matching in compressed images. J Algorithms 49(2):240–261
9. Amir A, Landau G, Sokol D (2003) Inplace run-length 2D compressed search. Theory Comput Sci 290(3):1361–1383
10. Berman P, Karpinski M, Larmore L, Plandowski W, Rytter W (1997) On the complexity of pattern matching for highly compressed two dimensional texts. In: Proceeding of 8th annual symposium on combinatorial pattern matching (CPM 97). LNCS, vol 1264. Springer, Berlin, pp 40–51
11. Crochemore M, Galil Z, Gasieniec L, Hariharan R, Muthukrishnan S, Park K, Ramesh H, Rytter W (1993) Parallel two-dimensional pattern matching. In: Proceeding of 34th annual IEEE FOCS, pp 248–258
12. Galil Z, Park K (1996) Alphabet-independent two-dimensional witness computation. SIAM J Comput 25(5):907–935
13. Hennessy JL, Patterson DA (1996) Computer architecture: a quantitative approach, 2nd edn. Morgan Kaufmann, San Francisco
14. Klein ST, Shapira D (2005) Compressed pattern matching in JPEG images. In: Proceeding Prague stringology conference, pp 125–134
15. Klein ST, Wiseman Y (2003) Parallel huffman decoding with applications to JPEG files. Comput J 46(5):487–497
16. Park K, Galil Z (1992) Truly alphabet-independent two-dimensional pattern matching. In: Proceeding 33rd IEEE FOCS, pp 247–256
17. Régnier M, Rostami L (1993) A unifying look at d-dimensional periodicities and space coverings. In: 4th symposium on combinatorial pattern matching, p 15
18. Vishkin U (1985) Optimal parallel pattern matching in strings. In: Proceeding 12th ICALP, pp 91–113
19. Welch TA (1984) A technique for high-performance data compression. IEEE Comput 17:8–19

Multidimensional String Matching

Juha Kärkkäinen
Department of Computer Science, University of Helsinki, Helsinki, Finland

Keywords

Approximate string matching; Combinatorial pattern matching; Image matching; Multidimensional array matching; Multidimensional string algorithms; Rotation invariance; Scaling invariance

Years and Authors of Summarized Original Work

1977; Bird
1978; Baker
1991; Amir, Landau
1994; Amir, Benson, Farach
1999; Kärkkäinen, Ukkonen
2000; Baeza-Yates, Navarro
2002; Fredriksson, Navarro, Ukkonen
2006; Amir, Kapah, Tsur
2009; Hundt, Liśkiewicz, Nevries
2010; Amir, Chencinski

Problem Definition

Given two two-dimensional arrays, the *text* $T[1 \ldots n, 1 \ldots n]$ and the *pattern* $P[1 \ldots m, 1 \ldots m]$, $m \leq n$, both with element values from *alphabet* Σ of size σ, the basic *two-dimensional*

string matching (2DSM) problem is to find all *occurrences* of P in T, i.e., all $m \times m$ subarrays of T that are identical to P. In addition to the basic problem, several types of generalizations are considered: *approximate matching* (allow local errors), *invariant matching* (allow global transformations), and *multidimensional matching*.

In approximate matching, an occurrence is a subarray S of the text, whose *distance* $d(S, P)$ from the pattern does not exceed a threshold k. Different distance measures lead to different variants of the problem. When no distance is explicitly mentioned, the *Hamming distance*, the number of mismatching elements, is assumed.

For one-dimensional strings, the most common distance is the Levenshtein distance, the minimum number of insertions, deletions, and substitutions for transforming one string into the other. A simple generalization to two dimensions is the *Krithivasan–Sitalakshmi (KS) distance*, which is the sum of row-wise Levenshtein distances. Baeza-Yates and Navarro [6] introduced several other generalizations, one of which, the *RC distance*, is defined as follows. A two-dimensional array can be decomposed into a sequence of rows and columns by removing either the last row or the last column from the array until nothing is left. Different decompositions are possible depending on whether a row or a column is removed at each step. The RC distance is the minimum cost of transforming a decomposition of one array into a decomposition of the other, where the minimum is taken over all possible decompositions as well as all possible transformations. A transformation consists of insertions, deletions, and modifications of rows and columns. The cost of inserting or deleting a row/column is the length of the row/column, and the cost of modification is the Levenshtein distance between the original and the modified row/column.

The invariant matching problems search for occurrences that match the pattern after some global transformation of the pattern. In the *scaling invariant matching* problem, an occurrence is a subarray that matches the pattern scaled by some factor. If only integral scaling factors are allowed, the definition of the problem is obvious. For real-valued scaling, a refined model is needed, where the text and pattern elements, called *pixels* in this case, are unit squares on a plane. Scaling the pattern means stretching the pixels. An occurrence is a matching M between text pixels and pattern pixels. The scaled pattern is placed on top of the text with one corner aligned, and each text pixel $T[r, s]$, whose center is covered by the pattern, is matched with the covering pattern pixel $P[r', s']$, i.e., $([r, s], [r', s']) \in M$.

In the *rotation invariant matching* problem, too, an occurrence is a matching between text pixels and pattern pixels. This time the center of the pattern is placed at the center of a text pixel, and the pattern is rotated around the center. The matching is again defined by which pattern pixels cover which text pixel centers.

All the problems can be generalized to more than two dimensions. In the d-dimensional problem, the text is an n^d array and the pattern an m^d array. The focus is on two dimensions, but multidimensional generalizations of the results are mentioned when they exist.

Many other variants of the problems are omitted here due to a lack of space. Some of them as well as some of the results in this entry are surveyed by Amir [1]. A wider range of problems as well as traditional image processing techniques for solving them can be found in [9].

Key Results

The classical solution to the 2DSM problem by Bird [8] and independently by Baker [7] reduces the problem to one-dimensional string matching. It has two phases:

1. Find all occurrences of pattern rows on the text rows and mark them. This takes $\mathcal{O}(n^2 \log \min(m, \sigma))$ time using the Aho–Corasick algorithm. On an integer alphabet $\Sigma = \{0, 1, \ldots, \sigma - 1\}$, the time can be improved to $\mathcal{O}(n^2 + m^2 \min(m, \sigma) + \sigma)$ using $\mathcal{O}(m^2 \min(m, \sigma) + \sigma)$ space.
2. The pattern is considered a sequence of m rows and each $n \times m$ subarray of the text a sequence of n rows. The Knuth–Morris–Pratt string matching algorithm is used for finding

the occurrences of the pattern in each subarray. The algorithm makes $\mathcal{O}(n)$ row comparisons for each of the $n - m + 1$ subarrays. With the markings from Step 1, a row comparison can be done in constant time, giving $\mathcal{O}(n^2)$ time complexity for Step 2.

The time complexity of the Bird–Baker algorithm is linear if the alphabet size σ is constant. The algorithm of Amir, Benson, and Farach [4] (with improvements by Galil and Park [13]) achieves linear time independent of the alphabet size using a quite different kind of algorithm based on string matching by duels and two-dimensional periodicity.

Theorem 1 (Bird [8]; Baker [7]; Amir, Benson, and Farach [4]) *The 2DSM problem can be solved in the optimal $\mathcal{O}(n^2)$ worst-case time.*

The Bird–Baker algorithm generalizes straightforwardly into higher dimensions by repeated application of Step 1 to reduce a problem in d dimensions into $n-m+1$ problems in $d-1$ dimensions. The time complexity is $\mathcal{O}(dn^d \log m^d)$. The Amir–Benson–Farach algorithm has been generalized to three dimensions with the time complexity $\mathcal{O}(n^3)$ [14].

The average-case complexity of the 2DSM problem was studied by Kärkkäinen and Ukkonen [16], who proved a lower bound and gave an algorithm matching the bound.

Theorem 2 (Kärkkäinen and Ukkonen [16]) *The 2DSM problem can be solved in the optimal $\mathcal{O}(n^2(\log_\sigma m)/m^2)$ average-case time.*

The result (both lower and upper bound) generalizes to the d-dimensional case with the $\Theta(n^d \log_\sigma m/m^d)$ average-case time complexity.

Amir and Landau [3] give algorithms for approximate 2DSM problems for both the Hamming distance and the KS distance. The RC model was developed and studied by Baeza–Yates and Navarro [6].

Theorem 3 (Amir and Landau [3]; Baeza–Yates and Navarro [6]) *The approximate 2DSM problem can be solved in $\mathcal{O}(kn^2)$ worst-case time for the Hamming distance, in $\mathcal{O}(k^2n^2)$ worst-case time for the KS distance, and in $\mathcal{O}(k^2mn^2)$ worst-case time for the RC distance.*

The results for the KS and RC distances generalize to d dimensions with the time complexities $\mathcal{O}(k(k+d)n^d)$ and $\mathcal{O}(d!m^{2d}n^d)$, respectively.

Approximate matching algorithms with good average-case complexity are described by Kärkkäinen and Ukkonen [16] for the Hamming distance and by Baeza–Yates and Navarro [6] for the KS and RC distances.

Theorem 4 (Kärkkäinen and Ukkonen [16]; Baeza–Yates and Navarro [6]) *The approximate 2DSM problem can be solved in $\mathcal{O}(kn^2(\log_\sigma m)/m^2)$ average-case time for the Hamming and KS distances and in $\mathcal{O}(n^2/m)$ average-case time for the RC distance.*

The results for the Hamming and the RC distance have d-dimensional generalizations with the time complexities $\mathcal{O}(kn^d(\log_\sigma m^d)/m^d)$ and $\mathcal{O}(kn^d/m^{d-1})$, respectively.

The scaling and rotation invariant 2DSM problems involve a continuous valued parameter (scaling factor or rotation angle). However, the corresponding matching between text and pattern pixels changes only at certain points, and there are only $\mathcal{O}(nm)$ effectively distinct scales and $\mathcal{O}(m^3)$ effectively distinct rotation angles. A separate search for each distinct scale or rotation would give algorithms with time complexities $\mathcal{O}(n^3m)$ and $\mathcal{O}(n^2m^3)$, but faster algorithms exist.

Theorem 5 (Amir and Chencinski [2]; Amir, Kapah, and Tsur [5]) *The scaling invariant 2DSM problem can be solved in $\mathcal{O}(n^2m)$ worst-case time, and the rotation invariant 2DSM problem in $\mathcal{O}(n^2m^2)$ worst-case time.*

Fast average-case algorithms for the rotation invariant problem are described by Fredriksson, Navarro, and Ukkonen [12]. They also consider approximate matching versions.

Theorem 6 (Fredriksson, Navarro, and Ukkonen [12]) *The rotation invariant 2DSM problem can be solved in the optimal $\mathcal{O}(n^2(\log_\sigma m)/m^2)$ average-case time. The rotation invariant approximate 2DSM problem can be solved in the optimal $\mathcal{O}(n^2(k+\log_\sigma m)/m^2)$ average-case time.*

Fredriksson, Navarro, and Ukkonen [12] also consider rotation invariant matching in d dimensions.

Hundt, Liśkiewicz, and Nevries [15] show that there are $\mathcal{O}(n^4m^2)$ effectively distinct combinations of scales and rotations and give an $\mathcal{O}(n^6m^2)$ time algorithm for finding the best match under a distance that generalizes the Hamming distance implying the following result.

Theorem 7 (Hundt, Liśkiewicz, and Nevries [15]) *The scaling and rotation invariant 2DSM and approximate 2DSM problems can be solved in $\mathcal{O}(n^6m^2)$ time.*

Applications

The main application area is pattern matching in images, particularly applications where the point of view in the image is well defined, such as aerial and astronomical photography, optical character recognition, and biomedical imaging. Even three-dimensional problems arise in biomedical applications [10].

Open Problems

There may be some room for improving the results under the combined scaling and rotation invariance using techniques similar to those in Theorems 5 and 6. Many combinations of the different variants of the problem have not been studied. With rotation invariant approximate matching under the RC distance even the problem needs further specification.

Experimental Results

No conclusive results exist though some experiments are reported in [10, 11, 15, 16].

Cross-References

- ▸ Approximate String Matching
- ▸ Indexed Two-Dimensional String Matching
- ▸ String Matching

Recommended Reading

1. Amir A (2005) Theoretical issues of searching aerial photographs: a bird's eye view. Int J Found Comput Sci 16(6):1075–1097
2. Amir A, Chencinski E (2010) Faster two dimensional scaled matching. Algorithmica 56(2): 214–234
3. Amir A, Landau GM (1991) Fast parallel and serial multidimensional approximate array matching. Theor Comput Sci 81(1):97–115
4. Amir A, Benson G, Farach M (1994) An alphabet independent approach to two-dimensional pattern matching. SIAM J Comput 23(2):313–323
5. Amir A, Kapah O, Tsur D (2006) Faster two-dimensional pattern matching with rotations. Theor Comput Sci 368(3):196–204
6. Baeza-Yates R, Navarro G (2000) New models and algorithms for multidimensional approximate pattern matching. J Discret Algorithms 1(1):21–49
7. Baker TP (1978) A technique for extending rapid exact-match string matching to arrays of more than one dimension. SIAM J Comput 7(4): 533–541
8. Bird RS (1977) Two dimensional pattern matching. Inf Process Lett 6(5):168–170
9. Brown LG (1992) A survey of image registration techniques. ACM Comput Surv 24(4):325–376
10. Fredriksson K, Ukkonen E (2000) Combinatorial methods for approximate pattern matching under rotations and translations in 3D arrays. In: Proceedings of the 7th international symposium on string processing and information retrieval, A Coruña. IEEE Computer Society, pp 96–104
11. Fredriksson K, Navarro G, Ukkonen E (2002) Faster than FFT: rotation invariant combinatorial template matching. In: Pandalai S (ed) Recent research developments in pattern recognition, vol II. Transworld Research Network, Trivandrum, pp 75–112
12. Fredriksson K, Navarro G, Ukkonen E (2005) Sequential and indexed two-dimensional combinatorial template matching allowing rotations. Theor Comput Sci 347(1–2):239–275
13. Galil Z, Park K (1996) Alphabet-independent two-dimensional witness computation. SIAM J Comput 25(5):907–935
14. Galil Z, Park JG, Park K (2004) Three-dimensional periodicity and its application to pattern matching. SIAM J Discret Math 18(2): 362–381
15. Hundt C, Liśkiewicz M, Nevries R (2009) A combinatorial geometrical approach to two-dimensional robust pattern matching with scaling and rotation. Theor Comput Sci 410(51): 5317–5333
16. Kärkkäinen J, Ukkonen E (1999) Two- and higher-dimensional pattern matching in optimal expected time. SIAM J Comput 29(2): 571–589

Multilevel Feedback Queues

Nikhil Bansal
Eindhoven University of Technology,
Eindhoven, The Netherlands

Keywords

Fairness; Low sojourn times; Scheduling with unknown job sizes

Years and Authors of Summarized Original Work

1968; Coffman, Kleinrock

Problem Definition

The problem is concerned with scheduling dynamically arriving jobs in the scenario when the processing requirements of jobs are unknown to the scheduler. This is a classic problem that arises, for example, in CPU scheduling, where users submit jobs (various commands to the operating system) over time. The scheduler is only aware of the existence of the job and does not know how long it will take to execute, and the goal is to schedule jobs to provide good quality of service to the users. Formally, this note considers the average flow time measure, defined as the average duration of time since a job is released until its processing requirement is met.

Notations

Let $\mathcal{J} = \{1, 2, \ldots, n\}$ denote the set of jobs in the input instance. Each job j is characterized by its release time r_j and its processing requirement p_j. In the online setting, job j is revealed to the scheduler only at time r_j. A further restriction is the *non-clairvoyant* setting, where only the existence of job j is revealed at r_j, in particular the scheduler does not know p_j until the job meets its processing requirement and leaves the system. Given a schedule, the completion time c_j of a job is the earliest time at which job j receives p_j amount of service. The flow time f_j of j is defined as $c_j - r_j$. A schedule is said to be preemptive, if a job can be interrupted arbitrarily, and its execution can be resumed later from the point of interruption without any penalty. It is well known that preemption is necessary to obtain reasonable guarantees even in the offline setting [4].

There are several natural non-clairvoyant algorithms such as first come first served, processor sharing (work on all current unfinished jobs at equal rate), and shortest elapsed time first (work on job that has received least amount of service thus far). Coffman and Kleinrock [2] proposed another natural algorithm known as the multilevel feedback queueing (MLF). MLF works as follows: there are queues $Q_0, Q_1, Q_2, \ldots$ and thresholds $0 < t_0 < t_1 < t_2 \ldots$. Initially upon arrival, a job is placed in Q_0. When a job in Q_i receives t_i amount of cumulative service, it is moved to Q_{i+1}. The algorithm at any time works on the lowest numbered nonempty queue. Coffman and Kleinrock analyzed MLF in a queuing theoretic setting, where the jobs arrive according to a Poisson process and the processing requirements are chosen identically and independently from a known probability distribution.

Recall that the online shortest remaining processing time (SRPT) algorithm that at any time works on the job with the least remaining processing time produces an optimum schedule. However, SRPT requires the knowledge of job sizes and hence is not non-clairvoyant. Since a non-clairvoyant algorithm only knows a lower bound on a jobs size (determined by the amount of service it has received thus far), MLF tries to mimic SRPT by favoring jobs that have received the least service thus far.

Key Results

While non-clairvoyant algorithms have been studied extensively in the queuing theoretic setting for many decades, this notion was considered relatively recently in the context of competitive

analysis by Motwani, Phillips, and Torng [5]. As in traditional competitive analysis, a non-clairvoyant algorithm is called c-competitive if for every input instance, its performance is no worse than c times that of the optimum offline solution for that instance. Motwani, Phillips, and Torng showed the following.

Theorem 1 ([5]) *For the problem of minimizing average flow time on a single machine, any deterministic non-clairvoyant algorithm must have a competitive ratio of at least $\Omega(n^{1/3})$, and any randomized algorithm must have a competitive ratio of at least $\Omega(\log n)$, where n is number of jobs in the instance.*

It is not too surprising that any deterministic algorithm must have a poor competitive ratio. For example, consider MLF where the thresholds are powers of 2, i.e., $1, 2, 4, \ldots$. Say $n = 2^k$ jobs of size $2^k + 1$ each arrive at times $0, 2^k, 2 \cdot 2^k, \ldots, (2^k - 1)2^k$, respectively. Then, it is easily verified that the average flow time under MLF is $\Omega(n^2)$, where as the average flow time is under the optimum algorithm is $\Omega(n)$.

Note that MLF performs poorly on the above instance since all jobs are stuck till the end with just one unit of work remaining. Interestingly, Kalyanasundaram and Pruhs [3] designed a randomized variant of MLF (known as RMLF) and proved that its competitive ratio is almost optimum. For each job j, and for each queue Q_i, the RMLF algorithm sets a threshold $t_{i,j}$ randomly and independently according to a truncated exponential distribution. Roughly speaking, setting a random threshold ensures that if a job is stuck in a queue, then its remaining processing is a reasonable fraction of its original processing time.

Theorem 2 ([3]) *The RMLF algorithm is $O(\log n \log \log n)$ competitive against an oblivious adversary. Moreover, the RMLF algorithm is $O(\log n \log \log n)$ competitive even against an adaptive adversary provided the adversary chooses all the job sizes in advance.*

Later, Becchetti and Leonardi [1] showed that in fact the RMLF is optimally competitive up to constant factors. They also analyzed RMLF on identical parallel machines.

Theorem 3 ([1]) *The RMLF algorithm is $O(\log n)$ competitive for a single machine. For multiple identical machines, RMLF achieves a competitive ratio of $O(\log n \log (\frac{n}{m}))$, where m is the number of machines.*

Applications

MLF and its variants are widely used in operating systems [6, 7]. These algorithms are not only close to optimum with respect to flow time but also have other attractive properties such as the amortized number of preemptions is logarithmic (preemptions occur only if a job arrives or departs or moves to another queue).

Open Problems

It is not known whether there exists a $o(n)$-competitive deterministic algorithm. It would be interesting to close the gap between the upper and lower bounds for this case. Often in real systems, even though the scheduler may not know the exact job size, it might have some information about its distribution based on historical data. An interesting direction of research could be to design and analyze algorithms that use this information.

Cross-References

- Flow Time Minimization
- Minimum Flow Time
- Shortest Elapsed Time First Scheduling

Recommended Reading

1. Becchetti L, Leonardi S (2004) Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. J ACM (JACM) 51(4):517–539
2. Coffman EG, Kleinrock L (1968) Feedback queueing models for time-shared systems. J ACM (JACM) 15(4):549–576
3. Kalyanasundaram B, Pruhs K (2003) Minimizing flow time nonclairvoyantly. J ACM (JACM) 50(4):551–567

4. Kellerer H, Tautenhahn T, Woeginger GJ (1999) Approximability and nonapproximability results for minimizing total flow time on a single machine. SIAM J Comput 28(4):1155–1166
5. Motwani R, Phillips S, Torng E (1994) Nonclairvoyant scheduling. Theor Comput Sci 130(1): 17–47
6. Nutt G (1999) Operating system projects using windows NT. Addison Wesley, Reading
7. Tanenbaum AS (1992) Modern operating systems. Prentice-Hall, Upper Saddle River

Multilinear Monomial Detection

Ioannis Koutis
Computer Science Department, University of Puerto Rico-Rio Piedras, San Juan, PR, USA

Keywords

Algebraic methods; Color coding; Parameterized algorithms; Subgraph containment

Years and Authors of Summarized Original Work

2008; Koutis
2009; Williams

Problem Definition

The topic of this article is the parameterized multilinear monomial detection problem:

> k-MLD: Given an arithmetic circuit C representing a polynomial $P(X)$ over $\mathbb{Z}_+$, decide whether $P(X)$ construed as a sum of monomials contains a multilinear monomial of degree k.

An *arithmetic circuit* is a directed acyclic graph with nodes corresponding to addition and multiplication gates, sources labeled with variables from a set X or positive integers, and one special terminal corresponding to the output gate. A *monomial* of degree k is a product of exactly k variables from X, and it is called *multilinear* if these k variables are distinct.

The k-MLD problem is arguably a fundamental problem, encompassing as special cases many natural and well-studied parameterized problems. Along with the algorithm for its solution, k-MLD provides a general framework for designing parameterized algorithms [11, 15]. The framework has yielded the fastest known algorithms for many parameterized problems, including all parameterized decision problems that were previously known to be solvable via dynamic programming combined with the color-coding method [2].

Key Results

Theorem 1 *The k-MLD problem can be solved by a randomized algorithm with one-sided error in $O^*(2^k)$ time and polynomial space. (The $O^*(\cdot)$ notation hides factors polynomial in the size of the input.)*

The algorithm claimed in Theorem 1 always reports the correct answer when the input polynomial does not contain a degree-k multilinear monomial. In the opposite case, it reports a correct answer with probability at least $1/4$.

Overview of the Algorithm

The algorithm utilizes a set of commutative matrices $\mathcal{M} \subseteq \mathbb{Z}_2^{2^k \times 2^k}$, with the following properties:

(i) For each $M \in \mathcal{M}$, we have $M^2 = \mathbf{0}\, mod\ 2$.
(ii) If $M_1, \ldots, M_k$ are randomly selected matrices from $\mathcal{M}$, then their product is equal to the "all-ones" matrix $1^{2^k \times 2^k}$, with probability at least $1/4$, and $0\ mod\ 2$ otherwise.

The construction of $\mathcal{M}$ is based on matrix representations of the group $\mathbb{Z}_2^k$, i.e., the abelian group of k-dimensional 0–1 vectors with addition mod 2 [11].

To simplify our discussion, let us assume that all monomials in $P(X)$ have total degree k. The role of $\mathcal{M}$ is then almost self-evident: evaluating

$P(X)$ on a random assignment $\bar{X} : X \to \mathcal{M}$ will annihilate (mod 2) all non-multilinear monomials in $P(X)$ due to property (i). On the other hand, each degree-k multilinear monomial will "survive" the assignment with constant probability, due to property (ii).

However, property (ii) clearly does not suffice for $P(X)$ to evaluate to nonzero (with some probability) in the presence of multilinear monomials. The main reason is that the coefficients of all multilinear monomials in $P(X)$ may be equal to 0 mod 2. The solution is the introduction of a new set A of "fingerprint" variables in order to construct an extended polynomial $\tilde{P}(X, A)$ over $\mathbb{Z}_2$. The key property of $\tilde{P}(X, A)$ is that its multilinear monomials are in a one-to-one correspondence with *copies* of the multilinear monomials in $P(X)$. Specifically, each copy of a multilinear monomial $\mu(X)$ of $P(X)$ gets its own distinct multilinear monomial of the form $q(A)\mu(X)$ in $\tilde{P}(X, A)$. The extended polynomial is constructed by applying simple operations on C. In most cases it is enough to attach a distinct "multiplier" variable from A to each edge of C; in the general case, some more work is needed that may increase the size of C by a quadratic factor in the worst case. We can then consider what is the effect of evaluating $\tilde{P}(X, A)$ on $\bar{X}$: (i) If $P(X)$ does not contain any degree-k multilinear monomial, then each monomial of $\tilde{P}(X, A)$ contains a squared variable from X. Hence $\tilde{P}(\bar{X}, A)$ is equal to $\mathbf{0}\ mod\ 2$. On the other hand, if $P(X)$ does contain a degree-k multilinear monomial, then, by construction, the diagonal entries of $\tilde{P}(\bar{X}, A)$ are all equal to a *nonzero* polynomial $Q(A)$, with probability at least $1/4$. Due to its size, we cannot afford to write down $Q(A)$, but we do have "black-box" access to it via evaluating it. We can thus test it for identity with zero via the Schwartz-Zippel Lemma [14]. This requires only a single evaluation of $Q(A)$ on a random assignment $\bar{A} : A \to GF[2^{\log_2 k+10}]$. Overall, the algorithm returns a "yes" if and only if $P(\bar{X}, \bar{A}) \neq \mathbf{0}\ mod\ 2$.

By the properties of $\mathcal{M}$, it suffices to compute the trace of $P(\bar{X}, \bar{A})$ or equivalently the sum of its eigenvalues. As observed in [11, 13], this can be done with 2^k evaluations of $P(X, A)$ over the ring of polynomials $\mathbb{Z}[A]$. This yields the $O^*(2^k)$ time and polynomial space claims.

The construction of an extended polynomial $\tilde{P}(X, A)$ was used in [11, 15] for two special cases, but it can be generalized to arbitrary polynomials as claimed in [13]. The idea to use the Schwartz-Zippel Lemma in order to test $Q(A)$ for identity with zero appeared in [15].

A Negative Result

The matrices in $\mathcal{M}$ together with multiplication and addition modulo 2 form a commutative matrix algebra $\mathcal{A}$ which has 2^{2^k} elements. Computations with this algebra require time at least 2^k, since merely describing one element of $\mathcal{A}$ requires 2^k bits. One may ask whether there is another significantly smaller algebra that can replace $\mathcal{A}$ in this evaluation-based framework and yield a faster algorithm. The question was answered in the negative in [13], for the special but important case when $k = |X|$. Concretely, it was shown that there is no evaluation-based algorithm that can detect a multilinear term of degree n in an n-variate polynomial in $o(2^n/\sqrt{n})$ time.

A Generalization

The CONSTRAINED k-MLD problem is a generalization of k-MLD that was introduced in [12]. The set X is a union of t *mutually disjoint* sets of variables $X_1, \ldots, X_t$, each associated with a positive number μ_i. The sets X_i and the numbers μ_i, for $i = 1, \ldots, t$ are part of the input. A multilinear monomial is *permissible* if it contains at most μ_i variables from X_i, for all i. The problem is then defined as follows:

> CONSTRAINED k-MLD: Given an arithmetic circuit C representing a polynomial $P(X)$ over $\mathbb{Z}_+$, decide whether $P(X)$ construed as a sum of monomials contains a permissible multilinear monomial of degree k.

Theorem 2 *The* CONSTRAINED k-MLD *problem can be solved by a randomized algorithm with one-sided error in* $O^*(2^k)$ *time and polynomial space.*

Theorem 2 was shown in [5]. It is worth noting that the algorithm in the proof of Theorem 2 does

not rely on matrix algebras. Thus, it provides an alternative proof for Theorem 1.

Applications

Many parameterized problems are reducible to the k-MLD problem. For several of these problems, the fastest – in terms of the exponential dependence on k – known algorithms are composed by a relatively simple reduction to a k-MLD instance and a subsequent invocation of the algorithm for its solution. Such problems include the k-TREE problem on directed graphs and certain packing problems [11, 13, 15].

The k-MLD framework provides an exponentially faster alternative to the color-coding method [2] for parameterized decision problems. As noted in [8], color-coding-based algorithms consist canonically of a random assignment of k colors to entities of the input (e.g., to the vertices of graph) followed by a dynamic programming procedure. From the steps of the dynamic programming procedure, one can delineate the construction of a k-MLD instance. This task was, for example, undertaken in [9], giving improved algorithms for all subgraph containment problems previously solved via color coding.

The algebraic language provided by the k-MLD framework has simplified the design of parameterized algorithms, yielding faster algorithms even for problems for which the applicability of color coding was not apparent due to the more complicated underlying dynamic programming procedures. This includes partial graph domination problems [13] and parameterized string problems in computational biology [6].

All *algebraizations* used in [11, 13] construct polynomials whose monomials are in one-to-one correspondence with potential solutions of the underlying combinatorial problem (e.g., length-k walks). The actual solutions (e.g., k-paths) are mapped to multilinear monomials, while nonsolutions are mapped to non-multilinear monomials. Andreas Björklund introduced a significant departure from this approach [4]. He viewed modulo 2 computations as a resource rather as a nuisance and worked on sharper algebraizations using appropriate determinant polynomials. These polynomials do contain multilinear monomials corresponding to nonsolutions, unlike previous algebraizations; however, these come *in pairs* and they cancel out modulo 2. On the other hand, with the appropriate use of "fingerprint" variables, the multilinear monomials corresponding to valid solutions appear with a coefficient of 1. Björklund's novel ideas led initially to a faster algorithm for d-dimensional matching problems [4] and subsequently to the breakthrough $O(1.67^n)$ time algorithm for the HAMILTONIAN PATH problem on n-vertex graphs [3], breaking below the $O^*(2^n)$ barrier and marking the first progress in the problem after nearly 50 years of stagnation.

Modulo 2 cancelations were also explicitly exploited in the design of the first single-exponential time algorithms for various graph connectivity problems parameterized by *treewidth* [7]. For example, the HAMILTONIAN PATH problem can now be solved in $O^*(4^t)$ time for n-vertex graphs, assuming a tree decomposition of width at most t is given as input. The previously fastest algorithm runs in $O^*(t^{O(t)})$ time.

Open Problems

The following problems are open:

- Color coding stood until recently as the fastest *deterministic* algorithm for the k-MLD problem, running in $O^*((2e)^k)$ time, the currently fastest deterministic algorithm in $O^*(2.85^k)$ time [10]. Is there a deterministic algorithm for k-MLD that runs in $O^*(2^k)$ time?
- The known deterministic algorithms can also solve the *weighted* version of k-PATH which asks for a k-path of minimum weight, in $O^*(2.85^k \log W)$ time, where W is the largest edge weight in the graph. The algorithm for k-MLD can be adapted to solve weighted k-PATH in $O^*(2^k W)$ time. Is there an $O^*(2^k \log W)$ time algorithm for weighted

k-PATH and, more generally, for weighted versions of k-MLD?

- Color coding with balanced hashing families has been used to approximately count k-paths in a graph, in $O^*((2e)^k)$ time [1]. Is there an $O^*(2^k)$ approximate counting algorithm?
- Finally, is there an algorithm for k-MLD that runs in $O^*(2^{\epsilon k})$ time, for any $\epsilon < 1$? Such an algorithm would constitute major progress in exact and parameterized algorithms for NP-hard problems.

Cross-References

▸ Color Coding
▸ Treewidth of Graphs

Recommended Reading

1. Alon N, Gutner S (2009) Balanced hashing, color coding and approximate counting. In: Chen J, Fomin FV (eds) Parameterized and exact computation. Springer, Berlin/Heidelberg, pp 1–16
2. Alon N, Yuster R, Zwick U (1995) Color coding. J ACM 42(4):844–856
3. Björklund A (2010) Determinant sums for undirected hamiltonicity. In: 51th annual IEEE symposium on foundations of computer science, FOCS 2010, Las Vegas, 23–26 Oct 2010, pp 173–182
4. Björklund A (2010) Exact covers via determinants. In: 27th international symposium on theoretical aspects of computer science, STACS, Nancy, pp 95–106
5. Björklund A, Kaski P, Kowalik L (2015) Constrained multilinear detection and generalized graph motifs. Algorithmica 1–21. doi:10.1007/s00453-015-9981-1
6. Blin G, Bonizzoni P, Dondi R, Sikora F (2012) On the parameterized complexity of the repetition free longest common subsequence problem. Inf Process Lett 112(7):272–276
7. Cygan M, Nederlof J, Pilipczuk M, Pilipczuk M, van Rooij JMM, Wojtaszczyk JO (2011) Solving connectivity problems parameterized by treewidth in single exponential time. In: IEEE 52nd annual symposium on foundations of computer science, FOCS, Palm Springs, pp 150–159
8. Downey RG, Fellows MR (2013) Fundamentals of parameterized complexity. Texts in computer science. Springer. doi:10.1007/978-1-4471-5559-1
9. Fomin FV, Lokshtanov D, Raman V, Saurabh S, Rao BVR (2012) Faster algorithms for finding and counting subgraphs. J Comput Syst Sci 78(3):698–706. doi:10.1016/j.jcss.2011.10.001
10. Fomin FV, Lokshtanov D, Saurabh S (2014) Efficient computation of representative sets with applications in parameterized and exact algorithms. In: Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms. SIAM, Philadelphia, pp 142–151
11. Koutis I (2008) Faster algebraic algorithms for path and packing problems. In: Proceedings of the 35th international colloquium on automata, languages and programming, Part I. Springer, Berlin/Heidelberg, pp 575–586
12. Koutis I (2012) Constrained multilinear detection for faster functional motif discovery. Inf Process Lett 112(22):889–892
13. Koutis I, Williams R (2009) Limits and applications of group algebras for parameterized problems. In: Proceedings of the 36th international colloquium on automata, languages and programming: Part I (ICALP '09). Springer, Berlin/Heidelberg, pp 653–664
14. Motwani R, Raghavan P (1997) Randomized algorithms. In: Tucker AB (ed) The computer science and engineering handbook. CRC Press, Boca Raton, pp 141–161
15. Williams R (2009) Finding paths of length k in $O^*(2^k)$ time. Inf Process Lett 109:315–318

Multiple String Matching

Maxime Crochemore[1,2,3] and Thierry Lecroq[4]
[1]Department of Computer Science, King's College London, London, UK
[2]Laboratory of Computer Science, University of Paris-East, Paris, France
[3]Université de Marne-la-Vallée, Champs-sur-Marne, France
[4]Computer Science Department and LITIS Faculty of Science, Université de Rouen, Rouen, France

Keywords

Failure function; Pattern matching; Shift function; String matching; Trie; DAWG

Years and Authors of Summarized Original Work

1975; Aho, Corasick
1979; Commentz-Walter

1999; Crochemore, Czumaj, Gąsieniec, Lecroq, Plandowski, Rytter

Problem Definition

Given a finite set of k *pattern strings* $\mathcal{P} = \{P^1, P^2, \ldots, P^k\}$ and a *text string* $T = t_1 t_2 \ldots t_n$, T and the P^is being sequences over an alphabet Σ of size σ, the *multiple string matching (MSM)* problem is to find one or, more generally, all the text positions where a P^i occurs in T. More precisely the problem is to compute the set $\{j \mid \exists i, P^i = t_j t_{j+1} \ldots t_{j+|P^i|-1}\}$, or equivalently the set $\{j \mid \exists i, P^i = t_{j-|P^i|+1} t_{j-|P^i|+2} \ldots t_j\}$. Note that reporting all the occurrences of the patterns may lead to a quadratic output (e.g., when P^is and T are drawn from a one-letter alphabet). The length of the shortest pattern in $\mathcal{P}$ is denoted by ℓmin. This problem is an extension of the exact string matching problem.

Both worst- and average-case complexities are considered. For the latter one assumes that pattern and text are randomly generated by choosing each character uniformly and independently from Σ. For simplicity and practicality the assumption $|P^i| = o(n)$ is made, for $1 \le i \le k$, in this entry.

Key Results

A first solution to the multiple string matching problem consists in applying an exact string matching algorithm for locating each pattern in $\mathcal{P}$. This solution has an $O(kn)$ worst-case time complexity. There are more efficient solutions along two main approaches. The first one, due to Aho and Corasick [1], is an extension of the automaton-based solution for matching a single string. The second approach, initiated by Commentz-Walter [5], extends the Boyer-Moore algorithm to several patterns.

The Aho-Corasick algorithm first builds a trie $T(\mathcal{P})$, a digital tree recognizing the patterns of $\mathcal{P}$. The trie $T(\mathcal{P})$ is a tree whose edges are labeled by letters and whose branches spell the patterns of $\mathcal{P}$. A node p in the trie $T(\mathcal{P})$ is associated with the unique word w spelled by the path of $T(\mathcal{P})$ from its root to p. The root itself is identified with the empty word ε. Notice that if w is a node in $T(\mathcal{P})$, then w is a prefix of some $P^i \in \mathcal{P}$. If in addition $a \in \Sigma$, then $child(w, a)$ is equal to wa if wa is a node in $T(\mathcal{P})$; it is equal to NIL otherwise.

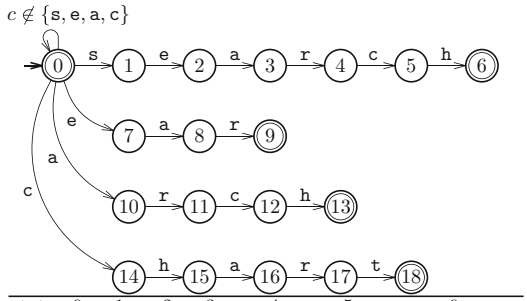
During a second phase, when patterns are added to the trie, the algorithm initializes an output function *out*. It associates the singleton $\{P^i\}$ with the nodes P^i $(1 \le i \le k)$ and associates the empty set with all other nodes of $T(\mathcal{P})$.

Finally, the last phase of the preprocessing consists in building a failure link for each node of the trie and simultaneously completing the output function. The failure function *fail* is defined on nodes as follows (w is a node): $fail(w) = u$ where u is the longest proper suffix of w that belongs to $T(\mathcal{P})$. Computation of failure links is done during a breadth-first traversal of $T(\mathcal{P})$. Completion of the output function is done while computing the failure function *fail* using the following rule: if $fail(w) = u$, then $out(w) = out(w) \cup out(u)$.

To stop going back with failure links during the computation of the failure links, and also to overpass text characters for which no transition is defined from the root during the searching phase, a loop is added on the root of the trie for these symbols. This finally produces what is called a pattern matching machine or an Aho-Corasick automaton (see Fig. 1).

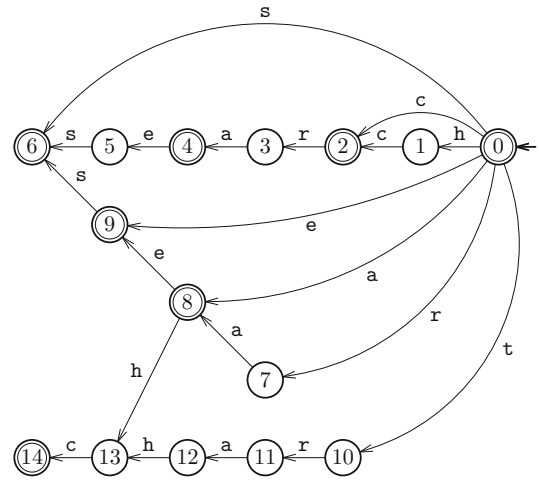
After the preprocessing phase is completed, the searching phase consists in parsing the text T with $T(\mathcal{P})$. This starts at the root of $T(\mathcal{P})$ and uses failure links whenever a character in T does not match any label of outgoing edges of the current node. Each time a node with a nonempty output is encountered, this means that the patterns of the output have been discovered in the text, ending at the current position. Then, the position is output.

Theorem 1 (Aho and Corasick [1]) *After preprocessing $\mathcal{P}$, searching for the occurrences of the strings of $\mathcal{P}$ in a text T can be done in time $O(n \times \log \sigma)$. The running time of the associated preprocessing phase is $O(|\mathcal{P}| \times \log \sigma)$. The extra memory space required for both operations is $O(|\mathcal{P}|)$.*



state	0	1	2	3	4	5	6
prefix	ε	s	se	sea	sear	searc	search
fail	0	0	7	8	9	12	13
out					{ear}		{search,arch}
state	7	8	9	10	11	12	13
prefix	e	ea	ear	a	ar	arc	arch
fail	0	10	11	0	0	14	15
out		{ear}				{arch}	
state	14	15	16	17	18	19	
prefix	c	ch	cha	char	chart		
fail	0	0	10	11	0		
out					{chart}		

Multiple String Matching, Fig. 1 The pattern matching machine or Aho-Corasick automaton for the set of strings {search, ear, arch, chart}



Multiple String Matching, Fig. 2 An example of DAWG, index structure used for matching the set of strings {search, ear, arch, chart}. The automaton accepts the reverse prefixes of the strings

The Aho-Corasick algorithm is actually a generalization to a finite set of strings of the Morris-Pratt exact string matching algorithm.

Commentz-Walter [5] generalized the Boyer-Moore exact string matching algorithm to multiple string matching. Her algorithm builds a trie for the reverse patterns in $\mathcal{P}$ together with two shift tables and applies a right to left scan strategy. However, it is intricate to implement and has a quadratic worst-case time complexity.

The DAWG-match algorithm [6] is a generalization of the BDM exact string matching algorithms. It consists in building an exact indexing structure for the reverse strings of $\mathcal{P}$ such as a factor automaton or a generalized suffix tree, instead of just a trie as in the Aho-Corasick and Commentz-Walter algorithms (see Fig. 2). The overall algorithm can be made optimal by using both an indexing structure for the reverse patterns and an Aho-Corasick automaton for the patterns. Then, searching involves scanning some portions of the text from left to right and some other portions from right to left. This enables to skip large portions of the text T.

Theorem 2 (Crochemore et al. [6]) *The DAWG-match algorithm performs at most* $2n$ *symbol comparisons. Assuming that the sum of the length of the patterns in* $\mathcal{P}$ *is less than* ℓmin^k*, the DAWG-match algorithm makes on average* $O((n \log_\sigma \ell min)/\ell min)$ *inspections of text characters.*

The bottleneck of the DAWG-match algorithm is the construction time and space consumption of the exact indexing structure. This can be avoided by replacing the exact indexing structure by a factor oracle for a set of strings. A factor oracle is a simple automaton that may recognize a few additional strings comparing to exact indexing structure. When the factor oracle is used alone, it gives the Set Backward Oracle Matching (SBOM) algorithm [2]. It is an exact algorithm that behaves almost as well as the DAWG-match algorithm.

The bit-parallelism technique can be used to simulate the DAWG-match algorithm. It gives the MultiBNDM algorithm of Navarro and Raffinot [9]. This strategy is efficient when $k \times \ell min$ bits fit in a few computer words. The prefixes of strings of $\mathcal{P}$ of length ℓmin are packed together in a bit vector. Then, the search is similar to the

BNDM exact string matching and is performed for all the prefixes at the same time.

The use of the generalization of the bad-character shift alone as done in the Horspool exact string matching algorithm gives poor performances for the MSM problem due to the high probability of finding each character of the alphabet in one of the strings of $\mathcal{P}$.

The algorithm of Wu and Manber [13] considers blocks of length ℓ. Blocks of such a length are hashed using a function h into values less than *maxvalue*. Then $shift[h(B)]$ is defined as the minimum between $|P^i| - j$ and $\ell min - \ell + 1$ with $B = p^i_{j-\ell+1} \dots p^i_j$ for $1 \le i \le k$ and $1 \le j \le |P^i|$. The value of ℓ varies with the minimum length of the strings in $\mathcal{P}$ and the size of the alphabet. The value of *maxvalue* varies with the memory space available.

The searching phase of the algorithm consists in reading blocks B of length ℓ. If $shift[h(B)] > 0$, then a shift of length $shift[h(B)]$ is applied. Otherwise, when $shift[h(B)] = 0$, the patterns ending with block B are examined one by one in the text. The first block to be scanned is $t_{\ell min-\ell+1} \dots t_{\ell min}$. This method is incorporated in the *agrep* command [12].

Recent works have been devoted to multiple string matching on packed strings where each symbol is encoded using $\log \sigma$ bits. In this context, Belazzougui [3] gave an efficient algorithm that works in $O(n + ((\log k + \log \ell min + \log\log|\mathcal{P}|)/\ell min + (\log\sigma)/\omega) + occ)$ where ω is the size of the machine word and *occ* is the number of occurrences of patterns of $\mathcal{P}$ in T. On average it is possible to solve the problem in $O(n/\ell min)$ time using $O(|\mathcal{P}|\log|\mathcal{P}|)$ bits of space [4].

Applications

MSM algorithms serve as basis for multidimensional pattern matching and approximate pattern matching with wildcards. The problem has many applications in computational biology, database search, bibliographic search, virus detection in data flows, and several others.

Experimental Results

The book of G. Navarro and M. Raffinot [10] is a good introduction to the domain. It presents experimental graphics that report experimental evaluation of multiple string matching algorithms for different alphabet sizes, pattern lengths, and sizes of pattern set.

URLs to Code and Data Sets

Well-known packages offering efficient MSM are *agrep* (https://github.com/Wikinaut/agrep) and *grep* with the -F option (http://www.gnu.org/software/grep/grep.html).

Cross-References

- Multidimensional String Matching is the case where the text dimension is greater than one.
- Regular Expression Matching is the more complex case where the pattern can be a regular expression;
- String Matching is the version where a single pattern is searched for in a text;
- Suffix Trees and Arrays refers to the case where the text can be preprocessed;

Further information can be found in the three following books: [7, 8] and [11].

Recommended Reading

1. Aho AV, Corasick MJ (1975) Efficient string matching: an aid to bibliographic search. C ACM 18(6):333–340
2. Allauzen C, Crochemore M, Raffinot M (1999) Factor oracle: a new structure for pattern matching. In: SOFSEM'99. LNCS, vol 1725, Milovy, Czech Republic, pp 291–306
3. Belazzougui D (2012) Worst-case efficient single and multiple string matching on packed texts in the word-ram model. J Discret Algorithms 14:91–106
4. Belazzougui D, Raffinot M (2013) Average optimal string matching in packed strings. In: Spirakis PG, Serna MJ (eds) Proceedings of the 8th international conference on algorithms and complexity (CIAC

2013), Barcelona. Lecture notes in computer science, vol 7878. Springer, Barcelona, Spain, pp 37–48
5. Commentz-Walter B (1979) A string matching algorithm fast on the average. In: Proceedings of ICALP'79. Lecture notes in computer science vol 71. Springer, Graz, Austria, pp 118–132
6. Crochemore M, Czumaj A, Gąsieniec L, Lecroq T, Plandowski W, Rytter W (1999) Fast practical multi-pattern matching. Inf Process Lett 71(3–4):107–113
7. Crochemore M, Hancart C, Lecroq T (2007) Algorithms on strings. Cambridge University Press, Cambridge, New York
8. Gusfield D (1997) Algorithms on strings, trees and sequences. Cambridge University Press, Cambridge, New York
9. Navarro G, Raffinot M (2000) Fast and flexible string matching by combining bit-parallelism and suffix automata. ACM J Exp Algorithms 5:4
10. Navarro G, Raffinot M (2002) Flexible pattern matching in strings – practical on-line search algorithms for texts and biological sequences. Cambridge University Press, Cambridge
11. Smyth WF (2002) Computing patterns in strings. Addison Wesley Longman, Harlow
12. Wu S, Manber U (1992) Agrep – a fast approximate pattern-matching tool. In: Proceedings of USENIX winter 1992 technical conference. USENIX Association, San Francisco, CA, pp 153–162
13. Wu S, Manber U (1994) A fast algorithm for multi-pattern searching. Report TR-94-17, Department of Computer Science, University of Arizona, Tucson

Multiple Unit Auctions with Budget Constraint

Tian-Ming Bu
Software Engineering Institute, East China Normal University, Shanghai, China

Keywords

Auction design; Optimal mechanism design

Years and Authors of Summarized Original Work

2005; Borgs, Chayes, Immorlica, Mahdian, Saberi
2006; Abrams

Problem Definition

In this problem, an auctioneer would like to sell an idiosyncratic commodity with m copies to n bidders, denoted by $i = 1, 2, \ldots, n$. Each bidder i has two kinds of privately known information: $t_i^u \in \mathbb{R}^+$, $t_i^b \in \mathbb{R}^+$. t_i^u represents the price buyer i is willing to pay for per copy of the commodity and t_i^b represents i's budget.

Then a *one-round sealed-bid* auction proceeds as follows. *Simultaneously*, all the bidders submit their bids to the auctioneer. When receiving the reported unit value vector $\mathbf{u} = (u_1, \ldots, u_n)$ and the reported budget vector $\mathbf{b} = (b_1 \ldots, b_n)$ of bids, the auctioneer computes and outputs the allocation vector $\mathbf{x} = (x_1, \ldots, x_n)$ and the price vector $\mathbf{p} = (p_1, \ldots, p_n)$. Each element of the allocation vector indicates the number of copies allocated to the corresponding bidder. If bidder i receives x_i copies of the commodity, he pays the auctioneer $p_i x_i$. Then bidder i's total payoff is $(t_i^u - p_i)x_i$ if $x_i p_i \leq t_i^b$ and $-\infty$ otherwise. Correspondingly, the revenue of the auctioneer is $\mathcal{A}(\mathbf{u}, \mathbf{b}, m) = \sum_i p_i x_i$.

If each bidder submits his privately *true* unit value t_i^u and budget t_i^b to the auctioneer, the auctioneer can determine the single price $p_{\mathcal{F}}$ (i.e., $\forall i,\ p_i = p_{\mathcal{F}}$) and the allocation vector which maximize the auctioneer's revenue. This optimal single price revenue is denoted by $\mathcal{F}(\mathbf{u}, \mathbf{b}, m)$.

Interestingly, in this problem, we assume bidders have free will and have complete knowledge of the auction mechanism. Bidders would just report the bid (maybe different from his corresponding privately true values) which could maximize his payoff according to the auction mechanism.

So the objective of the problem is to design a *truthful* auction satisfying *voluntary participation* to raise the auctioneer's revenue as much as possible. An auction is *truthful* if for every bidder i, bidding his true valuation would maximize his payoff, regardless of the bids submitted by the other bidders [7,8]. An auction satisfies *voluntary participation* if each bidder's payoff is guaranteed to be nonnegative if he reports his bid truthfully. The success of the auction $\mathcal{A}$ is determined by competitive ratio β which is defined as the

upper bound of $\frac{\mathcal{F}(\mathbf{u},\mathbf{b},m)}{\mathcal{A}(\mathbf{u},\mathbf{b},m)}$ [6]. Clearly, the smaller competitive ratio β is, the better the auction $\mathcal{A}$ is.

Definition 1 (Multiple-Unit Auctions with Budget Constraint)

INPUT: the number of copies m, the submitted unit value vector $\mathbf{u}$, and the submitted budget vector $\mathbf{b}$.

OUTPUT: the allocation vector $\mathbf{x}$ and the price vector $\mathbf{p}$.

CONSTRAINTS:

(a) Truthful;
(b) Voluntary participation;
(c) $\sum_i x_i \leq m$.

Key Results

Let $b_{\max}$ denote the largest budget among the bidders receiving copies in the optimal solution and define $\alpha = \frac{\mathcal{F}}{b_{\max}}$.

Theorem 1 ([3]) *A truthful auction satisfying voluntary participation with competitive ratio* $1/\max_{0<\delta<1}\left\{(1-\delta)(1-2e^{-\frac{\alpha\delta^2}{36}})\right\}$ *can be designed.*

Theorem 2 ([1]) *A truthful auction satisfying voluntary participation with competitive ratio* $\frac{4\alpha}{\alpha-1}$ *can be designed.*

Theorem 3 ([1]) *If* α *is known in advance, then a truthful auction satisfying voluntary participation with competitive ratio* $\frac{(x\alpha+1)\alpha}{(x\alpha-1)^2}$ *can be designed, where* $x = \frac{\alpha-1+((\alpha-1)^2-4\alpha)^{1/2}}{2\alpha}$.

Theorem 4 ([1]) *For any truthful randomized auction* $\mathcal{A}$ *satisfying voluntary participation, the competitive ratio is at least* $2-\epsilon$ *when* $\alpha \geq 2$.

Applications

This problem is motivated by the development of IT industry and the popularization of auctions, especially, auctions on the Internet. Multiple copy auctions of relatively low-value goods, such as the auction of online ads for search terms to bidders with budget constraints, is assuming a very important role. Companies such as Google and Yahoo!'s revenue depends almost on certain types of auctions. There are many papers including [2,4,5] which focus on different facets of the same model.

Cross-References

▸ Competitive Auction

Recommended Reading

1. Abrams Z (2006) Revenue maximization when bidders have budgets. In: Proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithms (SODA-06), Miami, pp 1074–1082
2. Bhattacharya S, Conitzer V, Munagala K, Xia L (2010) Incentive compatible budget elicitation in multi-unit auctions. In: Proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms (SODA-10), Austin, pp 554–572
3. Borgs C, Chayes JT, Immorlica N, Mahdian M, Saberi A (2005) Multi-unit auctions with budget-constrained bidders. In: ACM conference on electronic commerce (EC-05), Vancouver, pp 44–51
4. Bu TM, Qi Q, Sun AW (2008) Unconditional competitive auctions with copy and budget constraints. Theor Comput Sci 393(1–3):1–13
5. Dobzinski S, Lavi R, Nisan N (2012) Multi-unit auctions with budget limits. Games Econ Behav 74(2):486–503
6. Fiat A, Goldberg AV, Hartline JD, Karlin AR (2002) Competitive generalized auctions. In: Proceedings of the 34th annual ACM symposium on theory of computing (STOC-02), New York, pp 72–81
7. Nisan N, Ronen A (1999) Algorithmic mechanism design. In: Proceedings of the thirty-first annual ACM symposium on theory of computing (STOC-99), New York, pp 129–140
8. Parkes DC (2004) Chapter 2: iterative combinatorial auctions. PhD thesis, University of Pennsylvania

Multiplex PCR for Gap Closing (Whole-Genome Assembly)

Vera Asodi
Center for the Mathematics of Information, California Institute of Technology, Pasadena, CA, USA

Keywords

Multiplex PCR; Whole genome assemble

Years and Authors of Summarized Original Work

2002; Alon, Beigel, Kasif, Rudich, Sudakov

Problem Definition

This problem is motivated by an important and timely application in computational biology that arises in whole-genome shotgun sequencing. Shotgun sequencing is a high throughput technique that has resulted in the sequencing of a large number of bacterial genomes as well as Drosophila (fruit fly) and Mouse and the celebrated Human genome (at Celera) (see, e.g., [8]). In all such projects, one is left with a collection of DNA fragments. These fragments are subsequently assembled, in-silico, by a computational algorithm. The typical assembly algorithm repeatedly merges overlapping fragments into longer fragments called contigs. For various biological and computational reasons some regions of the DNA cannot covered by the contigs. Thus, the contigs must be ordered and oriented and the gaps between them must be sequenced using slower, more tedious methods. For further details see, e.g., [3]. When the number of gaps is small (e.g., less than ten) biologists often use combinatorial PCR. This technique initiates a set of "bi-directional molecular walks" along the gaps in the sequence; these walks are facilitated by PCR. In order to initiate the molecular walks biologists use primers. Primers are designed so that they bind to unique (with respect to the entire DNA sequence) templates occurring at the end of each contig. A primer (at the right temperature and concentration) anneals to the designated unique DNA substring and promotes copying of the template starting from the primer binding site, initiating a one-directional walk along the gap in the DNA sequence. A PCR reaction occurs, and can be observed as a DNA ladder, when two primers that bind to positions on two ends of the same gap are placed in the same test tube.

If there are N contigs, the combinatorial (exhaustive) PCR technique tests all possible pairs (quadratically many) of $2N$ primers by placing two primers per tube with the original uncut DNA strand. PCR products can be detected using gels or they can be read using sequencing technology or DNA mass-spectometry. When the number of gaps is large, the quadratic number of PCR experiments is prohibitive, so primers are pooled using $K > 2$ primers per tube; this technique is called multiplex PCR [4]. This problem deals with finding optimal strategies for pooling the primers to minimize the number of biological experiments needed in the gap-closing process.

This problem can be modeled as the problem of identifying or learning a hidden matching given a vertex set V and an allowed query operation: for a subset $F \subseteq V$, the query Q_F is "does F contain at least one edge of the matching"? In this formulation each vertex represents a primer, an edge of the matching represents a reaction, and the query represents checking for a reaction when a set of primers are combined in a test tube. The objective is to identify the matching asking as few queries as possible, that is performing as few tests as possible. For further discussion of this model see [3, 7].

This problem is of interest even in the deterministic, fully non-adaptive case. A family $\mathcal{F}$ of subsets of a vertex set V solves the matching problem on V if for any two distinct matchings M_1 and M_2 on V there is at least one $F \in \mathcal{F}$ that contains an edge of one of the matchings and does not contain any edge of the other. Obviously, any such family enables learning an unknown matching deterministically and non-adaptively, by asking the questions Q_F for each $F \in \mathcal{F}$. The objective here is to determine the minimum possible cardinality of a family that solves the matching problem on a set of n vertices.

Other interesting variants of this problem are when the algorithm may be randomized, or when it is adaptive, that is when the queries are asked in k rounds, and the queries of each round may depend on the answers from the previous rounds.

Key Results

In [2], the authors study the number of queries needed to learn a hidden matching in several

models. Following is a summary of the main results presented in this paper.

The trivial upper bound on the size of a family that solves the matching problem on n vertices is $\binom{n}{2}$, achieved by the family of all pairs of vertices. Theorem 1 shows that in the deterministic non-adaptive setting one cannot do much better than this, namely, that the trivial upper bound is tight up to a constant factor. Theorem 2 improves this upper bound by showing a family of approximately half that size that solves the matching problem.

Theorem 1 *For every $n > 2$, every family $\mathcal{F}$ that solves the matching problem on n vertices satisfies*

$$|\mathcal{F}| \geq \frac{49}{153}\binom{n}{2}.$$

Theorem 2 *For every n there exists a family of size*

$$\left(\frac{1}{2} + o(1)\right)\binom{n}{2}$$

that solves the matching problem on n vertices.

Theorem 3 shows that one can do much better using randomized algorithms. That is, one can learn a hidden matching asking only $O(n \log n)$ queries, rather than order of n^2. These randomized algorithms make no errors, however, they might ask more queries with some small probability.

Theorem 3 *The matching problem on n vertices can be solved by probabilistic algorithms with the following parameters:*

- *2 rounds and $(1/(2\ln 2))n \log n(1 + o(1)) \approx 0.72n \log n$ queries*
- *1 round and $(1/\ln 2)n \log n(1 + o(1)) \approx 1.44n \log n$ queries.*

Finally, Theorem 4 considers adaptive algorithms. In this case there is a tradeoff between the number of queries and the number of rounds. The more rounds one allows, the fewer tests are needed, however, as each round can start only after the previous one is completed, this increases the running time of the entire procedure.

Theorem 4 *For all $3 \leq k \leq \log n$, there is a deterministic k-round algorithm for the matching problem on n vertices that asks*

$$O\left(n^{1+\frac{1}{2(k-1)}}(\log n)^{1+\frac{1}{k-1}}\right)$$

queries per round.

Applications

As described in section "Problem Definition", this problem was motivated by the application of gap closing in whole-genome sequencing, where the vertices correspond to primers, the edges to PCR reactions between pairs of primers that bind to the two ends of a gap, and the queries to tests in which a set of primers are combined in a test tube.

This gap-closing problem can be stated more generally as follows. Given a set of chemicals, a guarantee that each chemical reacts with at most one of the others, and an experimental mechanism to determine whether a reaction occurs when several chemicals are combined in a test tube, the objective is to determine which pairs of chemicals react with each other with a minimum number of experiments.

Another generalization which may have more applications in molecular biology is when the hidden subgraph is not a matching but some other fixed graph, or a family of graphs. The paper [2], as well as some other related works (e.g., [1, 5, 6]), consider this generalization for other graphs. Some of these generalizations have other specific applications in molecular biology.

Open Problems

- Determine the smallest possible constant c such that there is a deterministic non-adaptive algorithm for the matching problem on n vertices that performs $c\binom{n}{2}(1 + o(1))$ queries.

- Find more efficient deterministic k-round algorithms or prove lower bounds for the number of queries in such algorithms.
- Find efficient algorithms and prove lower bounds for the generalization of the problem to graphs other than matchings.

Recommended Reading

1. Alon N, Asodi V (2004) Learning a hidden subgraph. In: ICALP. LNCS, vol 3142, pp 110–121; Also: SIAM J Discr Math 18:697–712 (2005)
2. Alon N, Beigel R, Kasif S, Rudich S, Sudakov B (2002) Learning a hidden matching. In: Proceedings of the 43rd IEEE FOCS, pp 197–206; Also: SIAM J Comput 33:487–501 (2004)
3. Beigel R, Alon N, Apaydin MS, Fortnow L, Kasif S (2001) An optimal procedure for gap closing in whole genome shotgun sequencing. In: Proceedings of RECOMB. ACM, pp 22–30
4. Burgart LJ, Robinson RA, Heller MJ, Wilke WW, Iakoubova OK, Cheville JC (1992) Multiplex polymerase chain reaction. Mod Pathol 5:320–323
5. Grebinski V, Kucherov G (1997) Optimal query bounds for reconstructing a Hamiltonian cycle in complete graphs. In: Proceedings of 5th Israeli symposium on theoretical computer science, pp 166–173
6. Grebinski V, Kucherov G (1998) Reconstructing a Hamiltonian cycle by querying the graph: application to DNA physical mapping. Discret Appl Math 88:147–165
7. Tettelin H, Radune D, Kasif S, Khouri H, Salzberg S (1999) Pipette optimal multiplexed PCR: efficiently closing whole genome shotgun sequencing project. Genomics 62:500–507
8. Venter JC, Adams MD, Sutton GG, Kerlavage AR, Smith HO, Hunkapiller M (1998) Shotgun sequencing of the human genome. Science 280:1540–1542

Multitolerance Graphs

George B. Mertzios
School of Engineering and Computing Sciences, Durham University, Durham, UK

Keywords

Intersection model; Maximum clique; Minimum coloring; Maximum-weight independent set; Multitolerance graphs; Tolerance graphs

Years and Authors of Summarized Original Work

2011; Mertzios

Problem Definition

Tolerance graphs model interval relations in such a way that intervals can tolerate a certain degree of overlap without being in conflict. A graph $G = (V, E)$ on n vertices is a *tolerance* graph if there exists a collection $I = \{I_v \mid v \in V\}$ of closed intervals on the real line and a set $t = \{t_v \mid v \in V\}$ of positive numbers, such that for any two vertices $u, v \in V$, $uv \in E$ if and only if $|I_u \cap I_v| \geq \min\{t_u, t_v\}$, where $|I|$ denotes the length of the interval I.

Tolerance graphs have been introduced in [3], in order to generalize some of the well-known applications of interval graphs. If in the definition of tolerance graphs we replace the operation "min" between tolerances by "max," we obtain the class of *max-tolerance* graphs [7]. Both tolerance and max-tolerance graphs have attracted many research efforts (e.g., [4, 5, 7–10]) as they find numerous applications, especially in bioinformatics, constraint-based temporal reasoning, and resource allocation problems, among others [4, 5, 7, 8]. In particular, one of their applications is in the comparison of DNA sequences from different organisms or individuals by making use of a software tool like BLAST [1].

In some circumstances, we may want to treat different parts of the genomic sequences in BLAST nonuniformly, since for instance some of them may be biologically less significant or we have less confidence in the exact sequence due to sequencing errors in more error-prone genomic regions. That is, we may want to be more tolerant at some parts of the sequences than at others. This concept leads naturally to the notion of *multitolerance* (known also as *bitolerance*) graphs [5, 11]. The main idea is to allow two different tolerances to each interval, one to the left and one to the right side, respectively. Then, every interval tolerates in its interior part the

intersection with other intervals by an amount that is a convex combination of these two border tolerances.

Formally, let $I = [l, r]$ be a closed interval on the real line and $l_t, r_t \in I$ be two numbers between l and r, called *tolerant points*; note that it is not necessary that $l_t \leq r_t$. For every $\lambda \in [0, 1]$, we define the interval $I_{l_t,r_t}(\lambda) = [l + (r_t - l)\lambda, l_t + (r - l_t)\lambda]$, which is the convex combination of $[l, l_t]$ and $[r_t, r]$. Furthermore, we define the set $\mathcal{I}(I, l_t, r_t) = \{I_{l_t,r_t}(\lambda) \mid \lambda \in [0, 1]\}$ of intervals. That is, $\mathcal{I}(I, l_t, r_t)$ is the set of all intervals that we obtain when we linearly transform $[l, l_t]$ into $[r_t, r]$. For an interval I, the *set of tolerance intervals* τ of I is defined either as $\tau = \mathcal{I}(I, l_t, r_t)$ for some values $l_t, r_t \in I$ of tolerant points or as $\tau = \{\mathbb{R}\}$. A graph $G = (V, E)$ is a *multitolerance* graph if there exists a collection $I = \{I_v \mid v \in V\}$ of closed intervals and a family $t = \{\tau_v \mid v \in V\}$ of sets of tolerance intervals, such that for any two vertices $u, v \in V$, $uv \in E$ if and only if there exists an element $Q_u \in \tau_u$ with $Q_u \subseteq I_v$ or there exists an element $Q_v \in \tau_v$ with $Q_v \subseteq I_u$. Then, the pair $\langle I, t\rangle$ is called a *multitolerance representation* of G. Tolerance graphs are a special case of multitolerance graphs.

Note that, in general, the adjacency of two vertices u and v in a multitolerance graph G depends on both sets of tolerance intervals τ_u and τ_v. However, since the real line $\mathbb{R}$ is not included in any finite interval, if $\tau_u = \{\mathbb{R}\}$ for some vertex u of G, then the adjacency of u with another vertex v of G depends *only* on the set τ_v of v. If G has a multitolerance representation $\langle I, t\rangle$, in which $\tau_v \neq \{\mathbb{R}\}$ for every $v \in V$, then G is called a *bounded multitolerance* graph. Bounded multitolerance graphs coincide with trapezoid graphs, i.e., the intersection graphs of trapezoids between two parallel lines L_1 and L_2 on the plane, and have received considerable attention in the literature [5, 11]. However, the trapezoid intersection model cannot cope with general multitolerance graphs, in which it can be $\tau_v = \{\mathbb{R}\}$ for some vertices v. Therefore, the only way until now to deal with general multitolerance graphs was to use the inconvenient multitolerance representation, which uses an infinite number of tolerance intervals.

Key Results

In this entry we introduce the first nontrivial intersection model for general multitolerance graphs, given by objects in the 3-dimensional space, called *trapezoepipeds*. This *trapezoepiped representation* unifies in a simple and intuitive way the widely known trapezoid representation for bounded multitolerance graphs and the parallelepiped representation for tolerance graphs [9]. The main idea is to exploit the third dimension to capture the information of the vertices with $\tau_v = \{\mathbb{R}\}$ as the set of tolerance intervals. This intersection model can be constructed efficiently (in linear time), given a multitolerance representation.

Apart from being important on its own, the trapezoepiped representation can be also used to design efficient algorithms and structural results. Given a multitolerance graph with n vertices and m edges, we present algorithms that compute a minimum coloring and a maximum clique in $O(n \log n)$ time (which turns out to be *optimal*), and a maximum-weight independent set in $O(m + n \log n)$ time (where $\Omega(n \log n)$ is a lower bound for the complexity of this problem [2]). Moreover, a variation of this algorithm can compute a maximum-weight independent set in optimal $O(n \log n)$ time, when the input is a tolerance graph, thus closing the complexity gap of [9].

Given a multitolerance representation of a graph $G = (V, E)$, vertex $v \in V$ is called *bounded* if $\tau_v = \mathcal{I}(I_v, l_{t_v}, r_{t_v})$ for some values $l_{t_v}, r_{t_v} \in I_v$. Otherwise, v is *unbounded*. V_B and V_U are the sets of bounded and unbounded vertices in V, respectively. Clearly $V = V_B \cup V_U$.

Definition 1 For a vertex $v \in V_B$ (resp. $v \in V_U$) in a multitolerance representation of G, the values $t_{v,1} = l_{t_v} - l_v$ and $t_{v,2} = r_v - r_{t_v}$ (resp. $t_{v,1} = t_{v,2} = \infty$) are the *left tolerance* and the *right tolerance* of v, respectively. Moreover, if $v \in V_U$, then $t_v = \infty$ is the *tolerance* of v.

It can be easily seen by Definition 1 that if we set $t_{v,1} = t_{v,2}$ for every vertex $v \in V$, then we obtain a tolerance representation, in which

$t_{v,1} = t_{v,2}$ is the (unique) tolerance of v. Let now L_1 and L_2 be two parallel lines at unit distance in the plane.

Definition 2 Given an interval $I_v = [l_v, r_v]$ and tolerances $t_{v,1}, t_{v,2}$, $\overline{T}_v$ is the trapezoid in $\mathbb{R}^2$ defined by the points c_v, b_v on L_1 and a_v, d_v on L_2, where $a_v = l_v$, $b_v = r_v$, $c_v = \min\{r_v, l_v + t_{v,1}\}$, and $d_v = \max\{l_v, r_v - t_{v,2}\}$. The values $\phi_{v,1} = \operatorname{arc\,cot}(c_v - a_v)$ and $\phi_{v,2} = \operatorname{arc\,cot}(b_v - d_v)$ are the *left slope* and the *right slope* of $\overline{T}_v$, respectively. Moreover, for every unbounded vertex $v \in V_U$, $\phi_v = \phi_{v,1} = \phi_{v,2}$ is the *slope* of $\overline{T}_v$.

Note that, in Definition 2, the endpoints a_v, b_v, c_v, d_v of any trapezoid $\overline{T}_v$ (on the lines L_1 and L_2) lie on the plane $z = 0$ in $\mathbb{R}^3$. Therefore, since we assumed that the distance between the lines L_1 and L_2 is one, these endpoints of $\overline{T}_v$ correspond to the points $(a_v, 0, 0)$, $(b_v, 1, 0)$, $(c_v, 1, 0)$, and $(d_v, 0, 0)$ in $\mathbb{R}^3$, respectively. For the sake of presentation, we may not distinguish in the following between these points in $\mathbb{R}^3$ and the corresponding real values a_v, b_v, c_v, d_v, whenever this slight abuse of notation does not cause any confusion.

We are ready to give the main definition of this entry, namely, the *trapezoepiped representation*. For a set X of points in $\mathbb{R}^3$, denote by $H_{\text{convex}}(X)$ the *convex hull* defined by the points of X. That is, $\overline{T}_v = H_{\text{convex}}(a_v, b_v, c_v, d_v)$ for every vertex $v \in V$ by Definition 2, where a_v, b_v, c_v, d_v are points of the plane $z = 0$ in $\mathbb{R}^3$.

Definition 3 (trapezoepiped representation) Let $G = (V, E)$ be a multitolerance graph with a multitolerance representation $\{I_v = [a_v, b_v], \tau_v \mid v \in V\}$ and $\Delta = \max\{b_v \mid v \in V\} - \min\{a_v \mid v \in V\}$ be the greatest distance between two interval endpoints. For every vertex $v \in V$, the *trapezoepiped* T_v of v is the convex set of points in $\mathbb{R}^3$ defined as follows:

(a) If $t_{v,1}, t_{v,2} \leq |I_v|$ (i.e., v is bounded), then $T_v = H_{\text{convex}}(\overline{T}_v, a'_v, b'_v, c'_v, d'_v)$.
(b) If $t_v = t_{v,1} = t_{v,2} = \infty$ (i.e., v is unbounded), then $T_v = H_{\text{convex}}(a'_v, c'_v)$.

Where $a'_v = (a_v, 0, \Delta - \cot\phi_{v,1})$, $b'_v = (b_v, 1, \Delta - \cot\phi_{v,2})$, $c'_v = (c_v, 1, \Delta - \cot\phi_{v,1})$, and $d'_v = (d_v, 0, \Delta - \cot\phi_{v,2})$. The set of trapezoepipeds $\{T_v \mid v \in V\}$ is a *trapezoepiped representation* of G (Fig. 1).

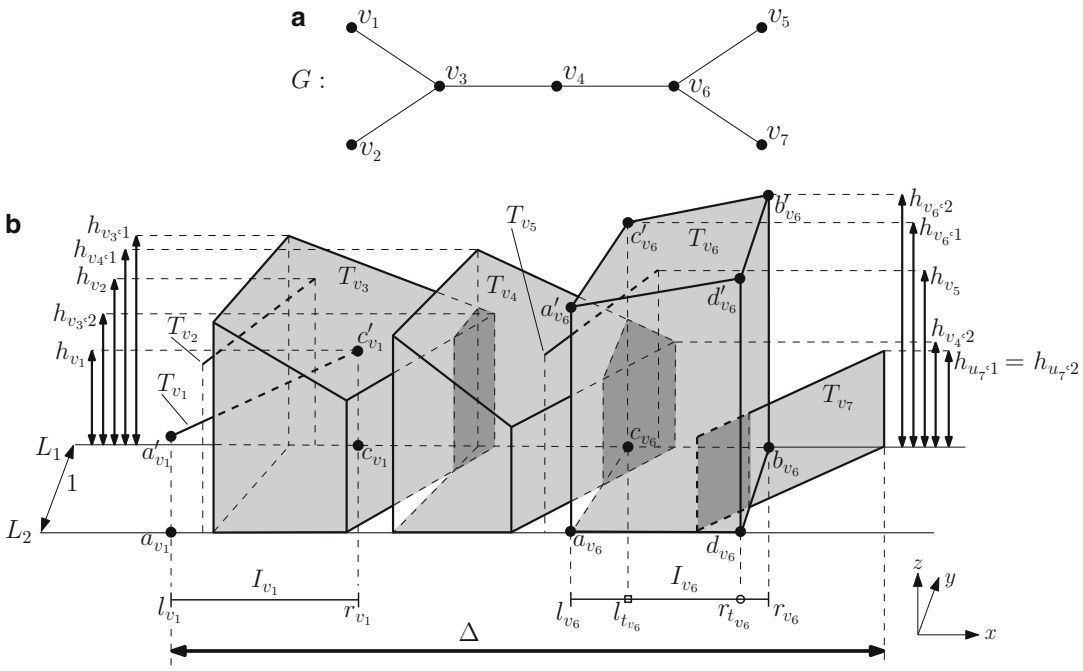
Theorem 1 *Let $G = (V, E)$ be a multitolerance graph with a multitolerance representation $\{I_v = [a_v, b_v], \tau_v \mid v \in V\}$. Then for every $u, v \in V$, $uv \in E$ if and only if $T_u \cap T_v \neq \emptyset$.*

Efficient Algorithms

As one of our main tools towards providing efficient algorithms on multitolerance graphs, we refine Definition 3 by introducing the notion of a *canonical* trapezoepiped representation. A trapezoepiped representation R of a multitolerance graph $G = (V, E)$ is called *canonical* if the following is true: for every unbounded vertex $v \in V_U$ in R, if we replace T_v by $H_{\text{convex}}(\overline{T}_v, a'_v, c'_v)$ in R, we would create a new edge in G. Note that replacing T_v by $H_{\text{convex}}(\overline{T}_v, a'_v, c'_v)$ in R is equivalent to replacing in the corresponding multitolerance representation of G the infinite tolerance $t_v = \infty$ by the finite tolerances $t_{v,1} = t_{v,2} = |I_v|$, i.e., to making v a bounded vertex. Clearly, every trapezoepiped representation R can be transformed to a canonical one by iteratively replacing unbounded vertices with bounded ones (without introducing new edges), as long as this is possible. Using techniques from computational geometry, we can prove the next theorem.

Theorem 2 *Every trapezoepiped representation of a multitolerance graph G with n vertices can be transformed to a canonical representation of G in $O(n \log n)$ time.*

The main idea for the proof of Theorem 2 is the following. We associate with every unbounded vertex $v \in V_U$ an (appropriately defined) point p_v and with every bounded vertex $u \in V_B$ three points $p_{u,1}, p_{u,2}, p_{u,3}$ in the plane. Furthermore we associate with every bounded vertex $u \in V_B$ the two line segments $\ell_{u,1}$ and $\ell_{u,2}$ in the plane, which have the points $\{p_{u,1}, p_{u,2}\}$ and $\{p_{u,2}, p_{u,3}\}$ as endpoints, respectively. We can prove that an unbounded vertex $v \in V_U$ can



Multitolerance Graphs, Fig. 1 (**a**) A multitolerance graph G and (**b**) a trapezoepiped representation R of G. Here, $h_{v_i,j} = \Delta - \cot \phi_{v_i,j}$ for every bounded vertex $v_i \in V_B$ and $j \in \{1, 2\}$, while $h_{v_i} = \Delta - \cot \phi_{v_i}$ for every unbounded vertex $v_i \in V_U$

be replaced by a bounded vertex without introducing a new edge if and only if, in the above construction, the point p_v lies above the *lower envelope* $\mathrm{Env}(L)$ of the line segments $L = \{\ell_{u,1}, \ell_{u,2} : u \in V_B\}$. Since $|L| = O(n)$, we can compute $\mathrm{Env}(L)$ in $O(n \log n)$ time using the algorithm of [6].

In the resulting canonical representation R' of G, for every unbounded vertex $v \in V_U$, there exists at least one bounded vertex $u \in V_B$ such that $uv \notin E$ and T_v lies "above" T_u in R'. Moreover, we can prove that in this case $N(v) \subseteq N(u)$, and thus there exists a minimum coloring of G where u and v have the same color. The main idea for our (optimal) $O(n \log n)$-time minimum coloring algorithm is the following. We first compute in $O(n \log n)$ time a minimum coloring of the induced subgraph $G[V_B]$ using the coloring algorithm of [2] for trapezoid graphs. Then, given this coloring, we assign in linear time a color to all unbounded vertices. Furthermore, using Theorem 2, the maximum clique algorithm of [2] for trapezoid graphs, and the fact that multitolerance graphs are perfect, we provide an (optimal) $O(n \log n)$-time maximum clique algorithm for multitolerance graphs.

Our $O(m + n \log n)$-time maximum-weight independent set algorithm for multitolerance graphs is based on dynamic programming. During its execution, the algorithm uses binary search trees to maintain two finite sets M and H of $O(n)$ weighted markers each, which are appropriately sorted on the real line. For the case where the input graph G is a tolerance graph, this algorithm can be slightly modified to compute a maximum-weight independent set in (optimal) $O(n \log n)$ time, thus closing the complexity gap of [9].

Classification of Multitolerance Graphs

Apart from its use in devising efficient algorithms, the trapezoepiped representation proved useful also in classifying multitolerance graphs inside the hierarchy of perfect graphs that is given in [5, Figure 2.8]. The resulting hierarchy of classes of perfect graphs is *complete*, i.e., all inclusions are strict.

Open Problems

The trapezoepiped representation provides geometric insight for multitolerance graphs, and it can be expected to prove useful in deriving new algorithmic as well as structural results. It remains open to close the gap between the lower bound of $\Omega(n \log n)$ and the upper bound of $O(m + n \log n)$ for the weighted independent set on general multitolerance graphs. Furthermore, interesting open problems for further research include the weighted clique problem, the Hamiltonian cycle problem, the dominating set problem, as well as the recognition problem of general multitolerance graphs.

Recommended Reading

1. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990) Basic local alignment search tool. J Mol Biol 215(3):403–410
2. Felsner S, Müller R, Wernisch L (1997) Trapezoid graphs and generalizations, geometry and algorithms. Discret Appl Math 74:13–32
3. Golumbic MC, Monma CL (1982) A generalization of interval graphs with tolerances. In: Proceedings of the 13th Southeastern conference on combinatorics, graph theory and computing, Boca Raton. Congressus Numerantium, vol 35, pp 321–331
4. Golumbic MC, Siani A (2002) Coloring algorithms for tolerance graphs: reasoning and scheduling with interval constraints. In: Proceedings of the joint international conferences on artificial intelligence, automated reasoning, and symbolic computation (AISC/Calculemus), Marseille, pp 196–207
5. Golumbic MC, Trenk AN (2004) Tolerance graphs. Cambridge studies in advanced mathematics. Cambridge University Press, Cambridge
6. Hershberger J (1989) Finding the upper envelope of n line segments in $O(n \log n)$ time. Inf Process Lett 33(4):169–174
7. Kaufmann M, Kratochvil J, Lehmann KA, Subramanian AR (2006) Max-tolerance graphs as intersection graphs: cliques, cycles, and recognition. In: Proceedings of the 17th annual ACM-SIAM symposium on discrete algorithms (SODA), Miami, pp 832–841
8. Lehmann KA, Kaufmann M, Steigele S, Nieselt K (2006) On the maximal cliques in c-max-tolerance graphs and their application in clustering molecular sequences. Algorithms Mol Biol 1:9
9. Mertzios GB, Sau I, Zaks S (2009) A new intersection model and improved algorithms for tolerance graphs. SIAM J Discret Math 23(4):1800–1813
10. Mertzios GB, Sau I, Zaks S (2011) The recognition of tolerance and bounded tolerance graphs. SIAM J Comput 40(5):1234–1257
11. Parra A (1998) Triangulating multitolerance graphs. Discret Appl Math 84(1–3):183–197

Multiway Cut

Gruia Calinescu
Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA

Keywords

Multiterminal cut

Years and Authors of Summarized Original Work

1998; Calinescu, Karloff, Rabani

Problem Definition

Given an undirected graph with edge costs and a subset of k nodes called *terminals*, a *multiway cut* is a subset of edges whose removal disconnects each terminal from the rest. MULTIWAY CUT is the problem of finding a multiway cut of minimum cost.

Previous Work

Dahlhaus, Johnson, Papadimitriou, Seymour, and Yannakakis [6] initiated the study of MULTIWAY CUT and proved that MULTIWAY CUT is MAX SNP-hard even when restricted to instances with three terminals and unit edge costs. Therefore, unless $P = NP$, there is no polynomial-time approximation scheme for MULTIWAY CUT. For $k = 2$, the problem is identical to the undirected version of the extensively studied $s-t$ min-cut problem of Ford and Fulkerson, and thus has polynomial-time algorithms (see, e.g., [1]). Prior to this paper, the best (and essentially the only) approximation algorithm for $k \geq 3$ was due to

the above-mentioned paper of Dahlhaus et al. They give a very simple combinatorial *isolation heuristic* that achieves an approximation ratio of $2(1-1/k)$. Specifically, for each terminal i, find a minimum-cost cut separating i from the remaining terminals, and then output the union of the $k-1$ cheapest of the k cuts. For $k=4$ and for $k=8$, Alon (see [6]) observed that the isolation heuristic can be modified to give improved ratios of 4/3 and 12/7, respectively.

In special cases, far better results are known. For fixed k in planar graphs, the problem is solvable in polynomial time [6]. For trees and 2-trees, there are linear-time algorithms [5]. For dense unweighted graphs, there is a polynomial-time approximation scheme [2, 8].

Key Results

Theorem 1 ([3]) *There is a deterministic polynomial time algorithm that finds a multiway cut of cost at most $(1.5-1/k)$ times the optimum multiway cut.*

The approximation algorithm from Theorem 1 is based on a novel linear programming relaxation described later. On the basis of the same linear program, the approximation ratio was subsequently improved to 1.3438 by Karger, Klein, Stein, Thorup, and Young [10]. For three terminals, [10] and Cheung, Cunningham, and Tang [4] give very different 12/11-approximation algorithms.

Two variations of the problem have been considered in the literature: Garg, Vazirani, and Yannakakis [9] obtain a $(2-2/k)$-approximation ratio for the node-weighted version, and Naor and Zosin [11] obtain 2-approximation for the case of directed graphs. It is known that any approximation ratio for these variations translates immediately into the same approximation ratio for VERTEX COVER, and thus it is hard to get any significant improvement over the approximation ratio of 2.

The algorithm from Theorem 1 appears next, giving a flavor of how this result is obtained. The complete proof of the approximation ratio is not long and appears in [3] or the book [12].

Notation

Let $G=(V,E)$ be an undirected graph on $V=\{1,2,\ldots,n\}$ in which each edge $uv \in E$ has a non-negative cost $c(u,v)=c(v,u)$, and let $T=\{1,2,\ldots,k\} \subseteq V$ be a set of *terminals*. MULTIWAY CUT is the problem of finding a minimum cost set $C \subseteq E$ such that in $(V, E \setminus C)$, each of the terminals $1, 2, \ldots, k$ is in a different component. Let $\mathrm{MWC} = \mathrm{MWC}(G)$ be the value of the optimal solution to MULTIWAY CUT.

Δ_k denotes the $(k-1)$-simplex, i.e., the $(k-1)$-dimensional convex polytope in $\mathbb{R}^k$ given by $\{x \in \mathbb{R}^k | (x \geq 0) \wedge (\sum_i x_i = 1)\}$.

For $x \in \mathbb{R}^k$, $\|x\|$ is its L_1 norm: $\|x\| = \sum_i |x_i|$. For $j=1,2,\ldots,k$, $e^j \in \mathbb{R}^k$ denotes the unit vector given by $(e^j)_j = 1$ and $(e^j)_i = 0$ for all $i \neq j$.

LP-Relaxation

The *simplex* relaxation for MULTIWAY CUT with edge costs has as variables k-dimensional real vectors x^u, defined for each vertex $u \in V$:

$$\text{Minimize } \frac{1}{2} \sum_{uv \in E} c(u,v) \cdot \|x^u - x^v\|$$

Subject to:

$$x^u \in \Delta_k \quad \forall u \in V$$

$$x^t = e^t \quad \forall t \in T.$$

In other words, the terminals stay at the vertices of the $(k-1)$-simplex, and the other nodes anywhere in the simplex, and measure an edge's length by the total variation distance between its endpoints. Clearly, placing all nodes at simplex vertices gives an integral solution: the lengths of edges are either 0 (if both endpoints are at the same vertex) or 1 (if the endpoints are at different vertices), and the removal of all unit length edges disconnects the graph into at least k components, each containing at most one terminal.

To solve this relaxation as a linear program, new variables are introduced: y^{uv}, defined for all $uv \in E$, and x_i^u, defined for all $u \in V$ and $i \in T$.

Also new variables are y_i^{uv}, defined for all $i \in T$ and $uv \in E$. Then one writes the linear program:

$$\text{Minimize } \frac{1}{2} \sum_{uv \in E} c(u,v) y^{uv}$$

Subject to :

$$x^u \in \Delta_k \qquad \forall u \in V$$

$$x^t = e^t \qquad \forall t \in T$$

$$y^{uv} = \sum_{i \in T} y_i^{uv} \qquad \forall uv \in E$$

$$y_i^{uv} \geq x_i^u - x_i^v \qquad \forall uv \in E\ ,\ \ i \in T$$

$$y_i^{uv} \geq x_i^v - x_i^u \qquad \forall uv \in E\ ,\ \ i \in T\ .$$

It is easy to see that this linear program optimally solves the simplex relaxation above, by noticing that an optimal solution to the linear program can be assumed to put $y_i^{uv} = |x_i^u - x_i^v|$ and $y^{uv} = \|x^u - x^v\|$. Thus, solving the simplex relaxation can be done in polynomial time. This is the first step of the approximation algorithm. Clearly, the value Z^* of this solution is a lower bound on the cost of the minimum multiway cut MWC.

The second step of the algorithm is a rounding procedure which transforms a feasible solution of the simplex relaxation into an integral feasible solution. The rounding procedure below differs slightly from the one given in [3], but can be proven to give exactly the same solution. This variant is easier to present, although if one wants to prove the approximation ratio then the only way we know of is by showing that indeed this variant gives the same solution as the more complicated algorithm given in [3].

Rounding

Set $B(i,\rho) = \{u \in V \mid x_i^u > 1 - \rho\}$, the set of nodes suitably "close" to terminal i in the simplex. Choose a permutation $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ to be either $\langle 1, 2, 3, \ldots, k-1, k \rangle$ or $\langle k-1, k-2, k-3, \ldots, 1, k \rangle$ with probability $1/2$ each. Independently, choose $\rho \in (0,1)$ uniformly at random. Then, process the terminals in the order $\sigma(1), \sigma(2), \sigma(3), \ldots, \sigma(k)$.

Algorithm 1 The rounding procedure

```
Let σ = ⟨1, ..., k−3, k−2, k−1, k⟩ or ⟨k−1, k−2, k−3, ..., 1, k⟩, each with prob. 1/2
Let ρ be a random real in (0, 1) /* See the paragraph below. */
for j = 1 to k − 1 do
    for all u such that x^u ∈ B(σ_j, ρ) \ ∪_{i:i<j} B(σ_i, ρ) do
        x̄^u := e^{σ_j}  /* assign node u to terminal σ_j */
    end for
end for
for all u such that x^u ∉ ∪_{i:i<k} B(σ_i, ρ) do
    x̄^u := e^k
end for
```

For each j from 1 to $k-1$, place the nodes that remain in $B(\sigma_j, \rho)$ at e^{σ_j}. Place whatever nodes remain at the end at e^k. The following code specifies the rounding procedure more formally. $\bar{x}$ denotes the rounded (integral) solution.

To derandomize and implement this algorithm in polynomial time, one tries both permutations σ and at most $k(n+1)$ values of ρ. Indeed, for any permutation σ, two different values of ρ, $\rho_1 < \rho_2$, produce combinatorially distinct solutions only if there is a terminal i and a node u such that $x_i^u \in (1-\rho_2, 1-\rho_1]$. Thus, there are at most $k(n+1)$ "interesting" values of ρ, which can be determined easily by sorting the nodes according to each coordinate separately. The resulting discrete sample space for (σ, ρ) has size at most $2k(n+1)$, so one can search it exhaustively.

The analysis of the algorithm, however, is based on the randomized algorithm above, as the proof shows that the expected total cost of edges whose endpoints are at different vertices of Δ_k in the rounded solution $\bar{x}$ is at most $1.5\,Z^*$. To get an $(1.5 - 1/k)Z^*$ upper bound, one must rename the terminals such that terminal k maximizes a certain quantity given by the simplex relaxation, or alternatively randomly pick a terminal as the last element of the permutation (the order of the first $k-1$ terminals does not matter as long as both the increasing and the decreasing permutations are tried by the rounding procedure). Exhaustive search of the sample space produces one integral solution whose cost does not exceed the average.

Applications

MULTIWAY CUT is used in Computer Vision, but unless one can solve the instance exactly, algorithms for the generalization METRIC LABELING are needed. MULTIWAY CUT has applications in parallel and distributed computing, as well as in chip design.

Open Problems

The improvements of [10, 4] are based on better rounding procedures and both compare the integral solution obtained to Z^*. This leads to the natural question: what is the supremum, over multiway cut instances G, of $Z^*(G)/\mathrm{MWC}(G)$. This supremum is called *integrality gap* or *integrality ratio*. For three terminals, [10] and [4] show that the integrality gap is exactly 12/11, while for general k, Freund and Karloff [7] give a lower bound of 8/7. The best-known upper bound is 1.3438, achieved by an approximation algorithm of [10].

Cross-References

- ▸ Multicut
- ▸ Sparsest Cut

Recommended Reading

1. Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows. Prentice Hall, Englewood Cliffs
2. Arora S, Karger D, Karpinski M (1999) Polynomial time approximation schemes for dense instances of NP-hard problems. J Comput Syst Sci 58(1):193–210, Preliminary version in STOC 1995
3. Calinescu G, Karloff HJ, Rabani Y (1998) An improved approximation algorithm for multiway cut. In: ACM symposium on theory of computing, pp 48–52. Journal version in J Comput Syst Sci 60:564–574 (2000)
4. Cheung K, Cunningham WH, Tang L (2006) Optimal 3-terminal cuts and linear programming. Math Prog 105:389–421, Preliminary version in IPCO 1999
5. Chopra S, Rao MR (1991) On the multiway cut polyhedron. Networks 21:51–89
6. Dahlhaus E, Johnson DS, Papadimitriou CH, Seymour PD, Yannakakis M (1994) The complexity of multiterminal cuts. SIAM J Comput 23:864–894, Preliminary version in STOC 1992, An extended abstract was first announced in 1983
7. Freund A, Karloff H (2000) A lower bound of 8/(7+1/k-1) on the integrality ratio of the Calinescu–Karloff–Rabani relaxation for multiway cut. Inf Process Lett 75:43–50
8. Frieze A, Kannan R (1996) The regularity lemma and approximation schemes for dense problems. In: Proceedings of 37th IEEE FOCS. IEEE Computer Society, Los Alamitos, pp 12–20
9. Garg N, Vazirani VV, Yannakakis M (2004) Multiway cuts in node weighted graphs. J Algorithms 50(1):49–61, Preliminary version in ICALP 1994
10. Karger DR, Klein P, Stein C, Thorup M, Young NE (2004) Rounding algorithms for a geometric embedding of minimum multiway cut. Math Oper Res 29(3):436–461, Preliminary version in STOC 1999
11. Naor JS, Zosin L (2001) A 2-approximation algorithm for the directed multiway cut problem. SIAM J Comput 31(2):477–492, Preliminary version in FOCS 1997
12. Vazirani VV (2001) Approximation algorithms. Springer, Berlin/Heidelberg/New York

M

Musite: Tool for Predicting Protein Phosphorylation Sites

Qiuming Yao[1], Jianjiong Gao[2], and Dong Xu[3]
[1]University of Missouri, Columbia, MO, USA
[2]Computational Biology Center, Memorial Sloan-Kettering Cancer Center, New York, NY, USA
[3]Bond Life Sciences Center, University of Missouri, Columbia, MO, USA

Keywords

Amino acid frequency; Disorder score; K nearest neighbors; Phosphorylation site prediction; Phosphoproteomics; Support vector machine; Substitution matrix

Years and Authors of Summarized Original Work

2010; Gao, Thelen, Dunker, Xu
2012; Yao, Gao, Bollinger, Thelen, Xu

Problem Definition

Protein phosphorylation plays an important role in various biological functions and cellular processes. Identifying potential phosphorylation sites in a protein often helps to reveal functional details at the molecular level and was always performed by in vivo or in vitro experiments. Since the last decade, bioinformatics has been contributing significantly in characterizing protein structures and functionalities solely from its primary information, which also sheds light on phosphorylation site prediction. As per our expectation, in silico prediction should not only provide an alternative way to identify protein phosphorylation sites at lower cost but also with much higher throughput (e.g., proteome-wide screening), so that biologists can quickly pinpoint the potential sites for further experiments from a long list of targets. Therefore, it is soon becoming valuable and imperative to build such a bioinformatics tool or framework that can predict general and kinase-specific phosphorylation sites in proteins.

In definition, protein phosphorylation prediction is a computational approach to determine whether a certain amino acid in a protein sequence can be potentially phosphorylated or not. More specifically, given a protein (or peptide) sequence $P = [a_i]$ $i = 1, \ldots n$ (where n is the sequence length, with amino acid a_i at ith position), the prediction algorithm is to tell if each of a_i (especially when a_i is serine, threonine, or tyrosine) can be phosphorylated in P or not. In a kinase-specific format, this question is asked with a proposed kinase name or kinase family. Moreover, this question can also be asked in a species-specific or condition-specific format.

Key Results

Machine Learning Approach

The algorithm we designed to fulfill this prediction task is able to resolve the association between phosphorylation and sequence information from the experimentally identified phosphorylation sites. Thus, it is formulated as a machine learning approach rather than an ab initio method. The collected experimental data need to be split into training and testing set to generate, tune, and validate our machine learning models. These machine learning models are then capable of predicting general or kinase-specific phosphorylation site for proteins with unknown sites. The general or specific prediction models are very dependent on the correspondent data sets in training. For example, kinase-specific prediction requires kinase-specific training data sets. After different preprocessing steps applied on data sets for general or specific purposes, the prediction models are generated from the same machine learning method (i.e., support vector machine [1–3] in our framework).

Technically speaking, per site prediction can be modeled as a binary classification problem, where the class label Y is either $+1$ for identified phosphorylation site or -1 for unidentified site, with X as its feature vector. A machine learning model can be considered as a map function from feature space X to the class label, i.e., $M : g(X) \rightarrow Y$, obtained from the training data set$\{(X_1, Y_1), (X_2, Y_2) \ldots, (X_m, Y_m)\}$. The prediction for the unknown X^* is simply calculated through $Y^* = g(X^*)$.

In our case, X_i is a feature vector from the protein sequence, extracted from a flanking peptide surrounding i-th amino acid a_i. The flanking sequence is often centralized by a_i and is a substring of the original protein sequence, denoted as $p(a_i) = [a_{i-w}, \ldots a_i, \ldots, a_{i+w}]$, where w is called the window size.

Support vector machine (SVM) then generates our prediction model M by maximizing the margin of the classification boundary.

Features

K nearest neighbor (KNN) scores, disorder scores, and amino acid information are used as the features in our SVM-based machine learning approach.

For amino acid a_i, **k** nearest neighbors are defined as the top k most similar peptides (within smallest distances) to the target peptide $p(a_i)$ in the training data set. Besides size k,

the neighborhood can be also defined as a certain percentage of the whole training data set (e.g., 1 % of the total population). KNN score is then the ratio between the numbers of positive and negative sites within this predefined neighborhood.

Notice that the peptide similarity needs to be defined and normalized. In a more clear illustration, the two flanking sequences centralized by amino acid a_i and a_j are represented as $p(a_i) = [a_{i-w}, a_{i-w+1}, \ldots, a_i, \ldots, a_{i+w-1}, a_{i+w}]$ and $p(a_j) = [a_{j-w}, a_{j-w+1}, \ldots, a_j, \ldots, a_{j+w-1}, a_{j+w}]$, respectively.

The distance $D(p(a_i), p(a_j))$ between peptides $p(a_i)$ and $p(a_j)$ is calculated by

$$D(p(a_i), p(a_j)) = 1 - \frac{\sum_{k=-w}^{w} S(a_{i+k}, a_{j+k})}{2w+1}$$

where w is the window size, and the function S(.) is to calculate the amino acid similarity between a_i and a_j based on the normalized amino acid substitution matrix Q. More specifically,

$$S(a_i, a_j) = \frac{Q(a_i, a_j) - \min(Q)}{\max(Q) - \min(Q)}$$

where a_i and a_j are two amino acids, Q is the substitution matrix, and max(Q) and min(Q) represent the maximal and minimal values in the matrix Q. By default, BLOSUM62 is used as the most general substitution matrix. In fact, Q can also be directly calculated from the training data set and then KNN score is very specific to the training samples.

Disorder score per amino acid site reflects the stability of the local structure and is calculated by VSL2B [4]. By considering the disorder property as a more neighborhood-dependent and continuous feature, we correct (smooth) the disorder score at the position a_i using the mean value across the flanking peptide $p(a_i)$, i.e.,

$$\begin{aligned} \text{Disorder}(a_i) &= \text{average}(p(a_i)) \\ &= \frac{1}{2w+1} \sum_{k=-w}^{w} \text{disorder}(a_{i+k}) \end{aligned}$$

Amino acid information for flanking sequence $p(a_i)$ can refer to both composition and position information. At one extreme, amino acid frequency reflects composition information but no position information. The amino acid preference in phosphorylated peptides [5] can be revealed by this frequency feature. The size of this frequency vector is 20, which stores the normalized counts for each amino acid type within the range of the flanking peptide $p(a_i)$. On the other extreme, amino acid binary vector can provide position-specific information by bookkeeping a 0-1 vector for each amino acid at each position. The length of amino acid binary vector is 20*w, much longer than the frequency, which may potentially cause over-fitting in machine learning when the sample size is small. Therefore, selecting the right way to encode and represent the amino acid information is a trade-off between the losslessness of the positional information and the length of the feature vector.

Bootstrap and Aggregation

Since the under-identified phosphorylation sites (negative data) are always overwhelming the identified ones (positive data) in the training data set, we resolved this problem with bootstrap procedures to avoid the potential bias in the final classifier due to this unbalancing situation. The bootstrap step is a randomized resampling to get a balanced training data each time, which is repeated many times in order to explore the whole sample space thoroughly. So we will get many models based on the actual number of bootstrap steps, such as $M_1, \ldots, M_k$, and k could be up to thousands. Then, we do the final classification based on the voting or mean value from these many models, i.e., $G : G[g_1(X), \ldots g_k(X)]$, where G is called the aggregation step. The aggregated model is thus unbiased despite the imbalance of the labels in the training data.

Cross Validation

With the trained model, the testing result is often displayed as a trade-off between specificity and sensitivity, e.g., by a receiver-operating characteristic (ROC) curve. Specificity and sensitivity are defined as follows:

$$\text{specificity} = \frac{TN}{TN + FP}$$

$$\text{sensitivity} = \frac{TP}{TP + FN}$$

where TN represents true negative, FP false positive, TP true positive, and FN false negative.

Cross validation is a way to measure if the power of the phosphorylation prediction model trained from the known data can be well extended to the unknown. Usually, the cross validation can be performed with leave-one-out strategy (for small data set, i.e., kinase-specific data set) or from non-overlapped testing and training sets with x folds settings (for general phosphorylation site).

Musite as a Toolkit

Musite is an open-source software toolkit designed for large-scale phosphorylation prediction for both general and kinase-specific cases [6, 7]. The framework is quite flexible, so that user can take advantage of different preprocessing steps for specifying training or testing data, as well as picking different features and tuning parameters. By default, Musite provides general phosphorylation prediction models, several popular kinase-specific models, and multiple species-specific predictions (e.g., a plant-specific tool was build using our in-house plant protein phosphorylation database P^3DB [8–10]). Moreover, trained with users' specific data sets, Musite is also capable of generating customized models to do precise prediction particularly on their own research focus.

Applications

This tool or framework can be used as a quick filter on a long list of candidate proteins for experimental biologists to narrow down the phosphorylation sites to perform biochemical assay. It can also help to evaluate or compare the experimental observations in discovery studies. On the other hand, the computational experts can use this tool to do comparative studies, by fast and cheap computer screening across multiple proteomes. This tool can be easily and freely incorporated into any translational bioinformatics pipeline to characterize or annotate protein functionality within large scale proteomics studies.

Open Problems

1. The current version is an alignment free method. Is it possible or necessary to consider alignment, i.e., allowing indels for peptides similarity calculation?
2. The current features are more or less local. Are there any feasible features representing long distance association with phosphorylation site?
3. Can we extract any interesting biological rules from the machine learning models for general or specific phosphorylation events?

URLs to Code

The source code can be downloaded from SourceForge.
http://musite.sourceforge.net/
The online prediction Web services are available at:
http://musite.net/
http://p3db.org/prediction.php

Cross-References

▸ Support Vector Machines

Recommended Reading

1. Schölkopf B, Burges CJC, Smola AJ (1999) Advances in kernel methods support vector learning. MIT, Cambridge
2. Wang L (2005) Support vector machines: theory and applications. In: Studies in fuzziness and soft computing. Springer, Berlin
3. Vapnik VN (2000) The nature of statistical learning theory, 2nd edn. Springer, New York
4. Obradovic Z, Peng K, Vucetic S, Radivojac P, Dunker AK (2005) Exploiting heterogeneous sequence properties improves prediction of protein disorder. Proteins 61(Suppl 7):176–182

5. Iakoucheva LM, Radivojac P, Brown CJ, O'Connor TR, Sikes JG, Obradovic Z, Dunker AK (2004) The importance of intrinsic disorder for protein phosphorylation. Nucleic Acids Res 32(3):1037–1049
6. Yao Q, Gao J, Bollinger C, Thelen JJ, Xu D (2012) Predicting and analyzing protein phosphorylation sites in plants using musite. Front Plant Science 3:186
7. Gao J, Thelen JJ, Dunker AK, Xu D (2010) Musite, a tool for global prediction of general and kinase-specific phosphorylation sites. Mol Cell Proteomics (MCP) 9(12):2586–2600
8. Gao J, Agrawal GK, Thelen JJ, Xu D (2009) P3DB: a plant protein phosphorylation database. Nucleic Acids Research 37(Database issue): D960–D962
9. Yao Q, Bollinger C, Gao J, Xu D, Thelen JJ (2012) P3DB: an integrated database for plant protein phosphorylation. Front Plant Sci 3:206
10. Yao Q, Ge H, Wu S, Zhang N, Chen W, Xu C, Gao J, Thelen JJ, Xu D (2014) P3DB 3.0: from plant phosphorylation sites to protein networks. Nucleic Acids Res 42(Database issue): D1206–D1213

N

Nash Equilibria and Dominant Strategies in Routing

Weizhao Wang[1], Xiang-Yang Li[2], and Xiaowen Chu[3]
[1]Google Inc., Irvine, CA, USA
[2]Department of Computer Science, Illinois Institue of Technology, Chicago, IL, USA
[3]Department of Computer Science, Hong Kong Baptist University, Hong Kong, China

Keywords

BB; Nash; Strategyproof; Truthful

Years and Authors of Summarized Original Work

2005; Wang, Li, Chu

Problem Definition

This problem is concerned with the multicast routing and cost sharing in a selfish network composed of relay terminals and receivers. This problem is motivated by the recent observation that the selfish behavior of the network could largely degraded existing system performance, even dysfunction. The work of Wang, Li and Chu [7] first presented some negative results of the strategyproof mechanism in multicast routing and sharing, and then proposed a new solution based on Nash Equilibrium that could greatly improve the performance.

Wang, Li and Chu modeled a network by a link weighted graph $G = (V, E, \mathbf{c})$, where V is the set of all nodes and $\mathbf{c}$ is the cost vector of the set E of links. For a multicast session, let Q denote the set of all receivers. In game theoretical networking literatures, usually there are two models for the multicast cost/payment sharing.

Axiom Model (AM) All receivers must receive the service, or equivalently, each receiver has an infinity valuation [3]. In this model, a sharing method ξ computes how much each receiver should pay when the receiver set is R and cost vector is $\mathbf{c}$.

Valuation Model (VM) There is a set $Q = \{q_1, q_2, \cdots, q_r\}$ of r possible receivers. Each receiver $q_i \in Q$ has a valuation $\boldsymbol{\eta}_i$ for receiving the service. Let $\boldsymbol{\eta} = (\eta_1, \eta_2, \ldots, \eta_r)$ be the valuation vector and $\boldsymbol{\eta}_R$ be the valuation vector of a set $R \subseteq Q$ of receivers. In this model, they are interested in a sharing mechanism $\mathcal{S}$ consisting of a *selection scheme* $\sigma(\boldsymbol{\eta}, \mathbf{c})$ and a *sharing method* $\xi(\boldsymbol{\eta}, \mathbf{c})$. $\sigma_i(\boldsymbol{\eta}, \mathbf{c})$ denotes whether receiver i receives the service or not, and $\xi_i(\boldsymbol{\eta}, \mathbf{c})$ computes how much the receiver q_i should pay for the multicast service. Let $\mathbb{P}(\boldsymbol{\eta}, \mathbf{c})$ be the total payment for providing the service to the receiver set.

In the valuation model, a receiver who is willing to receive the service is not guaranteed to receive the service. For notational simplicity, $\sigma(\boldsymbol{\eta}, \mathbf{c})$ is used to denote the set of actual receivers. Under the Valuation Model, a *fair*

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

Algorithm 1 The multicast system $\Psi^{\mathsf{DM}} = (\mathcal{M}^{\mathsf{DM}}, \mathcal{S}^{\mathsf{DM}})$ based on multicast tree LCPT

Compute path $\mathsf{LCP}(s, q_j, \mathbf{d})$ and set $\phi_j = \frac{\omega(\mathrm{B}_{mm}(s,q_j,\mathbf{d}),\mathbf{d})}{r}$ for every $q_j \in Q$.
Set $\mathcal{O}_i^{\mathsf{DM}}(\eta, \mathbf{d}) = 0$ and $\mathcal{P}_i^{\mathsf{DM}}(\eta, \mathbf{d}) = 0$ for each link $e_i \notin \mathsf{LCP}(s, q_j, \mathbf{d})$.
for each receiver q_j **do**
  if $\eta_j \geq \phi_j$ **then**
    Receiver q_j is granted the service and charged $\xi_j^{\mathsf{DM}}(\eta, \mathbf{d})$, set $R = R \cup q_j$.
  else
    Receiver q_j is not granted the service and is charged 0.
  end if
end for
Set $\mathcal{O}_i^{\mathsf{DM}}(\eta, \mathbf{d}) = 1$ and $\mathcal{P}_i^{\mathsf{DM}}(\eta, \mathbf{d}) = \mathcal{P}_i^{\mathsf{LCPT}}(\eta_R^{=\infty}, \mathbf{d})$ for each link $e_i \in \mathsf{LCPT}(R, \mathbf{d})$.

Algorithm 2 FPA Mechanism $\mathcal{M}^{\mathsf{AUC}}$

Each terminal bids a price b_i.
Every link sends a unit size dummy packet with property $\rho = \tau \cdot (n \cdot b_u - \sum_{e_i \in G} b_i)$ and receives payment $f_i(s, q_1, \mathbf{b}) = \tau \cdot \left[b_u \cdot (n \cdot b_u - \sum_{e_j \in G - e_i} b_j) - \frac{h_i^2}{2} \right]$. Here, b_u is the maximum cost any link can declare.
Compute the unique path $\mathsf{LCP}(s, q_1, \mathbf{b}')$ by applying certain fixed tie-breaking rule consistently.
Each terminal bids again for a price b_i'.
for each link e_i **do**
It is select to relay the packet and receives payment b_i' if and only if e_i is on path $\mathsf{LCP}(s, q_1, \mathbf{b}')$.
end for

sharing according to the following criteria is studied.

- **Budget Balance**: For the receiver set $R = \sigma(\boldsymbol{\eta}, \mathbf{c})$, $\mathbb{P}(\boldsymbol{\eta}, \mathbf{c}) = \sum_{q_i \in Q} \xi_i(\boldsymbol{\eta}, \mathbf{c})$. If $\alpha \cdot \mathbb{P}(\boldsymbol{\eta}, \mathbf{c}) \leq \sum_{i \in R} \xi_i(\boldsymbol{\eta}, \mathbf{c}) \leq \mathbb{P}(\boldsymbol{\eta}, \mathbf{c})$, for some given parameter $0 < \alpha \leq 1$, then $\mathcal{S} = (\sigma, \xi)$ is called α-budget-balance. If budget balance is not achievable, then a sharing scheme $\mathcal{S}$ may need to be α-budget-balance instead of budget balance.
- **No Positive Transfer** (NPT): Any receiver q_i's sharing should not be negative.
- **Free Leaving**: (FR) The potential receivers who do not receive the service should not pay anything.
- **Consumer Sovereignty** (CS): For any receiver q_i, if $\boldsymbol{\eta}_i$ is sufficiently large, then q_i is guaranteed to be an actual receiver.
- **Group-Strategyproof** (GS): Assume that $\boldsymbol{\eta}$ is the valuation vector and $\boldsymbol{\eta}' \neq \boldsymbol{\eta}$. If $\xi_i(\boldsymbol{\eta}', \mathbf{c}) \geq \xi_i(\boldsymbol{\eta}, \mathbf{c})$ for each $q_i \in \boldsymbol{\eta}$, then $\xi_i(\boldsymbol{\eta}', \mathbf{c}) = \xi_i(\boldsymbol{\eta}, \mathbf{c})$.

Notations

The path with the lowest cost between two odes s and t is denoted as $\mathsf{LCP}(s, t, \mathbf{c})$, and its cost is dented as $|\mathsf{LCP}(s, t, \mathbf{c})|$. Given a simple path P in the graph G with cost vector $\mathbf{c}$, the sum of the cost of links on path P is denoted as $|\mathsf{P}(\mathbf{c})|$. For a simple path $P = v_i \rightsquigarrow v_j$, if $\mathsf{LCP}(s, t, \mathbf{c}) \bigcap P = \{v_i, v_j\}$, then P is called a *bridge* over $\mathsf{LCP}(s, t, \mathbf{c})$. This bridge P *covers* link e_k if $e_k \in \mathsf{LCP}(v_i, v_j, \mathbf{c})$. Given a link $e_i \in \mathsf{LCP}(s, t, \mathbf{c})$, the path with the minimum cost that covers e_i is denoted as $\mathrm{B}_{\min}(e_i, \mathbf{c})$. The bridge $\mathrm{B}_{mm}(s, t, \mathbf{c}) = \max_{e_i \in \mathsf{LCP}(s,t,\mathbf{c})} \mathrm{B}_{\min}(e_i, \mathbf{c})$ is the *max-min cover* of the path $\mathsf{LCP}(s, t, \mathbf{c})$.

A bridge set $\mathcal{B}$ is a *bridge cover* for $\mathsf{LCP}(s, t, \mathbf{c})$, if for every link $e_i \in \mathsf{LCP}(s, t, \mathbf{c})$, there exists a bridge $\mathrm{B} \in \mathcal{B}$ such that $e_i \in \mathsf{LCP}(v_{s(\mathrm{B})}, v_{t(\mathrm{B})}, \mathbf{c})$. The *weight* of a bridge cover $\mathcal{B}(s, t, \mathbf{c})$ is defined as $|\mathcal{B}(s, t, \mathbf{c})| = \sum_{B \in \mathcal{B}(s,t,\mathbf{c})} \sum_{e_i \in B} c_i$. A bridge cover is a *least bridge cover* (LB), denoted by $\mathbb{LB}(s, t, \mathbf{c})$, if it has the smallest weight among all bridge covers that cover $\mathsf{LCP}(s, t, \mathbf{c})$.

Key Results

Theorem 1 *If $\Psi = (\mathcal{M}, \mathcal{S})$ is an α-stable multicast system, then $\alpha \leq 1/n$.*

Theorem 2 *Multicast system Ψ^{DM} is $1/(r \cdot n)$-stable, where r is the number of receivers.*

Theorem 1 gives an upper bound for α for any α-stable unicast system Ψ. It is not difficult to observe that even the receivers are cooperative, Theorem 1 still holds. Theorem 2 showed that

Algorithm 3 FPA based unicast system

1: Execute Line 1 – 3 in Algorithm 2.
2: Compute $\mathbb{LB}(s, q_1, \mathbf{b})$, and set $\phi = \frac{\mathbb{LB}(s,q_1,\mathbf{b})|}{2}$.
3: If $\phi \leq \eta_1$ then set $\sigma_1^{\mathsf{AU}}(\eta_1, \widetilde{\mathbf{b}}) = 1$ and $\xi_1^{\mathsf{AU}}(\eta_1, \widetilde{\mathbf{b}}) = \phi$. Every relay link on LCP is selected and receives an extra payment b_i'.
4: For each link $e_i \notin \mathrm{LCP}(s, q_1, \mathbf{b}')$, it receives a payment $\mathcal{P}_i^{\mathsf{AU}}(\eta_1, \widetilde{\mathbf{b}}) - \gamma \cdot (b_i' - b_i)^2$.

there exists a multicast system is $1/(r \cdot n)$-stable. When $r = 1$, the problem become traditional unicast system and the bound is tight. When relaxing the dominant strategy to the Nash Equilibria requirement, a First Price Auction (FPA) mechanism is proposed by Wang et al. under the Axiom Model that has many nice properties.

Theorem 3 *There exists NE for FPA mechanism $\mathcal{M}^{\mathsf{AUC}}$ and for any NE, (a) each link bids his true cost as the first bid b_i, (b) the actual shortest path is always selected, (c) the total cost for different NE differs at most 2 times.*

Based on the FPA Mechanism Ψ^{AUC}, Wang, Li and Chu design a unicast system as follows.

Theorem 4 *The FPA based unicast system not only has Nash Equilibria, but also is $\frac{1}{2}$-NE-stable with ϵ additive, for any given ϵ.*

By treating each receiver as a separate receiver and applying the similar process as in the unicast system, Wang, Li and Chu extended the unicast system to a multicast system.

Theorem 5 *The FPA based multicast system not only has Nash Equilibria, but also is $1/(2 \cdot r)$-NE-stable with ϵ additive, for any given ϵ.*

Applications

More and more research effort has been done to study the non-cooperative games recently. Among these various forms of games, the unicast/multicast routing game [2, 5, 6] and multicast cost sharing game [1, 3, 4] have received a considerable amount of attentions over the past few year due to its application in the Internet. However, both unicast/multicast routing game and multicast cost sharing game are one folded: the unicast/multicast routing game does not take the receivers into account while the multicast cost sharing game does not treat the links as non-cooperative. In this paper, they study the scenario, which was called *multicast system*, in which both the links and the receivers could be non-cooperative. Solving this problem paving a way for the real world commercial multicast and unicast application. A few examples are, but not limited to, the multicast of the video content in wireless mesh network and commercial WiFi system; the multicast routing in the core Internet.

Open Problems

A number of problems related to the work of Wang, Li and Chu [7] remain open. The first and foremost, the upper bound and lower bound on α still have a gap of r if the multicast system is α-stable; and a gap of $2r$ if the multicast system is α-Nash stable.

The second, Wang, Li and Chu only showed the existence of the Nash Equilibrium under their systems. They have not characterized the convergence of the Nash Equilibrium and the strategies of the user, which are not only interesting but also important problems.

Cross-References

▸ Non-approximability of Bimatrix Nash Equilibria

Recommended Reading

1. Feigenbaum J, Papadimitriou CH, Shenker S (2001) Sharing the cost of multicast transmissions. J Comput Syst Sci 63:21–41
2. Kao M-Y, Li X-Y, Wang W (2005) Towards truthful mechanisms for binary demand games: a general framework. In: ACM EC, Vancouver, pp 213–222
3. Herzog S, Shenker S, Estrin D (1997) Sharing the "cost" of multicast trees: an axiomatic analysis. IEEE/ACM Trans Netw 5:847–860

4. Moulin H, Shenker S (2001) Strategyproof sharing of submodular costs: budget balance versus efficiency. Econ Theory 18:511–533
5. Wang W, Li X-Y, Sun Z, Wang Y (2005) Design multicast protocols for non-cooperative networks. In: Proceedings of the 24th IEEE INFOCOM, Miami, vol 3, pp 1596–1607
6. Wang W, Li X-Y, Wang Y (2004) Truthful multicast in selfish wireless networks. In: Proceedings of the 10th ACM MOBICOM, Philadelphia, pp 245–259
7. Wang W, Li X-Y, Chu X (2005) Nash equilibria, dominant strategies in routing. In: Workshop for Internet and network economics (WINE). Lecture notes in computer science, vol 3828. Springer, Hong Kong, pp 979–988

Nearest Neighbor Interchange and Related Distances

Bhaskar DasGupta[1], Xin He[2], Ming Li[3], John Tromp[4], and Louxin Zhang[5]
[1]Department of Computer Science, University of Illinois, Chicago, IL, USA
[2]Department of Computer Science and Engineering, The State University of New York, Buffalo, NY, USA
[3]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada
[4]CWI, Amsterdam, The Netherlands
[5]Department of Mathematics, National University of Singapore, Singapore, Singapore

Keywords

Comparison of phylogenies; Network models of evolution

Years and Authors of Summarized Original Work

1997; DasGupta, He, Jiang, Li, Tromp, Zhang

Problem Definition

In this entry, the authors state results on some transformation-based distances for *evolutionary trees*. Several distance models for evolutionary trees have been proposed in the literature. Among them, the best known is perhaps the *nearest neighbor interchange* (nni) distance introduced independently in [10] and [9]. The authors will focus on the nni distance and a closely related distance called the *subtree-transfer* distance originally introduced in [5, 6]. Several papers that involved DasGupta, He, Jiang, Li, Tromp, and Zhang essentially showed the following results:

- A correspondence between the nni distance and the linear-cost subtree-transfer distance on unweighted trees.
- Computing the nni distance is NP-hard, but admits a fixed-parameter tractability and a logarithmic ratio approximation algorithms.
- A 2-approximation algorithm for the linear-cost subtree-transfer distance on weighted evolutionary trees.

The authors first define the nni and linear-cost subtree-transfer distances for unweighted trees. Then the authors extend the nni and linear-cost subtree-transfer distances to weighted trees. For the purpose of this entry, an evolutionary tree (also called *phylogeny*) is an *unordered* tree, has uniquely labeled leaves and unlabeled interior nodes, can be *unrooted* or *rooted*, can be *unweighted* or *weighted*, and has all internal nodes of degree 3.

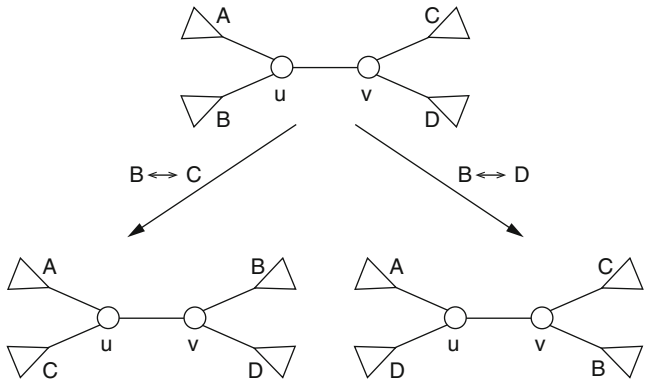
Unweighted Trees

An nni operation swaps two subtrees that are separated by an internal edge (u, v), as shown in Fig. 1.

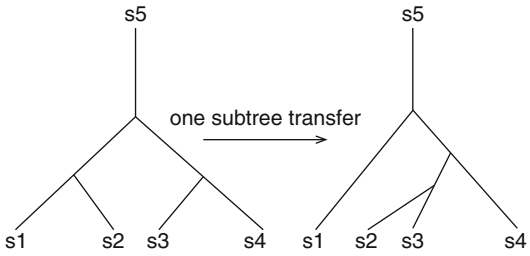
The nni operation is said to *operate* on this internal edge. The nni distance, $D_{\text{nni}}(T_1, T_2)$, between two trees T_1 and T_2 is defined as the *minimum* number of nni operations required to transform one tree into the other.

An nni operation can also be viewed as moving a subtree past a neighboring internal node. A more general operation is to transfer a subtree from one place to another arbitrary place. Figure 2 shows such a *subtree-transfer* operation.

The subtree-transfer distance between two trees T_1 and T_2 is the minimum number of



Nearest Neighbor Interchange and Related Distances, Fig. 1 The two possible nni operations on an internal edge (u, v): exchange $B \leftrightarrow C$ or $B \leftrightarrow D$



Nearest Neighbor Interchange and Related Distances, Fig. 2 An example of subtree-transfer

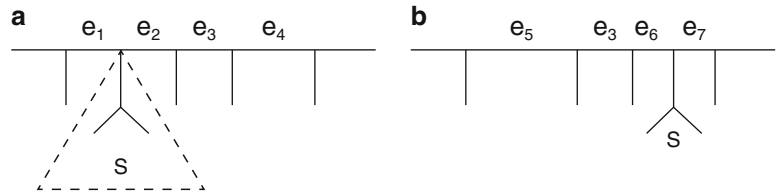
subtrees one needs to move to transform T_1 into T_2 [5–7]. It is sometimes appropriate in practice to discriminate among subtree-transfer operations as they occur with different frequencies. In this case, one can charge each subtree-transfer operation a cost equal to the distance (the number of nodes passed) that the subtree has moved in the current tree. The *linear-cost* subtree-transfer distance, $D_{\text{lcst}}(T_1, T_2)$, between two trees T_1 and T_2 is then the minimum total cost required to transform T_1 into T_2 by subtree-transfer operations [1, 3].

Weighted Trees

Both the linear-cost subtree-transfer and nni models can be naturally extended to weighted trees. The extension for nni is straightforward: an nni operation is simply charged a cost equal to the weight of the edge it operates on. For feasibility of weighted nni transformation between two given weighted trees T_1 and T_2, one also requires that the following conditions are satisfied: (1) for each leaf label a, the weight of the edge in T_1 incident on a is the same as the weight of the edge in T_2 incident on a and (2) the multisets of weights of internal edges of T_1 and T_2 are the same (Fig. 3).

In the case of linear-cost subtree-transfer, although the idea is immediate, i.e., a moving subtree should be charged for the weighted distance it travels, the formal definition needs some care and is given below. Consider (unrooted) trees in which each edge e has a weight $w(e) \geq 0$. To ensure feasibility of transforming a tree into another, one requires the total weight of all edges to equal one. A subtree-transfer is now defined as follows. Select a subtree S of T at a given node u and select an edge $e \notin S$. Split the edge e into two edges e_1 and e_2 with weights $w(e_1)$ and $w(e_2)$ $(w(e_1), w(e_2) \geq 0, w(e_1)+w(e_2) = w(e))$, and move S to the common end point of e_1 and e_2. Finally, merge the two remaining edges e' and e'' adjacent to u into one edge with weight $w(e') + w(e'')$. The cost of this subtree-transfer is the total weight of all the edges over which S is moved. Figure 3 gives an example. The edge-weights of the given tree are normalized so that their total sum is 1. The subtree S is transferred to split the edge e_4 to e_6 and e_7 such that $w(e_6)$, $w(e_7) \geq 0$ and $w(e_6) + w(e_7) = w(e_4)$; finally, the two edges e_1 and e_2 are merged to e_5 such that $w(e_5) = w(e_1) + w(e_2)$. The cost of transferring S is $w(e_2) + w(e_3) + w(e_6)$.

Note that for weighted trees, the linear-cost subtree-transfer model is more general than the nni model in the sense that one can slide a subtree along an edge with subtree-transfers. Such an operation is not realizable with nni moves.



Nearest Neighbor Interchange and Related Distances, Fig. 3 Subtree-transfer on weighted phylogenies. Tree (**b**) is obtained from tree (**a**) with one subtree-transfer

Key Results

Let T_1 and T_2 be the two trees, each with n nodes, that are being used in the distance computation.

Theorem 1 ([1,2,4]) *Assume that T_1 and T_2 are unweighted. Then, the following results hold:*

- $D_{\text{nni}}(T_1, T_2) = D_{\text{lcst}}(T_1, T_2)$.
- *Computing $D_{\text{nni}}(T_1, T_2)$ is NP-complete.*
- *Suppose that $D_{\text{nni}}(T_1, T_2) \leq d$. Then, an optimal sequence of* nni *operations transforming T_1 into T_2 can be computed in $O(n^2 \log n + n \cdot 2^{23d/2})$ time.*
- *$D_{\text{nni}}(T_1, T_2)$ can be approximated to within a factor of $\log n + O(1)$ in polynomial time.*

Theorem 2 ([1–4]) *Assume that T_1 and T_2 are weighted. Then, the following results hold:*

- *$D_{\text{nni}}(T_1, T_2)$ can be approximated to within a factor of $6 + 6\log n$ in $O(n^2 \log n)$ time.*
- *Assume that T_1 and T_2 are allowed to have leaves that are not necessarily uniquely labeled. Then, computing $D_{\text{lcst}}(T_1, T_2)$ is NP-hard.*
- *$D_{\text{lcst}}(T_1, T_2)$ can be approximated to within a factor of 2 in $O(n^2 \log n)$ time.*

Applications

The results reported here are on transformation-based distances for evolutionary trees. Such a tree can be *rooted* if the evolutionary origin is known and can be *weighted* if the evolutionary length on each edge is known. Reconstructing the correct evolutionary tree for a set of species is one of the fundamental yet difficult problems in evolutionary genetics. Over the past few decades, many approaches for reconstructing evolutionary trees have been developed, including (not exhaustively) parsimony, compatibility, distance, and maximum likelihood approaches. The outcomes of these methods usually depend on the data and the amount of computational resources applied. As a result, in practice they often lead to different trees on the same set of species [8]. It is thus of interest to compare evolutionary trees produced by different methods or by the same method on different data.

Another motivation for investigating the linear-cost subtree-transfer distance comes from the following motivation. When *recombination* of DNA sequences occurs in an evolution, two sequences meet and generate a new sequence, consisting of genetic material taken left of the recombination point from the first sequence and right of the point from the second sequence [5, 6]. From a phylogenetic viewpoint, before the recombination, the ancestral material on the present sequence was located on two sequences, one having all the material to the left of the recombination point and another having all the material to the right of the breaking point. As a result, the evolutionary history can no longer be described by a single tree. The recombination event partitions the sequences into two neighboring regions. The history for the left and the right regions could be described by separate evolutionary trees. The recombination makes the two evolutionary trees describing neighboring regions differ. However, two neighbor trees cannot be arbitrarily different,

one must be obtainable from the other by a *subtree-transfer operation*. When more than one recombination occurs, one can describe an evolutionary history using a list of evolutionary trees, each corresponds to some region of the sequences and each can be obtained by several subtree-transfer operations from its predecessor [6]. The computation of a linear-cost subtree-transfer distance is useful in reconstructing such a list of trees based on parsimony [5, 6].

Open Problems

1. Is there a constant ratio approximation algorithm for the nni distance on unweighted evolutionary trees or is the $O(\log n)$-approximation the best possible?
2. Is the linear-cost subtree-transfer distance NP-hard to compute on weighted evolutionary trees if leaf labels are not allowed to be nonunique?
3. Can one improve the approximation ratio for linear-cost subtree-transfer distance on weighted evolutionary trees?

Cross-References

- ▸ Algorithms for Combining Rooted Triplets into a Galled Phylogenetic Network
- ▸ Maximum Agreement Subtree (of 2 Binary Trees)
- ▸ Maximum Agreement Subtree (of 3 or More Trees)
- ▸ Phylogenetic Tree Construction from a Distance Matrix

Recommended Reading

1. DasGupta B, He X, Jiang T, Li M, Tromp J, Zhang L (1997) On distances between phylogenetic trees. In: 8th annual ACM-SIAM symposium on discrete algorithms, New Orleans, pp 427–436
2. DasGupta B, He X, Jiang T, Li M, Tromp J, Wang L, Zhang L (1998) Computing distances between evolutionary trees. In: Du DZ, Pardalos PM (eds) Handbook of combinatorial optimization, vol 2. Kluwer Academic, Norwell, pp 35–76
3. DasGupta B, He X, Jiang T, Li M, Tromp J (1999) On the linear-cost subtree-transfer distance. Algorithmica 25(2):176–195
4. DasGupta B, He X, Jiang T, Li M, Tromp J, Zhang L (2000) On computing the nearest neighbor interchange distance. In: Du DZ, Pardalos PM, Wang J (eds) Proceedings of the DIMACS workshop on discrete problems with medical applications. DIMACS series in discrete mathematics and theoretical computer science, vol 55. American Mathematical Society, Providence, Rhode Island, USA, pp 125–143
5. Hein J (1990) Reconstructing evolution of sequences subject to recombination using parsimony. Math Biosci 98:185–200
6. Hein J (1993) A heuristic method to reconstruct the history of sequences subject to recombination. J Mol Evol 36:396–405
7. Hein J, Jiang T, Wang L, Zhang K (1996) On the complexity of comparing evolutionary trees. Discret Appl Math 71:153–169
8. Kuhner M, Felsenstein J (1994) A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. Mol Biol Evol 11(3):459–468
9. Moore GW, Goodman M, Barnabas J (1973) An iterative approach from the standpoint of the additive hypothesis to the dendrogram problem posed by molecular data sets. J Theor Biol 38:423–457
10. Robinson DF (1971) Comparison of labeled trees with valency three. J Comb Theory Ser B 11:105–119

N

Negative Cycles in Weighted Digraphs

Christos Zaroliagis
Department of Computer Engineering and Informatics, University of Patras, Patras, Greece

Years and Authors of Summarized Original Work

1994; Kavvadias, Pantziou, Spirakis, Zaroliagis

Problem Definition

Let $G = (V, E)$ be an n-vertex, m-edge directed graph (digraph), whose edges are associated with a real-valued cost function $wt : E \to \mathbb{R}$. The cost, $wt(P)$, of a path P in G is the sum of the costs of the edges of P. A simple path C whose

starting and ending vertices coincide is called a cycle. If $wt(C) < 0$, then C is called a *negative cycle*. The goal of the negative cycle problem is to detect whether there is such a cycle in a given digraph G with real-valued edge costs, and if indeed exists to output the cycle.

The negative cycle problem is closely related to the shortest path problem. In the latter, a minimum cost path between two vertices s and t is sought. It is easy to see that an s-t shortest path exists if and only if no s-t path in G contains a negative cycle [1, 13]. It is also well-known that shortest paths from a given vertex s to all other vertices form a tree called *shortest path tree* [1, 13].

Key Results

For the case of general digraphs, the best algorithm to solve the negative cycle problem (or to compute the shortest path tree, if such a cycle does not exist) is the classical Bellman–Ford algorithm that takes $O(nm)$ time (see e.g., [1]). Alternative methods with the same time complexity are given in [4, 7, 12, 13]. Moreover, in [11, Chap. 7] an extension of the Bellman–Ford algorithm is described which, in addition to detecting and reporting the existing negative cycles (if any), builds a shortest path tree rooted a some vertex s reaching those vertices u whose shortest s-u path does not contain a negative cycle. If edge costs are integers larger than $-L$ ($L \geq 2$), then a better algorithm was given in [6] that runs in $O(m\sqrt{n}\log L)$ time, and it is based on bit scaling.

A simple deterministic algorithm that runs in $O(n^2 \log n)$ expected time with high probability is given in [10] for a large class of input distributions, where the edge costs are chosen randomly according to the endpoint-independent model (this model includes the common case where all edge costs are chosen independently from the same distribution).

Better results are known for several important classes of sparse digraphs (i.e., digraphs with $m = O(n)$ edges) such as planar digraphs, outerplanar digraphs, digraphs of small genus, and digraphs of small treewidth.

For general sparse digraphs, an algorithm is given in [8] that solves the negative cycle problem in $O(n + \tilde{\gamma}^{1.5} \log \tilde{\gamma})$ time, where $\tilde{\gamma}$ is a topological measure of the input sparse digraph G, and whose value varies from 1 up to $\Theta(n)$. Informally, $\tilde{\gamma}$ represents the minimum number of outerplanar subgraphs, satisfying certain separation properties, into which G can be decomposed. In particular, $\tilde{\gamma}$ is proportional to $\gamma(G) + q$, where G is supposed to be embedded into an orientable surface of genus $\gamma(G)$ so as to minimize the number q of faces that collectively cover all vertices. For instance, if G is outerplanar, then $\tilde{\gamma} = 1$, which implies an optimal $O(n)$ time algorithm for this case. The algorithm in [8] does not require such an embedding to be provided by the input. In the same paper, it is shown that random $G_{n,p}$ graphs with threshold function $1/n$ are planar with probability one and have an expected value for $\tilde{\gamma}$ equal to $O(1)$. Furthermore, an efficient parallelization of the algorithm on the CREW PRAM model of computation is provided in [8].

Better bounds for planar digraphs are as follows. If edge costs are integers, then an algorithm running in $O(n^{4/3}\log(nL))$ time is given in [9]. For real edge costs, an $O(n \log^3 n)$-time algorithm was given in [5].

An optimal $O(n)$-time algorithm is given in [3] for the case of digraphs with small treewidth (and real edge costs). Informally, the treewidth t of a graph G is a parameter which measures how close is the structure of G to a tree. For instance, the class of graphs of small treewidth includes series-parallel graphs ($t = 2$) and outerplanar graphs ($t = 2$). An optimal parallel algorithm for the same problem, on the EREW PRAM model of computation, is provided in [2].

Applications

Finding negative cycles in a digraph is a fundamental combinatorial and network optimization problem that spans a wide range of applications including: shortest path computation, two dimensional package element, minimum cost flows, minimal cost-to-time ratio, model verification, compiler construction, software engineering,

VLSI design, scheduling, circuit production, constraint programming and image processing. For instance, the isolation of negative feedback loops is imperative in the design of VLSI circuits. It turns out that such loops correspond to negative cost cycles in the so-called amplifier-gain graph of the circuit. In constraint programming, it is required to check the feasibility of sets of constraints. Systems of difference constraints can be represented by constraint graphs, and one can show that such a system is feasible if and only if there are no negative cost cycles in its corresponding constraint graph. In zero-clairvoyant scheduling, the problem of checking whether there is a valid schedule in such a scheduling system can be reduced to detecting negative cycles in an appropriately defined graph. For further discussion on these and other applications see [1, 12, 14].

Open Problems

The negative cycle problem is closely related to the shortest path problem. The existence of negative edge costs makes the solution of the negative cycle problem or the computation of a shortest path tree more difficult and thus more time consuming compared to the time required to solve the shortest path tree problem in digraphs with non-negative edge costs. For instance, for digraphs with real edge costs, compare the $O(nm)$-time algorithm in the former case with the $O(m + n \log n)$-time algorithm for the latter case (Dijkstra's algorithm implemented with an efficient priority queue; see e.g., [1]).

It would therefore be interesting to try to reduce the gap between the above two time complexities, even for special classes of graphs or the case of integer costs.

The only case where these two complexities coincide concerns the digraphs of small treewidth [3], making it the currently most general such class of graphs. For planar digraphs, the result in [5] is only a polylogarithmic factor away from the $O(n)$-time algorithm in [9] that computes a shortest path tree when the edge costs are non-negative.

Experimental Results

An experimental study for the negative cycle problem is conducted in [4]. In that paper, several methods that combine a shortest path algorithm (based on the Bellman–Ford approach) with a cycle detection strategy are investigated, along with some new variations of them. It turned out that the performance of algorithms for the negative cycle problem depends on the number and the size of the negative cycles. This gives rise to a collection of problem families for testing negative cycle algorithms.

A follow-up of the above study is presented in [14], where two new heuristics are introduced and are incorporated on three of the algorithms considered in [4] (the original Bellman–Ford and the variations in [13] and [7]), achieving dramatic improvements. The data sets considered in [14] are those in [4].

Data Sets

Data set generators and problem families are described in [4], and are available from http://www.avglab.com/andrew/soft.html.

URL to Code

The code used in [4] is available from http://www.avglab.com/andrew/soft.html.

Cross-References

- All Pairs Shortest Paths in Sparse Graphs
- All Pairs Shortest Paths via Matrix Multiplication
- Single-Source Shortest Paths

Recommended Reading

1. Ahuja R, Magnanti T, Orlin J (1993) Network flows. Prentice-Hall, Englewood Cliffs
2. Chaudhuri S, Zaroliagis C (1998) Shortest paths in digraphs of small treewidth. Part II: optimal parallel algorithms. Theor Comput Sci 203(2):205–223

3. Chaudhuri S, Zaroliagis C (2000) Shortest paths in digraphs of small treewidth. Part I: sequential algorithms. Algorithmica 27(3):212–226
4. Cherkassky BV, Goldberg AV (1999) Negative-cycle detection algorithms. Math Program 85:277–311
5. Fakcharoenphol J, Rao S (2001) Planar graphs, negative weight edges, shortest paths, and near linear time. In: Proceedings of 42nd IEEE symposium on foundations of computer science (FOCS 2001). IEEE Computer Society, Los Alamitos, pp 232–241
6. Goldberg AV (1995) Scaling algorithms for the shortest paths problem. SIAM J Comput 24:494–504
7. Goldberg AV, Radzik T (1993) A heuristic improvement of the Bellman-Ford algorithm. Appl Math Lett 6(3):3–6
8. Kavvadias D, Pantziou G, Spirakis P, Zaroliagis C (1994) Efficient sequential and parallel algorithms for the negative cycle problem. In: Algorithms and computation (ISAAC'94). Lecture notes computer science, vol 834. Springer, Heidelberg, pp 270–278
9. Klein P, Rao S, Rauch M, Subramanian S (1997) Faster shortest path algorithms for planar graphs. J Comput Syst Sci 5(1):3–23
10. Kolliopoulos SG, Stein C (1998) Finding real-valued single-source shortest paths in $o(n^3)$ expected time. J Algorithm 28:125–141
11. Mehlhorn K, Näher S (1999) LEDA: a platform for combinatorial and geometric computing. Cambridge University Press, Cambridge
12. Spirakis P, Tsakalidis A (1986) A very fast, practical algorithm for finding a negative cycle in a digraph. In: Proceedings of 13th ICALP, pp 397–406
13. Tarjan RE (1983) Data structures and network algorithms. SIAM, Philadelphia
14. Wong CH, Tam YC (2005) Negative cycle detection problem. In: Algorithms – ESA 2005. Lecture notes in computer science, vol 3669. Springer, Heidelberg, pp 652–663

Network Creation Games

Erik D. Demaine[1], Mohammad Taghi Hajiaghayi[2], Hamid Mahini[2], and Morteza Zadimoghaddam[3]
[1]MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA
[2]Department of Computer Science, University of Maryland, College Park, MD, USA
[3]Google Research, New York, NY, USA

Keywords

Game theory; Network formation; Price of anarchy; Small-world phenomenon

Years and Authors of Summarized Original Work

2015; Demaine, Hajiaghayi, Mahini, Zadimoghaddam

Problem Definition

Over the last few decades, a wide variety of networks have emerged. The general structure of these networks including their global connectivity properties has been studied extensively. On the other hand, strategic aspects of them are also very interesting to explore by considering the nodes as independent agents. The exciting area of network creation games attempts to understand how real-world networks (such as the Internet) develop when multiple independent agents (e.g., ISPs) build pieces of the network to selfishly improve their own objective functions which heavily depend on their connectivity properties.

We start by elaborating on these connectivity objectives and its relation to the global design and structure of the network. Network design is a fundamental family of problems at the intersection between computer science and operations research, amplified in importance by the sustained growth of computer networks such as the Internet. Traditionally, the goal is to find a minimum-cost (sub) network that satisfies some specified property such as k-connectivity or connectivity on terminals (as in the classic Steiner tree problem). Such a formulation captures the (possibly incremental) creation cost of the network but does not incorporate the cost of actually using the network. By contrast, network routing has the goal of optimizing the usage cost of the network but assumes that the network has already been created. The network creation game attempts to unify the network design and network routing problems by modeling both creation and usage costs. Specifically, each node in the system is an independent selfish agent that can create a link (edge) to any other node, at a cost of α. In addition to these creation costs, each node incurs a usage cost related to the distances to the other nodes. In the model introduced by

Fabrikant, Luthra, Maneva, Papadimitriou, and Shenker [11], the usage cost incurred by a node is the sum of distances to all other nodes. Equivalently, if we divide the cost (and thus α) by the number n of nodes, the usage cost is the average distance to other nodes. In another natural model, the usage cost incurred by a node is the maximum distance to all other nodes: this model captures the worst-case instead of average-case behavior of routing. To model the dominant behavior of large-scale networking scenarios such as the Internet, we consider each node to be an agent (player) [12] that selfishly tries to minimize its own creation and usage costs [1, 6, 11]. In this context, the price of anarchy [14, 15, 17] is the worst possible ratio of the total cost found by some independent selfish behavior and the optimal total cost possible by a centralized, social welfare-maximizing solution. The price of anarchy is a well-studied concept in algorithmic game theory for problems such as load balancing, routing, and network design; see, e.g., [1, 3–7, 11, 15, 16].

Equilibria To model the dominant behavior of large-scale networking scenarios such as the Internet, we consider the case where every node (player) selfishly tries to minimize its own creation and usage cost. This game-theoretic setting naturally leads to the various kinds of equilibria and the study of their structure. Two frequently considered notions are Nash equilibrium, where no player can change its strategy (which edges to buy) to locally improve its cost, and strong Nash equilibrium, where no coalition of players can change their collective strategy to locally improve the cost of each player in the coalition. Nash equilibria capture the combined effect of both selfishness and lack of coordination, while strong Nash equilibria separate these issues, enabling coordination and capturing the specific effect of selfishness. However, the notion of strong Nash equilibrium is extremely restrictive in our context, because all players can simultaneously change their entire strategies, abusing the local optimality intended by original Nash equilibria and effectively forcing globally near-optimal solutions. Thus it makes sense to focus on weaker notions of equilibria.

Structure of equilibria What structural properties can be predicted about equilibria in network creation games? For example, Fabrikant et al. [11] conjectured that equilibrium graphs in the unilateral model were all trees, but this is not always the case as shown by Albers et al. [1] One particularly interesting structural feature is whether all equilibrium graphs have small diameter (say, polylogarithmic), analogous to the small-world phenomenon. A closely related issue is the price of anarchy, that is, the worst possible ratio of the total cost of an equilibrium (found by independent selfish behavior) and the optimal total cost possible by a centralized solution (maximizing social welfare). The price of anarchy is a well-studied concept in algorithmic game theory for problems such as load balancing, routing, and network design. Upper bounds on diameter of equilibrium graphs translate to approximately equal upper bounds on the price of anarchy but not necessarily vice versa.

Notation

Formally, we define four games depending on the objective (sum or max) and the consent (unilateral or bilateral). In all versions, we have n players; call them $1, 2, \ldots, n$. The strategy of player i is specified by a subset s_i of $1, 2, \ldots, n \setminus i$, which corresponds to the set of neighbors to which player i forms a link. Together, let $s = \{s_1, s_2, \ldots, s_n\}$ denote the strategies of all players.

To define the cost of a strategy, we introduce an undirected graph G_s with vertex set $\{1, 2, \ldots, n\}$. In the unilateral game, G_s has an edge (i, j) if either $i \in s_j$ or $j \in s_i$. In the bilateral game, G_s has an edge (i, j) if both $i \in s_j$ and $j \in s_i$. Define $d_s(i, j)$ to be the distance (the number of edges in a shortest path) between vertices i and j in graph G_s. In the sum game, the cost incurred by player i is $c_i(s) = \alpha|s_i| + \sum_{j=1}^{n} d_s(i, j)$, and in the max game, the cost incurred by player i is $c_i(s) = \alpha|s_i| + \max_{j=1}^{n} d_s(i, j)$. In both cases, the total cost incurred by strategy s is $c(s) = \sum_{i=1}^{n} c_i(s)$. In the unilateral game, a (pure) Nash equilibrium is a strategy s such that $c_i(s) \leq c_i(s')$ for all strategies s' that differ from s in only one player i. The price of anarchy is then the maximum cost of

a Nash equilibrium divided by the minimum cost of any strategy (called the social optimum).

In the bilateral game, Nash equilibria are not so interesting because the game requires coalition between two players to create an edge (in general). For example, if every player i chooses the empty strategy $s_i = \emptyset$, then we obtain a Nash equilibrium inducing an empty graph G_s, which has an infinite cost $c(s)$. To address this issue, Corbo and Parkes [6] use the notion of pairwise stability [13]: a strategy is pairwise stable if (1) for any edge (i, j) of G_s, both $c_i(s) \leq c_i(s')$ and $c_j(s) \leq c_j(s')$ where s' differs from s only in deleting edge (i, j) from G_s and (2) for any non-edge (i, j) of G_s, either $c_i(s) < c_i(s')$ or $c_j(s) < c_j(s')$ where s' differs from s only in adding edge (i, j) to $G_{s'}$. The price of anarchy is then the maximum cost of a pairwise-stable strategy divided by the social optimum (the minimum cost of any strategy).

Key Results

We start by the sum unilateral games. Fabrikant et al. [11] introduce these games and prove an upper bound of $O(\sqrt{\alpha})$ on the price of anarchy for all α. Albers et al. [1] prove that the price of anarchy is constant for $\alpha = O(\sqrt{n})$, as well as for the larger range $\alpha \geq 12n \lg(n)$. In addition, Albers et al. prove a general upper bound of $15\left(1 + \left(\min\{\frac{\alpha^2}{n}, \frac{n^2}{\alpha}\}\right)^{1/3}\right)$. The latter bound shows the first sublinear worst-case bound, $O(n^{1/3})$, for all α. Demaine et al. [10] prove the first $o(n^{\epsilon})$ upper bound on the price of anarchy for general α, namely, $2^{O(\sqrt{\log(n)})}$. They also prove that price of anarchy is constant for $\alpha = O(n^{1\epsilon})$ for any fixed $\epsilon > 0$, substantially reducing the range of α for which constant bounds have not been obtained. Demain et al. also prove that in the max unilateral games, the price of anarchy is at most 2 for $\alpha \geq n$, $O\left(\min\{4^{\sqrt{\log(n)}}, (n/\alpha)^{1/3}\}\right)$ for $2\sqrt{\log(n)} \leq \alpha \leq n$, and $O(n^{2/\alpha})$ for $\alpha < 2\sqrt{\log(n)}$. Alon et al. [2] consider a natural version of network creation games in which nodes only can switch their edges instead of drastically changing their strategies. In these simpler games, they achieve similar bounds on the price of anarchy. The advantage of their model is its simplicity in both agents strategies at each point and the fact that there is no α to be considered in their model.

The bilateral variation on the network creation game, considered by Corbo and Parkes [6], requires both nodes to agree before they can create a link between them. In the sum bilateral network creation game, Corbo and Parkes prove that the price of anarchy is $O(\min\{\sqrt{\alpha}, n/\sqrt{\alpha}\})$. Demaine et al. [10] prove that this upper bound is tight by showing a matching lower bound of $\Omega(\min\{\sqrt{\alpha}, n/\sqrt{\alpha}\})$. For the max bilateral case, Demaine et al. show that the price of anarchy is $\Theta(\frac{n}{1+\alpha})$ for $\alpha \leq n$ and at most 2 for $\alpha > n$.

Finding a polylogarithmic upper bound on price of anarchy for all values of α in these four network creation settings remains an open problem. In an effort to reduce the upper bounds, Demaine et al. [9] introduce the cooperative network creation games in which all agents can contribute in the construction of any edge even if they are not an endpoint of the edge. They prove that in the sum cooperative network creation game the price of anarchy is at most polylogarithmic in terms of the number of nodes. As a result, they exhibit the small-world phenomenon (polylogarithmic diameter) in the equilibrium graphs of these games. To reduce the price of anarchy even further, Demaine et al. [8] consider a special version of network creation games, and using some kind of an advertising campaign, they show that the price of anarchy is a constant number independent of the number of nodes.

Techniques

To keep this survey of results short, we just overview some of the nice combinatorial techniques in this area. Albers et al. [1] observe that any node u has the option of just connecting to another node v and exploits the BFS tree rooted at v. In an equilibrium graph G_s, this should not be a better strategy for u. Applying this trick and summing up all these inequalities for different

choices of u, Albers et al. prove that for any Nash equilibrium s and any vertex v in G_s, the cost $c(s)$ is at most $2\alpha(n-1)+n\text{Dist}(v)+(n-1)^2$ where $\text{Dist}(v)=\sum_{v'\in V(G_s)} d_s(v,v')$.

Demaine et al. use this lemma to prove that price of anarchy is $O(D)$ where D is the diameter of the graph. To upper bound the diameter, they develop different techniques for different ranges of α. For instance, for $\alpha = O(n^{1-\epsilon})$, they prove that the neighborhood sizes around any node grows exponentially with a rate of $n/\alpha = \Omega(n^{\epsilon})$. Formally, they prove that when the radius of the neighborhood around a node is doubled, the number of nodes inside the neighborhood is multiplied by n/α until this radius becomes comparable with the diameter of the graph D. Clearly, it takes $O(1/\epsilon)$ rounds of doubling the neighborhood radius to cover all nodes which means that the diameter is at most exponentially growing in $1/\epsilon$ which is a constant for a fixed ϵ. For other ranges of α, more complicated bounds are needed.

Recommended Reading

1. Albers S, Eilts S, Even-Dar E, Mansour Y, Roditty L (2006) On nash equilibria for a network creation game. In: Proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithms (SODA 2006), Miami, 22–26 Jan 2006, pp 89–98. http://dl.acm.org/citation.cfm?id=1109557.1109568
2. Alon N, Demaine ED, Hajiaghayi MT, Leighton T (2013) Basic network creation games. SIAM J Discret Math 27(2):656–668. doi:10.1137/090771478, http://dx.doi.org/10.1137/090771478
3. Anshelevich E, Dasgupta A, Tardos É, Wexler T (2003) Near-optimal network design with selfish agents. In: Proceedings of the 35th annual ACM symposium on theory of computing, San Diego, 9–11 June 2003, pp 511–520. doi:10.1145/780542.780617, http://doi.acm.org/10.1145/780542.780617
4. Anshelevich E, Dasgupta A, Kleinberg JM, Tardos É, Wexler T, Roughgarden T (2004) The price of stability for network design with fair cost allocation. In: Proceedings of the 45th symposium on foundations of computer science (FOCS 2004), Rome, 17–19 Oct 2004, pp 295–304. doi:10.1109/FOCS.2004.68, http://dx.doi.org/10.1109/FOCS.2004.68
5. Chun B, Fonseca R, Stoica I, Kubiatowicz J (2004) Characterizing selfishly constructed overlay routing networks. In: Proceedings IEEE INFOCOM 2004, the 23rd annual joint conference of the IEEE computer and communications societies, Hong Kong, 7–11 Mar 2004. http://www.ieee-infocom.org/2004/Papers/28_4.PDF
6. Corbo J, Parkes DC (2005) The price of selfish behavior in bilateral network formation. In: Proceedings of the twenty-fourth annual ACM symposium on principles of distributed computing (PODC 2005), Las Vegas, 17–20 July 2005, pp 99–107. doi:10.1145/1073814.1073833, http://doi.acm.org/10.1145/1073814.1073833
7. Czumaj A, Vöcking B (2002) Tight bounds for worst-case equilibria. In: Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms, San Francisco, 6–8 Jan 2002, pp 413–420. http://dl.acm.org/citation.cfm?id=545381.545436
8. Demaine ED, Zadimoghaddam M (2012) Constant price of anarchy in network-creation games via public-service advertising. Internet Math 8(1–2):29–45. doi:10.1080/15427951.2012.625251, http://dx.doi.org/10.1080/15427951.2012.625251
9. Demaine ED, Hajiaghayi MT, Mahini H, Zadimoghaddam M (2009) The price of anarchy in cooperative network creation games. SIGecom Exch 8(2):2. doi:10.1145/1980522.1980524, http://doi.acm.org/10.1145/1980522.1980524
10. Demaine ED, Hajiaghayi MT, Mahini H, Zadimoghaddam M (2012) The price of anarchy in network creation games. ACM Trans Algorithms 8(2):13. doi:10.1145/2151171.2151176, http://doi.acm.org/10.1145/2151171.2151176
11. Fabrikant A, Luthra A, Maneva EN, Papadimitriou CH, Shenker S (2003) On a network creation game. In: Proceedings of the twenty-second ACM symposium on principles of distributed computing (PODC 2003), Boston, 13–16 July 2003, pp 347–351. doi:10.1145/872035.872088, http://doi.acm.org/10.1145/872035.872088
12. Jackson MO (2003) A survey of models of network formation: stability and efficiency. In: Demange G, Wooders M (eds) Group formation in economics: networks, clubs and coalitions. Cambridge University Press, Cambridge
13. Jackson M, Wolinsky A (1996) A strategic model of social and economic networks. J Econ Theory 71(1):44–74. http://EconPapers.repec.org/RePEc:eee:jetheo:v:71:y:1996:i:1:p:44-74
14. Koutsoupias E, Papadimitriou CH (1999) Worst-case equilibria. In: Proceedings of the 16th annual symposium on theoretical aspects of computer science (STACS 99), Trier, 4–6 Mar 1999, pp 404–413. doi:10.1007/3-540-49116-3_38, http://dx.doi.org/10.1007/3-540-49116-3_38
15. Papadimitriou CH (2001) Algorithms, games, and the internet. In: Proceedings on 33rd annual ACM symposium on theory of computing, Heraklion, 6–8 July 2001, pp 749–753. doi:10.1145/380752.380883, http://doi.acm.org/10.1145/380752.380883
16. Roughgarden T (2002) The price of anarchy is independent of the network topology. In: Proceed-

ings on 34th annual ACM symposium on theory of computing, Montréal, 19–21 May 2002, pp 428–437. doi:10.1145/509907.509971, http://doi.acm.org/10.1145/509907.509971
17. Roughgarden T (2002) Selfish routing. PhD thesis, Cornell University

Non-approximability of Bimatrix Nash Equilibria

Xi Chen[1,2] and Xiaotie Deng[3,4]
[1]Computer Science Department, Columbia University, New York, NY, USA
[2]Computer Science and Technology, Tsinghua University, Beijing, China
[3]AIMS Laboratory (Algorithms-Agents-Data on Internet, Market, and Social Networks), Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[4]Department of Computer Science, City University of Hong Kong, Hong Kong, China

Keywords

Approximate Nash equilibrium

Years and Authors of Summarized Original Work

2006; Chen, Deng, Teng

Problem Definition

In this entry, the following two problems are considered: (1) the problem of finding an approximate Nash equilibrium in a positively normalized bimatrix (or two-player) game; and (2) the smoothed complexity of finding an exact Nash equilibrium in a bimatrix game. It turns out that these two problems are strongly correlated [3].

Let $\mathcal{G} = (\mathbf{A}, \mathbf{B})$ be a bimatrix game, where $\mathbf{A} = (a_{i,j})$ and $\mathbf{B} = (b_{i,j})$ are both $n \times n$ matrices. Game $\mathcal{G}$ is said to be positively normalized, if $0 \le a_{i,j}, b_{i,j} \le 1$ for all $1 \le i, j \le n$.

Let $\mathbb{P}^n$ denote the set of all probability vectors in $\mathbb{R}^n$, i.e., non-negative vectors whose entries sum to 1. A Nash equilibrium [8] of $\mathcal{G} = (\mathbf{A}, \mathbf{B})$ is a pair of mixed strategies $(\mathbf{x}^* \in \mathbb{P}^n, \mathbf{y}^* \in \mathbb{P}^n)$ such that for all $\mathbf{x}, \mathbf{y} \in \mathbb{P}^n$,

$$(\mathbf{x}^*)^{\mathrm{T}}\mathbf{A}\mathbf{y}^* \ge \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{y}^* \quad \text{and} \quad (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y}^* \ge (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y},$$

while an ϵ-approximate Nash equilibrium is a pair $(\mathbf{x}^* \in \mathbb{P}^n, \mathbf{y}^* \in \mathbb{P}^n)$ that satisfies

$$(\mathbf{x}^*)^{\mathrm{T}}\mathbf{A}\mathbf{y}^* \ge \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{y}^* - \epsilon \quad \text{and}$$
$$(\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y}^* \ge (\mathbf{x}^*)^{\mathrm{T}}\mathbf{B}\mathbf{y} - \epsilon, \quad \text{for all} \quad \mathbf{x}, \mathbf{y} \in \mathbb{P}^n.$$

In the smoothed analysis [11] of bimatrix games, a perturbation of magnitude $\sigma > 0$ is first applied to the input game: For a positively normalized $n \times n$ game $\mathcal{G} = (\overline{\mathbf{A}}, \overline{\mathbf{B}})$, let $\mathbf{A}$ and $\mathbf{B}$ be two matrices with

$$a_{i,j} = \overline{a}_{i,j} + r^A_{i,j} \quad \text{and} \quad b_{i,j} = \overline{b}_{i,j} + r^B_{i,j},$$
$$\forall 1 \le i, j \le n,$$

while $r^A_{i,j}$ and $r^B_{i,j}$ are chosen independently and uniformly from interval $[-\sigma, \sigma]$ or from Gaussian distribution with variance σ^2. These two kinds of perturbations are referred to as *σ-uniform* and *σ-Gaussian perturbations*, respectively. An algorithm for bimatrix games has *polynomial smoothed complexity* (under σ-uniform or σ-Gaussian perturbations) [11], if it finds a Nash equilibrium of game $(\mathbf{A}, \mathbf{B})$ in expected time poly $(n, 1/\sigma)$, for all $(\overline{\mathbf{A}}, \overline{\mathbf{B}})$.

Key Results

The complexity class **PPAD** [9] is defined in entry ▸ Complexity of Bimatrix Nash Equilibria. The following theorems are proved in [3].

Theorem 1 *For any constant $c > 0$, the problem of computing a $1/n^c$-approximate Nash equilibrium of a positively normalized $n \times n$ bimatrix game is* **PPAD***-complete.*

Theorem 2 *The problem of computing a Nash equilibrium in a bimatrix game is not in smoothed polynomial time, under uniform or Gaussian perturbations, unless* **PPAD** $\subseteq$ **RP**.

Corollary 1 *The smoothed complexity of the Lemke-Howson algorithm is not polynomial, under uniform or Gaussian perturbations, unless* **PPAD** $\subseteq$ **RP**.

Applications

See entry ▸ Complexity of Bimatrix Nash Equilibria.

Open Problems

There remains a complexity gap on the approximation of Nash equilibria in bimatrix games: The result of [7] shows that, an ϵ-approximate Nash equilibrium can be computed in $n^{O(\log n/\epsilon^2)}$-time, while [3] show that no algorithm can find an ϵ-approximate Nash equilibrium in poly$(n, 1/\epsilon)$-time for ϵ of order $1/\text{poly}(n)$, unless **PPAD** is in **P**. However, the hardness result of [3] does not cover the case when ϵ is a constant between 0 and 1. Naturally, it is unlikely that the problem of finding an ϵ-approximate Nash equilibrium is **PPAD**-complete when ϵ is an absolute constant, for otherwise, all the search problems in **PPAD** would be solvable in $n^{O(\log n)}$-time, due to the result of [7]. An interesting open problem is that, for every constant $\epsilon > 0$, is there a polynomial-time algorithm for finding an ϵ-approximate Nash equilibrium? The following conjectures are proposed in [3]:

Conjecture 1 There is an $O(n^{k+\epsilon^{-c}})$-time algorithm for finding an ϵ-approximate Nash equilibrium in a bimatrix game, for some constants c and k.

Conjecture 2 There is an algorithm to find a Nash equilibrium in a bimatrix game with smoothed complexity $O(n^{k+\sigma^{-c}})$ under perturbations with magnitude σ, for some constants c and k.

It is also conjectured in [3] that Corollary 1 remains true without any complexity assumption on class **PPAD**. A positive answer would extend the result of [10] to the smoothed analysis framework.

Cross-References

- ▸ Complexity of Bimatrix Nash Equilibria
- ▸ General Equilibrium
- ▸ Leontief Economy Equilibrium

Recommended Reading

1. Chen X, Deng X (2005) 3-Nash is PPAD-complete. ECCC, TR05-134
2. Chen X, Deng X (2006) Settling the complexity of two-player Nash equilibrium. In: FOCS'06: proceedings of the 47th annual IEEE symposium on foundations of computer science, pp 261–272
3. Chen X, Deng X, Teng SH (2006) Computing Nash equilibria: approximation and smoothed complexity. In: FOCS'06: proceedings of the 47th annual IEEE symposium on foundations of computer science, pp 603–612
4. Daskalakis C, Goldberg PW, Papadimitriou CH (2006) The complexity of computing a Nash equilibrium. In: STOC'06: proceedings of the 38th ACM symposium on theory of computing, pp 71–78
5. Daskalakis C, Papadimitriou CH (2005) Three-player games are hard. ECCC, TR05-139
6. Goldberg PW, Papadimitriou CH (2006) Reducibility among equilibrium problems. In: STOC'06: proceedings of the 38th ACM symposium on theory of computing, pp 61–70
7. Lipton R, Markakis E, Mehta A (2003) Playing large games using simple strategies. In: Proceedings of the 4th ACM conference on electronic commerce, pp 36–41
8. Nash JF (1950) Equilibrium point in n-person games. Proc Natl Acad Sci USA 36(1):48–49
9. Papadimitriou CH (1994) On the complexity of the parity argument and other inefficient proofs of existence. J Comput Syst Sci 48:498–532
10. Savani R, von Stengel B (2004) Exponentially many steps for finding a Nash equilibrium in a bimatrix game. In: FOCS'04: proceedings of the 45th annual IEEE symposium on foundations of computer science, Rome, pp 258–267

11. Spielman DA, Teng SH (2006) Smoothed analysis of algorithms and heuristics: progress and open questions. In: Pardo LM, Pinkus A, Süli E, Todd MJ (eds) Foundations of computational mathematics, Cambridge University Press, Cambridge, pp 274–342

Non-shared Edges

Wing-Kai Hon
Department of Computer Science, National Tsing Hua University, Hsin Chu, Taiwan

Keywords

Robinson-Foulds distance; Robinson-Foulds metric

Years and Authors of Summarized Original Work

1985; Day

Problem Definition

Phylogenies are binary trees whose leaves are labeled with distinct leaf labels. This problem in this article is concerned with a well-known measurement, called *non-shared edge distance*, for comparing the dissimilarity between two phylogenies. Roughly speaking, the non-shared edge distance counts the number of edges that differentiate one phylogeny from the other.

Let e be an edge in a phylogeny T. Removing e from T splits T into two subtrees. The leaf labels are partitioned into two subsets according to the subtrees. The edge e is said to *induce* a partition of the set of leaf labels. Given two phylogenies T and T' having the same number of leaves with the same set of leaf labels, an edge e in T is *shared* if there exists some edge e' in T' such that the edges e and e' induce the same partition of the set of leaf labels in their corresponding tree. Otherwise, e is *non-shared*. Notice that T and T' have the same number of edges, so that the number of non-shared edges in T (with respect to T') is the same as the number of non-shared edges in T' (with respect to T). Such a number is called the *non-shared edge distance* between T and T'. Two problems are defined as follows:

Non-shared Edge Distance Problem

INPUT: Two phylogenies on the same set of leaf labels
OUTPUT: The non-shared edge distance between the two input phylogenies

All-Pairs Non-shared Edge Distance Problem

INPUT: A collection of phylogenies on the same set of leaf labels
OUTPUT: The non-shared edge distance between each pair of the input phylogenies

Extension

Phylogenies that are commonly used in practice have weights associated to the edges. The notion of non-shared edge can be easily extended for edge-weighted phylogenies. In this case, an edge e will induce a partition of the set of leaf labels as well as the multi-set of edge weights (here, edge weights are allowed to be non-distinct). Given two edge-weighted phylogenies R and R' having the same set of leaf labels and the same multi-set of edge weights, an edge e in R is *shared* if there exists some edge e' in R' such that the edges e and e' induce the same partition of the set of leaf labels and the multi-set of edge weights. Otherwise, e is *non-shared*. The *non-shared edge distance* between R and R' is similarly defined, giving the following problem:

General Non-shared Edge Distance Problem

INPUT: Two edge-weighted phylogenies on the same set of leaf labels and same multi-set of edge weights
OUTPUT: The non-shared edge distance between the two input phylogenies

Key Results

Day [3] proposed the first linear-time algorithm for the Non-shared Edge Distance Problem.

Theorem 1 *Let T and T' be two input phylogenies with the same set of leaf labels and n be the number of leaves in each phylogeny. The non-shared edge distance between T and T' can be computed in $O(n)$ time.*

Let Δ be a collection of k phylogenies on the same set of leaf labels and n be the number of leaves in each phylogeny. The All-Pairs Non-shared Edge Distance Problem can be solved by applying Theorem 1 on each pair of phylogenies, thus solving the problem in a total of $O(k^2 n)$ time. Pattengale and Moret [9] proposed a randomized result based on [7] to solve the problem approximately, whose running time is faster when $n \leq k \leq 2^n$.

Theorem 2 *Let ε be a parameter with $\varepsilon > 0$. Then, there exists a randomized algorithm such that with probability at least $1 - k^{-2}$, the non-shared edge distance between each pair of phylogenies in Δ can be approximated within a factor of $(1 + \varepsilon)$ from the actual distance; the running time of the algorithm is $O(k(n^2 + k \log k) / \varepsilon^2)$.*

For general phylogenies, let R and R' be two input phylogenies with the same set of leaf labels and the same multi-set of edge weights and n be the number of leaves in each phylogeny. The General Non-shared Edge Distance Problem can be solved easily in $O(n^2)$ time by applying Theorem 1 in a straightforward manner. The running time is improved by Hon et al. in [5].

Theorem 3 *The non-shared edge distance between R and R' can be computed in $O(n \log n)$ time.*

Applications

Phylogenies are commonly used by biologists to model the evolutionary relationship among species. Many reconstruction methods (such as maximum parsimony, maximum likelihood, compatibility, distance matrix) produce different phylogenies based on the same set of species, and it is interesting to compute the dissimilarities between them. Also, through the comparison, information about rare genetic events such as recombinations or gene conversions may be uncovered. The most common dissimilarity measure is the Robinson-Foulds metric [11], which is exactly the same as the non-shared edge distance.

Other dissimilarity measures, such as the nearest-neighbor interchange (NNI) distance and the subtree-transfer (STT) distance (see [2] for details), are also proposed in the literature. These measures are sometimes preferred by the biologists since they can be used to deduce the biological events that create the dissimilarity. Nevertheless, these measures are usually difficult to compute. In particular, computing the NNI distance and the STT distance is shown to be NP-hard by DasGupta et al. [1, 2]. Approximation algorithms are devised for these problems (NNI, [4, 8]; STT, [1, 6]). Interestingly, all these algorithms make use of the non-shared edge distance to bound their approximation ratios.

Recommended Reading

1. DasGupta B, He X, Jiang T, Li M, Tromp J (1999) On the linear-cost subtree-transfer distance between phylogenetic trees. Algorithmica 25(2–3):176–195
2. DasGupta B, He X, Jiang T, Li M, Tromp J, Zhang L (1997) On distances between phylogenetic trees. In: Proceedings of the eighth ACM-SIAM annual symposium on discrete algorithms (SODA), New Orleans. SIAM, pp 427–436
3. Day WHE (1985) Optimal algorithms for comparing trees with labeled leaves. J Classif 2:7–28
4. Hon WK, Lam TW (2001) Approximating the nearest neighbor intercharge distance for non-uniform-degree evolutionary trees. Int J Found Comp Sci 12(4):533–550
5. Hon WK, Kao MY, Lam TW, Sung WK, Yiu SM (2004) Non-shared edges and nearest neighbor interchanges revisited. Inf Process Lett 91(3):129–134
6. Hon WK, Lam TW, Yiu SM, Kao MY, Sung WK (2004) Subtree transfer distance for degree-D phylogenies. Int J Found Comp Sci 15(6):893–909
7. Johnson W, Lindenstrauss J (1984) Extensions of lipschitz mappings into a hilbert space. Contemp Math 26:189–206

8. Li M, Tromp J, Zhang L (1996) Some notes on the nearest neighbour interchange distance. J Theor Biol 26(182):463–467
9. Pattengale ND, Moret BME (2006) A sublinear-time randomized approximation scheme for the robinson-foulds metric. In: Proceedings of the tenth ACM annual international conference on research in computational molecular biology (RECOMB), Venice, pp 221–230
10. Robinson DF (1971) Comparison of labeled trees with valency three. J Comb Theor 11: 105–119
11. Robinson DF, Foulds LR (1981) Comparison of phylogenetic trees. Math Biosci 53:131–147

Nowhere Crownful Classes of Directed Graphs

Stephan Kreutzer
Chair for Logic and Semantics, Technical University, Berlin, Germany

Keywords

Algorithmic digraph structure theory; Algorithms for directed graphs; Digraph width measures; Dominating set problems; Nowhere dense classes of graphs; Sparse digraphs

Years and Authors of Summarized Original Work

2012; Kreutzer, Tazari

Problem Definition

Many common computational problems on directed graphs are computationally intractable; they are NP-complete and sometimes even harder. Examples include domination problems such as directed dominating set, Kernel, directed Steiner networks, directed disjoint paths, and many other problems.

For undirected graphs, there is an extensive structure theory available to help dealing with this computational intractability. In particular, there is a well-developed hierarchy of classes of undirected graphs and a rich set of algorithmic tools which allow to solve hard computational problems on these classes of graphs. Most notably in this context are classes of graphs of bounded tree width, planar graphs or graphs embeddable on any other fixed surface, classes excluding a fixed minor, and many other graph classes. This theory is closely related to parameterized complexity theory.

For directed graphs, to date, there is no comparable theory available. A directed version of tree width was introduced by Reed [9] and Johnson et al. [4]. Further proposals for "tree width"-like width measures for directed graphs have been made in the literature; see, e.g., references in [1]. Algorithmically, the main application is that on classes of bounded directed tree width, the directed k-disjoint paths problem can be solved in polynomial time for any fixed value of k.

Almost all of these proposals have in common that the class of acyclic digraphs (DAGs) have small width, i.e., acyclic digraphs are taken as particularly simple digraphs. While this is certainly useful for problems such as directed disjoint paths, problems such as directed dominating set remain NP-complete and fixed-parameter intractable on acyclic directed graphs.

What is needed, therefore, are digraph parameters and structural classes of digraphs which separate acyclic digraphs into simple and hard instances. Nowhere crownful classes propose a solution to this problem based on the concept of excluded directed minors.

Key Results

While there is a well-defined concept of a minor for undirected graphs, there is as yet no commonly agreed concept of directed minors. A widely used, and very conservative, version of directed minor is a *butterfly minor* (see, e.g., [4]) in which a directed edge (u, v) is contractible if it is the only outgoing edge of u or the only incoming edge of v. In [5] a much more general concept of directed minors is used to give a classification of classes of digraphs in terms of shallow directed minors. For the sake of brevity, we introduce

directed minors here only for digraphs called *crowns*, which is enough for defining nowhere crownful classes of digraphs.

An *out-branching* is a digraph H whose underlying undirected graph is a tree and in which there is a unique vertex r, the *root* of H, such that all edges are oriented away from the root, i.e., every vertex in H is reachable by a unique directed path from the root. An *in-branching* is the same as an out-branching but all edges are oriented towards the root.

Definition 1 A *crown* of order q, for $q > 0$, is the graph S_q with

- $V(S_q) := \{v_1, \ldots, v_q\} \dot\cup \{u_{i,j} : 1 \leq i < j \leq q\}$ and
- $E(S_q) := \{(u_{i,j}, v_i), (u_{i,j}, v_j) : 1 \leq i < j \leq q\}$.

Definition 2 Let H with $V(H) := \{v_1, \ldots, v_q\} \dot\cup \{u_{i,j} : 1 \leq i < j \leq q\}$ be a crown of order q, for some $q > 0$. A digraph G contains H as a *directed minor*, if for every v_i, $1 \leq i \leq q$, there is an in-branching $T_i \subseteq G$ and for every $u_{i,j}$, $1 \leq i < j \leq q$, there is an out-branching $S_{i,j} \subseteq G$ such that all subgraphs T_i, $S_{i,j}$ are pairwise vertex disjoint and for all $1 \leq i < j \leq q$, there are edges e_i, e_j from a vertex in $S_{i,j}$ to a vertex in T_i and T_j, respectively.

H is a *depth-r-minor* of G, or an *r-shallow minor* of G, denoted $H \preceq_r^d G$, for some $r \geq 0$, if all $S_{i,j}$ and all T_i are of height at most r.

Definition 3 A class $\mathcal{C}$ of directed graphs is *nowhere crownful* if for every $r \geq 0$ there exists a $q = q(r)$ so that $S_q \not\preceq_r^d G$ for all $G \in \mathcal{C}$. If the function taking each r to $q(r)$ as above is computable, then we call $\mathcal{C}$ *effectively nowhere crownful*.

Nowhere crownful classes of digraphs are very general. For instance, if $\mathcal{C}$ is a class of digraphs and $\tilde{\mathcal{C}}$ is the class of underlying undirected graphs (obtained from digraphs in $\mathcal{C}$ by ignoring edge direction), then if $\tilde{\mathcal{C}}$ has bounded genus, excludes a fixed minor, or is nowhere dense, then $\mathcal{C}$ is nowhere crownful. But there are nowhere crownful classes $\mathcal{C}$ of digraphs such that $\tilde{\mathcal{C}}$ does not have any of the properties above. On the other hand, the class of acyclic digraphs is not nowhere crownful as it contains every crown.

Nowhere crownful classes of digraphs can be characterized equivalently as follows. Let G be a digraph and $d \geq 0$. A set $U \subseteq V(G)$ is *d-scattered* if there is no $v \in V(G)$ and $u \neq u' \in U$ with $u, u' \in N_d^+(v)$. That is, no two elements of U can be reached from a single vertex v by paths of length at most d.

Definition 4 A class $\mathcal{C}$ of directed graphs is *uniformly quasi-wide* if there are functions $s : \mathbb{N} \to \mathbb{N}$ and $N : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for every $G \in \mathcal{C}$ and all $d, m \in \mathbb{N}$ and $W \subseteq V(G)$ with $|W| > N(d, m)$, there is a set $S \subseteq V(G)$ with $|S| \leq s(d)$ and a set $U \subseteq W$ with $|U| = m$ such that U is d-scattered in $G - S$. The functions s, N are called the *margin* of $\mathcal{C}$. If s and N are computable, then we call $\mathcal{C}$ effectively uniformly quasi-wide.

Theorem 1 *A class $\mathcal{C}$ of digraphs is nowhere crownful if, and only if, it is directed uniformly quasi-wide.*

Nowhere crownful classes of digraphs were defined as a directed analogue to the concept of *nowhere dense* classes of undirected graphs, for which a similar equivalence to uniformly quasi-wideness can be proved. See [2, 3, 6–8] for nowhere dense classes of graphs and algorithmic applications. As mentioned above, nowhere crownful classes properly generalise nowhere denseness to digraphs.

Applications

A *directed dominating set* in a digraph G is a set $X \subseteq V(G)$ such that every $u \in V(G) \setminus X$ is the out-neighbour of a vertex in X. A *distance-d directed dominating set* is a set $X \subseteq V(G)$ such that every vertex $v \in V(G)$ can be reached from a vertex in X by a directed path of length at most d. An important variant of the undirected dominating set problem is the *connected dominating set problem*, where we are asked to find a dominating set D of size k such that D induces a connected subgraph. There are various natural translations

of this problem to the directed case: we can require the dominating set to induce a strongly connected subgraph or we can simply require it to induce an out-branching. The second variation, which we call *dominating out-branching*, still captures the idea that information can flow from the root to all vertices in the dominating set.

There is an easy reduction from the undirected dominating set problem to its directed counterpart proving that the directed dominating set problem is fixed-parameter intractable and NP-complete. In fact, the problem to decide whether an undirected graph G contains a dominating set of order k can be reduced to the question whether an acyclic digraph contains a directed dominating set of order k. The result of the reduction is a crown (plus one extra vertex). So, classes of digraphs where this problem and its extension to distance-d versions are to become tractable should exclude crowns. This observation was one of the motivations for defining and studying nowhere crownful classes of digraphs. Furthermore, if besides the directed dominating set we also want to solve its distance-d version, we need to exclude crown minors in some form.

However, excluding shallow-crowns is sufficient for these problems to become fixed-parameter tractable.

Theorem 2 *Let $\mathcal{C}$ be a class of directed graphs which is nowhere crownful. Then the directed (independent or unrestricted) dominating set problem, the dominating out-branching problem, as well as their distance-d versions are fixed-parameter tractable on $\mathcal{C}$.*

In the same way, several other similar problems can be shown to become tractable on nowhere crownful classes.

Open Problems

As mentioned before, nowhere crownful classes are modelled after nowhere dense classes of undirected graphs. For such classes, many other equivalent characterizations and powerful algorithmic applications are known. For instance, nowhere dense classes of graphs allow for very efficient sparse neighborhood covers (and this is again if, and only if) and can be defined by a game yielding bounded search tree techniques. Furthermore, there is a close connection between nowhere dense classes of graphs and generalized colouring numbers (see, e.g., [8]).

It is open in how far these characterizations and applications can be generalized to the digraph setting.

A particular open problem is the tractability of strongly connected Steiner networks and strongly connected dominating set on nowhere crownful classes of digraphs.

Nowhere crownful classes provide a way for dealing with domination-type problems. Directed tree width on the other hand provides a way of dealing with linkage problems such as disjoint paths. It is open how to bring the two concepts together.

Cross-References

- ▸ Efficient Dominating and Edge Dominating Sets for Graphs and Hypergraphs
- ▸ Exact Algorithms for Dominating Set

Recommended Reading

1. Ganian R, Hlinený P, Kneis J, Langer A, Obdrzálek J, Rossmanith P (2009) On digraph width measures in parameterized algorithmics. In: International workshop in parameterized and exact computation (IWPEC), Copenhagen, pp 185–197
2. Grohe M, Kreutzer S, Siebertz S (2013) Characterisations of nowhere dense graphs. In: Foundations of software technology and theoretical computer science (FSTTCS 2013), Guwahati, pp 21–40
3. Grohe M, Kreutzer S, Siebertz S (2014) Deciding first-order properties of nowhere dense graphs. In: 46th annual symposium on the theory of computing (STOC), New York
4. Johnson T, Robertson N, Seymour PD, Thomas R (2001) Directed tree-width. J Comb Theory Ser B 82(1):138–154
5. Kreutzer S, Tazari S (2012) Directed nowhere dense classes of graphs. In: Proceedings of the 23rd ACM-SIAM symposium on discrete algorithms (SODA), Kyoto, pp 1552–1562
6. Nešetřil J, Ossona de Mendez P (2008) Grad and classes with bounded expansion I–III. Eur J Comb 29. Series of 3 papers appearing in volumes (3) and (4)

7. Nešetřil J, Ossona de Mendez P (2010) First order properties of nowhere dense structures. J Symb Log 75(3):868–887
8. Nešetřil J, Ossona de Mendez P (2012) Sparsity. Vol. 28 of Algorithms and Combinatorics. Springer Heidelberg
9. Reed B (1999) Introducing directed tree-width. Electron Notes Discret Math 3:222–229

Nucleolus

Qizhi Fang
School of Mathematical Sciences, Ocean University of China, Qingdao, Shandong Province, China

Keywords

Kernel; Nucleon

Years and Authors of Summarized Original Work

2006; Deng, Fang, Sun

Problem Definition

Cooperative game theory considers how to distribute the total income generated by a set of participants in a joint project to individuals. The Nucleolus, trying to capture the intuition of minimizing dissatisfaction of players, is one of the most well-known solution concepts among various attempts to obtain a unique solution. In Deng, Fang, and Sun's work [3], they study the Nucleolus of flow games from the algorithmic point of view. It is shown that, for a flow game defined on a simple network (arc capacity being all equal), computing the Nucleolus can be done in polynomial time, and for flow games in general cases, both the computation and the recognition of the Nucleolus are $\mathcal{NP}$-hard.

A cooperative (profit) game (N, v) consists of a player set $N = \{1, 2, \cdots, n\}$ and a characteristic function $v : 2^N \to R$ with $v(\emptyset) = 0$, where the value $v(S)(S \subseteq N)$ is interpreted as the profit achieved by the collective action of players in S. Any vector $x \in R^n$ with $\sum_{i \in N} x_i = v(N)$ is an allocation. An allocation x is called an *imputation* if $x_i \geq v(\{i\})$ for all $i \in N$. Denote by $\mathcal{I}(v)$ the set of imputations of the game.

Given an allocation x, the *excess* of a coalition $S(S \subseteq N)$ at x is defined as

$$e(S, x) = x(S) - v(S),$$

where $x(S) = \sum_{i \in S} x_i$ for $S \subseteq N$. The value $e(S, x)$ can be interpreted as a measure of satisfaction of coalition S with the allocation x. The core of the game (N, v), denoted by $\mathcal{C}(v)$, is the set of allocations whose excesses are all nonnegative. For an allocation x of the game (N, v), let $\theta(x)$ denote the $(2^n - 2)$-dimensional vector whose components are the nontrivial excesses $e(S, x)$, $\emptyset \neq S \neq N$, arranged in a non-decreasing order. That is, $\theta_i(x) \leq \theta_j(x)$, for $1 \leq i < j \leq 2^n - 2$. Denote by $\succeq_l$ the "lexicographically greater than" relationship between vectors of the same dimension.

Definition 1 The Nucleolus $\eta(v)$ of game (N, v) is the set of imputations that lexicographically maximize $\theta(x)$ over all imputations $x \in \mathcal{I}(v)$. That is,

$$\eta(v) = \{x \in \mathcal{I}(v) : \theta(x) \succeq_l \theta(y) \text{ for all } y \in \mathcal{I}(v)\}.$$

Even though, the Nucleolus may contain multiple points by the definition, it was proved by Schmeidler [14] that the Nucleolus of a game with nonempty imputation set contains exactly one element. Kopelowitz [12] proposed that the Nucleolus can be obtained by recursively solving sequential linear programs (SLP):

$$\mathrm{LP}_k: \begin{array}{l} \max\ \varepsilon \\ x(S) = v(S) + \varepsilon_r \quad \forall S \in \mathcal{J}_r \\ r = 0, 1, \cdots, k-1 \\ x(S) \geq v(S) + \varepsilon \quad \forall S \in 2^N \setminus \bigcup_{r=0}^{k-1} \mathcal{J}_r \\ x \in \mathcal{I}(v). \end{array}$$

Here, $\mathcal{J}_0 = \{\emptyset, N\}$ and $\varepsilon_0 = 0$ initially; the number ε_r is the optimum value of the rth program (LP_r) and $\mathcal{J}_r = \{S \in 2^N : x(S) = v(S) + \varepsilon_r$ for every $x \in X_r$, where $X_r = \{x \in \mathcal{I}(v) : (x, \varepsilon_r)$ is an optimal solution to $LP_r\}$. It can be shown that after at most $n - 1$ iterations, one arrives at a unique optimal solution (x^*, ε_k), where x^* is just the Nucleolus of the game. In addition, the set of optimal solutions X_1 to the first program LP_1 is called the least core of the game.

The definition of the Nucleolus entails comparisons between vectors of exponential length. And with linear programming approach, each linear programs in SLP may possess exponential size in the number of players. Clearly, both do not provide an efficient solution in general.

Flow games, first studied in Kailai and Zemel [9, 10], arise from the profit distribution problem related to the maximum flow in a network. Let $D = (V, E; \omega; s, t)$ be a directed flow network, where V is the vertex set, E is the arc set, $\omega : E \to R^+$ is the arc capacity function, and s and t are the source and the sink of the network, respectively. The network D is *simple* if $\omega(e) = 1$ for each $e \in E$, which is denoted briefly by $D = (V, E; s, t)$.

Definition 2 The flow game $\Gamma_f = (E, v)$ associated with network $D = (V, E; \omega; s, t)$ is defined by:

(i) The player set is E.
(ii) $\forall S \subseteq E$, $v(S)$ is the value of a maximum flow from s to t in the subnetwork of D consisting only of arcs belonging to S.

Problem 1 (Computing the Nucleolus)

INSTANCE: A flow network $D = (V, E; \omega; s, t)$.
QUESTION: Is there a polynomial time algorithm to compute the Nucleolus of the flow game associated with D?

Problem 2 (Recognizing the Nucleolus)

INSTANCE: A flow network $D = (V, E; \omega; s, t)$ and $y : E \to R^+$.
QUESTION: Is it true that y is the Nucleolus of the flow game associated with D?

Key Results

Theorem 1 *Let $D = (V, E; s, t)$ be a simple network and $\Gamma_f = (E, v)$ be the associated flow game. Then the Nucleolus $\eta(v)$ can be computed in polynomial time.*

By making use of duality technique in linear programming, Kalai and Zemel [10] gave a characterization on the core of a flow game. They further conjectured that their approach may serve as a practical basis for computing the Nucleolus. In fact, the proof of Theorem 1 in the work of Deng, Fang, and Sun [3] is just an elegant application of Kalai and Zemel's approach (especially the duality technique) and hence settling their conjecture.

Theorem 2 *Given a flow game $\Gamma_f = (E, v)$ defined on network $D = (V, E; \omega; s, t)$, computing the Nucleolus $\eta(v)$ is $\mathcal{NP}$-hard.*

Theorem 3 *Given a flow game $\Gamma_f = (E, v)$ defined on network $D = (V, E; \omega; s, t)$ and an imputation $y \in \mathcal{I}(v)$, checking whether y is the Nucleolus of Γ_f is $\mathcal{NP}$-hard.*

Although a flow game can be formulated as a linear production game [2], the size of reduction may in general be exponential in space, and consequently, their complexity results on the Nucleolus are independent. However, in the $\mathcal{NP}$-hardness proof of Theorems 2 and 3, the flow game constructed possesses a polynomial size formulation of linear production game [3]. Therefore, as a direct corollary, the same $\mathcal{NP}$-hardness conclusions for linear production games are obtained. That is, both computing and recognizing the Nucleolus of a linear production game are $\mathcal{NP}$-hard.

Applications

As an important solution concept in economics and game theory, the Nucleolus and related solution concepts have been applied to insurance policies, real estate and bankruptcy, etc. However, it is a challenging problem to decide what classes

of cooperative games permit polynomial time computation of the Nucleolus.

The first polynomial time algorithm for Nucleolus in a special tree game was proposed by Megiddo [13], in advocation of efficient algorithms for cooperative game solutions. Subsequently, some efficient algorithms have been developed for computing the Nucleolus, such as for assignment games [15] and matching games [1, 11]. On the negative side, $\mathcal{NP}$-hardness result was obtained for minimum cost spanning tree games [5] and weighted voting games [4].

Granot, Granot, and Zhu [8] observed that most of the efficient algorithms for computing the Nucleolus are based on the fact that the information needed to completely characterize the Nucleolus is much less than that dictated by its definition. Therefore, they introduced the concept of a characterization set for the Nucleolus to embody the notion of "minimum" relevant information needed for the Nucleolus. Furthermore, based on the sequential linear programs (SLP), they established a general relationship between the size of a characterization set and the complexity of computing the Nucleolus. Following this approach, some known efficient algorithms for computing the Nucleolus are derived directly.

Another approach to computing the Nucleolus is taken by Faigle, Kern, and Kuipers [6], which is motivated by Schmeidler's observation that the Nucleolus of a game lies in the kernel [14]. In the case where the kernel of the game contains exactly one core vector and the minimum excess for any given allocation can be computed efficiently, their approach derives a polynomial time algorithm for the Nucleolus. However, their algorithm uses the ellipsoid method as a subroutine, implying that the efficiency of the algorithm is of a more theoretical kind.

Open Problems

The field of combinatorial optimization has much to offer for the study of cooperative games. It is usually the case that the values of subgroups can be obtained via a combinatorial optimization problem, where the game is called a combinatorial optimization game. This class of games leads to the applications of a variety of combinatorial optimization techniques in design and analysis of algorithms, as well as establishing complexity results. One of the most interesting result is the LP duality characterization of the core [2]. However, little work dealt with the Nucleolus by using the duality technique so far. Hence, the work of Deng, Fang, and Sun [3] on computing the Nucleolus may be of independent interest.

There are still many unsolved complexity questions concerning the Nucleolus. For the computation of the Nucleolus of matching games, Kern and Paulusma [11] proposed an efficient algorithm in unweighted case and conjectured that it is in general $\mathcal{NP}$-hard. Biro, Kern, and Paulusma [1] partly settled the conjecture by showing that in weighted case, when the matching game has a nonempty core, the Nucleolus can be computed in polynomial time. Since both the flow game and the matching game fall into the class of packing/covering games, it is interesting to know the complexity of computing the Nucleolus for other game models in this class, such as vertex covering games and minimum coloring games.

For cooperative games arising from $\mathcal{NP}$-hard combinatorial optimization problems, the computation of the Nucleolus may in general be a hard task. For example, in a traveling salesman game, nodes of the graph are the players and an extra node 0, and the value of a subgroup S of players is the length of a minimum Hamiltonian tour in the subgraph induced by $S \cup \{0\}$ [2]. It would not be surprising if one shows that both the computation and the recognition of the Nucleolus for this game model are $\mathcal{NP}$-hard. However, this is not known yet. The same questions are proposed for facility location games [7], though there have been efficient algorithms for some special cases.

Moreover, when the computation of the Nucleolus is difficult, it is also interesting to seek for meaningful approximation concepts of the Nucleolus, especially from the political and economic background.

Cross-References

▸ Complexity of Core

Recommended Reading

1. Biro P, Kern W, Paulusma D (2012) Computing solutions for matching games. Int J Game Theory 41:75–90
2. Deng X (1998) Combinatorial optimization and coalition games. In: Du D, Pardalos PM (eds) Handbook of combinatorial optimization, vol 2. Kluwer, Boston, pp 77–103
3. Deng X, Fang Q, Sun X (2006) Finding nucleolus of flow games. In: Proceedings of the 17th annual ACM-SIAM symposium on discrete algorithm (SODA 2006), Miami. Lecture Notes in Computer Science, vol 3111, pp 124–131
4. Elkind P, Goldberg LA, Goldberg PW, Wooldridge M (2009) On the computational complexity of weighted voting games. Ann Math Artif Intell 56:109–131
5. Faigle U, Kern W, Kuipers J (1998) Computing the nucleolus of min-cost spanning tree games is $\mathcal{NP}$-hard. Int J Game Theory 27:443–450
6. Faigle U, Kern W, Kuipers J (2001) On the computation of the nucleolus of a cooperative game. Int J Game Theory 30:79–98
7. Goemans MX, Skutella M (2004) Cooperative facility location games. J Algorithms 50:194–214
8. Granot D, Granot F, Zhu WR (1998) Characterization sets for the nucleolus. Int J Game Theory 27:359–374
9. Kalai E, Zemel E (1982) Totally balanced games and games of flow. Math Oper Res 7:476–478
10. Kalai E, Zemel E (1982) Generalized network problems yielding totally balanced games. Oper Res 30:998–1008
11. Kern W, Paulusma D (2003) Matching games: the least core and the nucleolus. Math Oper Res 28:294–308
12. Kopelowitz A (1967) Computation of the kernels of simple games and the nucleolus of n-person games. RM-31, Math. Dept., The Hebrew University of Jerusalem, Jerusalem
13. Megiddo N (1978) Computational complexity and the game theory approach to cost allocation for a tree. Math Oper Res 3:189–196
14. Schmeidler D (1969) The nucleolus of a characteristic function game. SIAM J Appl Math 17:1163–1170
15. Solymosi T, Raghavan TES (1994) An algorithm for finding the nucleolus of assignment games. Int J Game Theory 23:119–143

O

O(log log *n*)-Competitive Binary Search Tree

Chengwen Chris Wang and Daniel Sleator
Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

Keywords

Competitive BST algorithms; Splay trees; Tango trees

Years and Authors of Summarized Original Work

2004; Demaine, Harmon, Iacono, Patrascu

Problem Definition

Here is a precise definition of BST algorithms and their costs. This model is implied by most BST papers and developed in detail by Wilber [22]. A static set of n keys is stored in the nodes of a binary tree. The keys are from a totally ordered universe, and they are stored in symmetric order. Each node has a pointer to its left child, to its right child, and to its parent. Also, each node may keep $o(\log\ n)$ bits of additional information but no additional pointers.

A BST algorithm is required to process a sequence of m accesses (without insertions or deletions), $S = s_1, s_2, s_3, s_4 \ldots s_m$. The ith access starts from the root and follows pointers until s_i is reached. The algorithm can update the fields in any node or rotate any edges that it touches along the way. The cost of the algorithm to execute an access sequence is defined to be the number of nodes touched plus the number of rotations. A BST algorithm is *on-line* if it processes access s_i without making use of anything after s_i in the access sequence.

Let A be any online BST algorithm, and define $A(S)$ to be the cost to algorithm A of processing sequence S and $\mathrm{OPT}(S, T_0)$ to be the minimum possible (online or off-line) cost to process the sequence S, starting from an initial tree T_0. The algorithm A is T-competitive if for all possible sequences S, $A(S) \leq T^*\ \mathrm{OPT}(S, T_0) + O(m+n)$.

Since the number of rotations needed to change any binary tree of n keys into another one (with the same n keys) is at most $2n - 6$ [4,5,12,13,15], it follows that $\mathrm{OPT}(S, T_0)$ differs from $\mathrm{OPT}(S, T_0{}')$ by at most $2n - 6$. Thus, if $m > n$, then the initial tree can only affect the constant factor.

Key Results

The *interleave bound* is a lower bound on $\mathrm{OPT}(S, T_0)$ that depends only on S. Consider any binary search tree P of all the elements in T_0. For each node y in P, define the *left side*

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

of y to include all nodes in y's left subtree and y. And define the *right side* of y to include all nodes in y's right subtree. For each node y, label each access s_i in S by whether it is in the left or right side of y, ignoring all accesses not in y's subtree. Denote the number of times the label changes for y as $\mathrm{IB}(S, y)$. The *interleave* bound $\mathrm{IB}(S)$ is $\sum_y \mathrm{IB}(S, y)$.

Theorem 1 (Interleave Lower Bound [6, 22]) $\mathrm{IB}(S)/2 - n$ *is a lower bound on* $\mathrm{OPT}(S, T_0)$.

Demaine et al. observe that it is impossible to use this lower bound to improve the competitive ratio beyond $\Theta(\log \log n)$.

Theorem 2 (Tango is O(log log n)-competitive BST [6]) *The running time of Tango BST on a sequence S of m accesses is* $O(((\mathrm{OPT}(S, T_0)) + n)^*(1 + \log \log n))$.

Applications

Binary search tree (BST) is one of the oldest data structures in the history of computer science. It is frequently used to maintain an ordered set of data. In the last 40 years, many specialized binary search trees have been designed for specific applications. Almost every one of them supports access, insertion, and deletion in worst-case $O(\log n)$ time on average for random sequences of access. This matches the best theoretically possible worst-case bound. For most of these data structures, a random sequence of m accesses will use $\Theta(m \log n)$ time.

While it is impossible to have better asymptotic performance for a random sequence of m accesses, many of the real-world access sequences are not random. For instance, if the set of accesses are randomly drawn from a *small* subset of k element, it's possible to answer all the accesses in $O(m \log k)$ time. A notable binary search tree is splay tree. It is proved to perform well for many access patterns [2, 3, 8, 14, 16–18]. As a result, Sleator and Tarjan [14] conjectured that splay tree is $O(1)$-competitive to the optimal off-line BST. After more than 20 years, the conjecture remains an open problem.

Over the years, several restricted types of optimality have been proved. Many of these restrictions and usage patterns are based on real-world applications. If each access is drawn independently at random from a fixed distribution, D, Knuth [11] constructed a BST based on D that is expected to run in optimal time up to a constant factor. Sleator and Tarjan [14] achieve the same bound without knowing D ahead of time. Other types includes key-independent optimality [10] and BST with free rotations [1].

In 2004, Demaine et al. suggested searching for alternative BST algorithms that have small but nonconstant competitive factors [6]. They proposed *Tango*, the first data structure proved to achieve a nontrivial competitive factor of $O(\log \log n)$. This is a major step toward developing a $O(1)$-competitive BST, and this line of research could potentially replace a large number of specialized BSTs.

Extensions and Promising Research Directions

Following this paper, several new $O(\log \log n)$-competitive BSTs have emerged [9, 21]. A notable example is multi-splay trees [21]. It generalizes the interleave bound to include insertions and deletions. Multi-splay trees also have many theorems analogous to splay trees [20, 21], such as the access lemma and the working set theorem. Wang [21] conjectured that multi-splay trees is $O(1)$-competitive, but it remains an open problem.

Returning to the original motivation for this research, the problem of finding an $o(\log \log n)$-competitive online BST remains open. Several attempts have been made to improve the lower bound [6, 7, 22], but none of them have led to a lower competitive ratio. Even in the off-line model, the problem of finding an $O(1)$-competitive BST is difficult. The best known off-line constant competitive algorithm uses dynamic programming and requires exponential time.

In 2009 Demaine et al. [23] described a *geometric view* of BST algorithms. This is an equivalent model of BST algorithms, but sufficiently

different that it has allowed progress to be made in a number of directions. First of all it has simplified and unified BST lower bounds. It has also allowed progress to be made toward proving a different algorithm to be O(1) competitive. The algorithm is called *GreedyFuture* and was proposed as an off-line algorithm in 1988 by Joan Lucas [24]. After each access, the algorithm restructures the access path according to the future accesses. Specifically if the next access is on this path, then that node is made the new root and the left and right sides are built in a similar fashion recursively. If the next access is to a subtree of that path, then the path node to the left of that subtree is made the root, and the path node to the right of it is made the right child of the root, and the process again continues recursively. A remarkable result of the geometric view is that it shows how the GreedyFuture algorithm can actually be implemented as an online algorithm. At the moment, this seems to be the most likely candidate to be proven to be O(1) competitive.

Cross-References

▸ B-trees

Recommended Reading

Based on Wilber [22]'s lower bound, Tango [6] is the first $O(\log \log n)$-competitive binary search tree. Using many of the ideas in Tango and Link-cut Trees [14, 19], Multi-Splay Trees [21] generalize the competitive framework to include insertion and deletion. The recommended readings are *Self-adjusting binary search trees* by Sleator and Tarjan, *Lower bounds for accessing binary search trees with rotations* by Wilber, *Dynamic Optimality – Almost* by Demaine, et al., and $O(\log \log n)$ *dynamic competitive binary search tree by Wang, et al.*

Recommended Reading

1. Blum A, Chawla S, Kalai A (2003) Static optimality and dynamic search-optimality in lists and trees. Algorithmica 36:249–260
2. Cole R (2000) On the dynamic finger conjecture for splay trees II: the proof. SIAM J Comput 30(1):44–85
3. Cole R, Mishra B, Schmidt J, Siegel A (2000) On the dynamic finger conjecture for splay trees I: splay sorting log n-block sequences. SIAM J Comput 30(1):1–43
4. Crane CA (1972) Linear lists and priority queues as balanced binary trees. Technical report STAN-CS-72-259, Computer Science Department, Stanford University
5. Culik II K, Wood D (1982) A note on some tree similarity measures. Inf Process Lett 15(1): 39–42
6. Demaine ED, Harmon D, Iacono J, Patrascu M (2007) Dynamic optimality-almost. SIAM J Comput 37(1):240–251
7. Derryberry J, Sleator DD, Wang CC (2005) A lower bound framework for binary search trees with rotations. Technical report CMU-CS-05-187, Carnegie Mellon University
8. Elmasry A (2004) On the sequential access theorem and deque conjecture for splay trees. Theor Comput Sci 314(3):459–466
9. Georgakopoulos GF (2005) How to splay for log log n-competitiveness. In: Proceedings of the 4th international workshop on experimental and efficient algorithms (WEA), Santorini Island, pp 570–579
10. Iacono J (2005) Key-independent optimality. Algorithmica 42(1):3–10
11. Knuth DE (1971) Optimum binary search trees. Acta Inf 1:14–25
12. Luccio F, Pagli L (1989) On the upper bound on the rotation distance of binary trees. Inf Process Lett 31(2):57–60
13. Mäkinen E (1988) On the rotation distance of binary trees. Inf Process Lett 26(5):271–272
14. Sleator DD, Tarjan RE (1985) Self-adjusting binary search trees. J ACM 32(3):652–686
15. Sleator DD, Tarjan RE, Thurston WP (1986) Rotation distance, triangulations, and hyperbolic geometry. In: Proceedings 18th ACM symposium on theory of computing (STOC), Berkeley, pp 122–135
16. Sundar R (1989) Twists, turns, cascades, deque conjecture, and scanning theorem. In: Proceedings 30th IEEE symposium on foundations of computer science (FOCS), pp 555–559
17. Sundar R (1992) On the deque conjecture for the splay algorithm. Combinatorica 12(1):95–124
18. Tarjan R (1985) Sequential access in play trees takes linear time. Combinatorica 5(4):367–378
19. Tarjan RE (1983) Data structures and network algorithms. In: CBMS-NSF regional conference series in applied mathematics, vol. 44. SIAM, Philadelphia
20. Wang CC (2006) Multi-splay trees. Ph.D. thesis, Carnegie Mellon University
21. Wang CC, Derryberry J, Sleator DD (2006) $O(\log \log n)$-competitive dynamic binary search trees. In: Proceedings of the 17th annual ACM-SIAM symposium on discrete algorithms (SODA), Miami, pp 374–383

22. Wilber R (1989) Lower bounds for accessing binary search trees with rotations. SIAM J Comput 18(1):56–67
23. Demaine ED, Harmon D, Iacono J, Kane D, Patrascu, M (2009) The Geometry of binary search trees. In: Proceedings of the 20th annual ACM-SIAM symposium on discrete algorithms (SODA), New York, pp 496–505
24. Lucas J (1988) Canonical forms for competitive binary search tree algorithms. Technical report DCS-TR-250, Rutgers University

Oblivious Routing

Nikhil Bansal
Eindhoven University of Technology,
Eindhoven, The Netherlands

Keywords

Fixed path routing

Years and Authors of Summarized Original Work

2002; Räcke

Problem Definition

Consider a communication network, for example, the network of cities across the country connected by communication links. There are several sender-receiver pairs on this network that wish to communicate by sending traffic across the network. The problem deals with routing all the traffic across the network such that no link in the network is overly congested. That is, no link in the network should carry too much traffic relative to its capacity. The obliviousness refers to the requirement that the routes in the network must be designed without the knowledge of the actual traffic demands that arise in the network, i.e., the route for every sender-receiver pair stays fixed irrespective of how much traffic any pair chooses to send. Designing a good oblivious routing strategy is useful since it ensures that the network is robust to changes in the traffic pattern.

Notations

Let $G = (V, E)$ be an undirected graph with nonnegative capacities $c(e)$ on edges $e \in E$. Suppose there are k source-destination pairs (s_i, t_i) for $i = 1, \ldots, k$, and let d_i denote the amount of flow (or demand) that pair i wishes to send from s_i to t_i. Given a routing of these flows on G, the congestion of an edge e is defined as $u(e)/c(e)$, the ratio of the total flow crossing edge e divided by its capacity. The congestion of the overall routing is defined as the maximum congestion over all edges. The congestion minimization problem is to find the routing that minimizes the maximum congestion. Observe that specifying a flow from s_i to t_i is equivalent to finding a probability distribution (not necessarily unique) on a collection of paths from s_i to t_i.

The congestion minimization problem can be studied in many settings. In the offline setting, the instance of the flow problem is provided in advance, and the goal is to find the optimum routing. In the online setting, the demands arrive in an arbitrary adversarial order, and a flow must be specified for a demand immediately upon arrival; this flow is fixed forever and cannot be rerouted later when new demands arrive. Several distributed approaches have also been studied where each pair routes its flow in a distributed manner based on some global information such as the current congestion on the edges.

In this note, the oblivious setting is considered. Here, a routing scheme is specified for each pair of vertices in advance without any knowledge of which demands will actually arrive. Note that an algorithm in the oblivious setting is severely restricted. In particular, if d_i units of demand arrive for pair (s_i, t_i), the algorithm must necessarily route this demand according to the prespecified paths irrespective of the other demands or any other information such as congestion of other edges. Thus, given a network graph G, the oblivious flows need to be computed just once. After this is done, the job of the routing algorithm is trivial; whenever a demand arrives, it simply routes it along the precomputed path. An

oblivious routing scheme is called $c$competitive if for any collection of demands D, the maximum congestion of the oblivious routing is no more than c times the congestion of the optimum offline solution for D. Given this stringent requirement on the quality of oblivious routing, it is not a priori clear that any reasonable oblivious routing scheme should exist at all.

Key Results

Oblivious routing was first studied in the context of permutation routing where the demand pairs form a permutation and have unit value each. It was shown that any oblivious routing that specifies a single path (instead of a flow) between every two vertices must necessarily perform badly. This was first shown by Borodin and Hopcroft [6] for hypercubes, and the argument was later extended to general graphs by Kaklamanis, Krizanc, and Tsantilas [10], who showed the following.

Theorem 1 ([6, 10]) *For every graph G of size n and maximum degree d and every oblivious routing strategy using only a single path for every source-destination pair, there is a permutation that causes an overlap of at least $(n/d)^{1/2}$ paths at some node. Thus, if each edge in G has unit capacity, the edge congestion is at least $(n/d)^{1/2}/d$.*

Since there exists constant degree graphs such as the butterfly graphs that can route any permutation with logarithmic congestion, this implies that such oblivious routing schemes must necessarily perform poorly on certain graphs.

Fortunately, the situation is substantially better if the single path requirement is relaxed and a probability distribution on paths (equivalently a flow) is allowed between each pair of vertices. In a seminal paper, Valiant and Brebner [13] gave the first oblivious permutation routing scheme with low congestion on the hypercube. It is instructive to consider their scheme. Consider an hypercube with $N = 2^n$ vertices. Represent vertex i by the binary expansion of i. For any two vertices s and t, there is a canonical path (of length at most $n = \log N$) from s to t obtained by starting from s and flipping the bits of s in left to right order to match with that of t. Consider routing scheme that for a pair s and t, it first chooses some node p uniformly at random, routes the flow from s to p along the canonical path, and then routes it again from p to t along the canonical path (or equivalently it sends $1/N$ units of flow from s to each intermediate vertex p and then routes it to t). A relatively simple analysis shows that

Theorem 2 ([13]) *The above oblivious routing scheme achieves a congestion of $O(1)$ for hypercubes.*

Subsequently, oblivious routing schemes were proposed for few other special classes of networks. However, the problem of designing oblivious routing schemes for general graphs remained open until recently, when in a breakthrough result Räcke showed the following.

Theorem 3 ([11]) *For any undirected capacitated graph $G = (V, E)$, there exist an oblivious routing scheme with congestion $O(\log^3 n)$ where n is the number of vertices in G.*

The key to Räcke's theorem is a hierarchical decomposition procedure of the underlying graph (described in further detail below). This hierarchical decomposition is a fundamental combinatorial result about the cut structure of graphs and has found several other applications, some of which are mentioned in section "Applications." Räcke's proof of Theorem 3 only showed the existence of a good hierarchical decomposition and did not give an efficient polynomial time algorithm to find it. In subsequent work, Harrelson, Hildrum, and Rao [9] gave a polynomial time procedure to find the decomposition and improved the competitive ratio of the oblivious routing to $O(\log^2 n \log\log n)$.

Theorem 4 ([9]) *There exists an $O(\log^2 n \log\log n)$-competitive oblivious routing scheme for general graphs, and moreover, it can be found in polynomial time.*

Recently, Räcke [12] has given a tight O(log n)-competitive oblivious routing scheme together

with an efficient algorithm to find it. His algorithm is based on an elegant connection to probabilistic embedding of arbitrary metrics into tree metrics.

Interestingly, Azar et al. [4] show that the problem of finding the optimum oblivious routing for a graph can be formulated as a linear program. They consider a formulation with exponentially many constraints, one for each possible demand matrix that has optimum congestion 1 that enforces that the oblivious routing should have low congestion for this demand matrix. Azar et al. [4] give a separation oracle for this problem, and hence, it can be solved using the ellipsoid method. A more practical polynomial size linear program was given later by Applegate and Cohen [2]. Bansal et al. [5] considered a more general variant referred to as the online oblivious routing that can also be used to find an optimum oblivious routing. However, note that without Räcke's result, it would not be clear whether these optimum routings were any good. Moreover, these techniques do not give a hierarchical decomposition and hence may be less desirable in certain contexts. On the other hand, they may be more useful sometimes since they produce an optimum routing (while [9] implies an $O(\log^2 n \log\log n)$-competitive routing for any graph, the best oblivious routing could have a much better guarantee for a specific graph).

Oblivious routing has also been studied for directed graphs; however, the situation is much worse here. Azar et al. [4] show that there exist directed graphs where any oblivious routing is $\Omega(\sqrt{n})$ competitive. Some positive results are also known. Hajiaghayi et al. [7] show a substantially improved guarantee of $O(\log^2 n)$ for directed graphs in the random demands model. Here, each source-sink pair has a distribution (which is known by the algorithm) from which it chooses its demand independently. A relaxation of oblivious routing known as semi-oblivious routing has also been studied recently [8].

Techniques

This section describes the high-level idea of Räcke's result. For a subset $S \subset V$, let $\text{cap}(S)$ denote the total capacity of the edges that cross the cut $(S, V\setminus S)$, and let $\text{dem}(S)$ denote the total demand that must be routed across the cut $(S, V\setminus S)$. Observe that $q = \max_{S\subset V} \text{dem}(S)/\text{cap}(S)$ is a lower bound on the congestion of any solution. On the other hand, the key result [3, 13] relating multicommodity flows and cuts implies that there is a routing such that the maximum congestion is at most $O(q \log k)$ where k is the number of distinct source sink pairs. However, note that this by itself does not suffice to obtain good oblivious routings, since a pair (s_i, t_i) can have different routing for different demand sets. The main idea of Räcke was to impose a treelike structure for routing on the graph to achieve obliviousness. This is formalized by a hierarchical decomposition described below.

Consider a hierarchical decomposition of the graph $G = (V, E)$ as follows. Starting from the set $S = V$, the sets are partitioned successively until each set becomes singleton vertex. This hierarchical decomposition can be viewed naturally as a tree T, where the root corresponds to the set V and leaves corresponds to the singleton sets $\{v\}$. Let S_i denote the subset of V corresponding to node i in T. For an edge (i, j) in the tree where i is the child of j, assign it a capacity equal to $\text{cap}(S_i)$ (note that this is the capacity from S_i to the rest of G and not just capacity between S_i and S_j in G). The tree T is used to simulate routing in G and vice versa. Given a demand from u to v in G, consider the corresponding (unique) route among leaves corresponding to $\{u\}$ and $\{v\}$ in T. For any set of demands, it is easily seen that the congestion in T is no more than the congestion in G. Conversely, Räcke showed that there also exists a tree T where the routes in T can be mapped back to flows in G, such that for any set of demands, the congestion in G is at most $O(\log^3 n)$ times that in T. In this mapping, a flow along the (i, j) in the tree T corresponds to a suitably constructed flow between sets S_i and S_j in G. Since route between any two vertices in T is unique, this gives an oblivious routing in G.

Räcke uses very clever ideas to show the existence of such a hierarchical decomposition. Describing the construction is beyond the scope of this note, but it is instructive to understand the

properties that must be satisfied by such a decomposition. First, the tree T should capture the bottlenecks in G, i.e., if there is a set of demands that produces high congestion in G, then it should also produce a high congestion in T. A natural approach to construct T would be to start with V, split V along a bottleneck (formally, along a cut with low sparsity), and recurse. However, this approach is too simple to work. As discussed below, T must also satisfy two other natural conditions, known as the *bandwidth* property and the *weight* property which are motivated as follows. Consider a node i connected to its parent j in T. Then, i needs to route $\mathrm{dem}(S_i)$ flow out of S_i, and it incurs congestion $\mathrm{dem}(S_i)/\mathrm{cap}(S_i)$ in T. However, when T is mapped back to G, all the flow going out of S_i must pass via S_j. To ensure that the edges from S_i to S_j are not overloaded, it must be the case that the capacity from S_i to S_j is not too small compared to the capacity from S_i to the rest of the graph $V \setminus S_i$. This is referred to as the bandwidth property. Räcke guarantees that this is ratio is always $\Omega(1/\log n)$ for every S_i and S_j corresponding to edges (i, j) in the tree. The weight property is motivated as follows. Consider a node j in T with children $i_1, \ldots, i_p$; then the weight property essentially requires that the sets $S_{i1}, \ldots, S_{ip}$ should be well connected among themselves even when restricted to the subgraph S_j. To see why this is needed, consider any communication between, say, nodes i_1 and i_2 in T. It takes the route i_1 to j to i_2, and hence, in G,S_{i1} cannot use edges that lie outside S_j to communicate with S_{i2}. Räcke shows that these conditions suffice and that a decomposition can be obtained that satisfies them.

The factor $O(\log^3 n)$ in Räcke's guarantee arises from three sources. The first logarithmic factor is due to the flow-cut gap [3, 13]. The second is due to the logarithmic height of the tree, and the third is due to the loss of a logarithmic factor in the bandwidth and weight properties.

Applications

The problem has widespread applications to routing in networks. In practice, it is often required that the routes must be a single path (instead of flows). This can often be achieved by randomized rounding techniques (sometimes under an assumption that the demands to capacity ratios be not too large). The flow formulation provides a much cleaner framework for studying the problems above.

Interestingly, the hierarchical decomposition also found widespread uses in other seemingly unrelated areas such as obtaining good preconditioners for solving systems of linear equations, finding edge-disjoint paths and multicommodity flow problems, online network optimization, speeding up the running time of graph algorithms, and so on.

Cross-References

- Routing
- Separators in Graphs
- Sparsest Cut

Recommended Reading

1. Alon N, Awerbuch B, Azar Y, Buchbinder N, Naor J (2004) A general approach to online network optimization problems. In: Symposium on discrete algorithms, pp 570–579
2. Applegate D, Cohen E (2003) Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. In: SIGCOMM, pp 313–324
3. Aumann Y, Rabani Y (1998) An O(log k) approximate min-cut max-flow theorem and approximation algorithm. SIAM J Comput 27(1):291–301
4. Azar Y, Cohen E, Fiat A, Kaplan H, Räcke H (2003) Optimal oblivious routing in polynomial time. In: Proceedings of the 35th ACM symposium on the theory of computing, pp 383–388
5. Bansal N, Blum A, Chawla S, Meyerson A (2003) Online oblivious routing. In: Symposium on parallelism in algorithms and architectures, pp 44–49
6. Borodin A, Hopcroft J (1985) Routing, merging and sorting on parallel models of computation. J Comput Syst Sci 10(1):130–145
7. Hajiaghayi M, Kim JH, Leighton T, Räcke H (2005) Oblivious routing in directed graphs with random demands. In: Symposium on theory of computing, pp 193–201
8. Hajiaghayi M, Kleinberg R, Leighton T (2007) Semi-oblivious routing: lower bounds. In: Proceedings of

the 18th ACM-SIAM symposium on discrete algorithms, pp 929–938
9. Harrelson C, Hildrum K, Rao S (2003) A polynomial-time tree decomposition to minimize congestion. In: Proceedings of the 15th annual ACM symposium on parallel algorithms and architectures, pp 34–43
10. Kaklamanis C, Krizanc D, Tsantilas T (1991) Tight bounds for oblivious routing in the hypercube. In: Proceedings of the 3rd annual ACM symposium on parallel algorithms and architectures, pp 31–36
11. Räcke H (2002) Minimizing congestion in general networks. In: Proceedings of the 43rd annual symposium on the foundations of computer science, pp 43–52
12. Räcke H (2008) Optimal hierarchical decompositions for congestion minimization networks. In: Symposium on theory of computing, pp 255–264
13. Valiant L, Brebner GJ (1981) Universal schemes for parallel communication. In: Proceedings of the 13th ACM symposium on theory of computing, pp 263–277

Oblivious Subspace Embeddings

Jelani Nelson
Harvard John A. Paulson School of Engineering and Applied Sciences, Cambridge, MA, USA

Keywords

Dimensionality reduction; Oblivious subspace embeddings; Randomized linear algebra

Years and Authors of Summarized Original Work

2006; Sarlós

Problem Definition

This entry surveys some of the applications of "oblivious subspace embeddings," introduced by Sarlós in [19], to problems in linear algebra.

Definition 1 ([19]) Given $0 < \varepsilon < 1/2$ and a d-dimensional subspace $E \subseteq \mathbb{R}^n$, we say an $m \times n$ matrix Π is an *ε-subspace embedding for E* if

$$\forall x \in E \ (1-\varepsilon)\|x\|_2^2 \le \|\Pi x\|_2^2 \le (1+\varepsilon)\|x\|_2^2.$$

The goal is to have m small so that Π provides dimensionality reduction for E.

Given $0 < \varepsilon, \delta < 1/2$, and integers $1 \le d \le n$, an $(\varepsilon, \delta, d, n)$*-oblivious subspace embedding (OSE)* is a distribution $\mathcal{D}$ over $m \times n$ matrices such that for every d-dimensional linear subspace $E \subseteq \mathbb{R}^N$ of dimension d,

$$\mathbb{P}_{\Pi \sim \mathcal{D}}(\Pi \text{ is an } \varepsilon\text{-subspace embedding for } E) > 1-\delta.$$

Sometimes we omit a subset of the variables $\varepsilon, \delta, d, n$ if they are understood from context.

In the definition of an OSE, note that we can write $E = \{Ux : x \in \mathbb{R}^d\}$ where $U \in \mathbb{R}^{n \times d}$ and $U^T U = I$. That is, the columns of U form an orthonormal basis for E. Therefore, we would like that $\|\Pi U x\|_2^2 \approx \|Ux\|_2^2 = x^T U^T U x = \|x\|_2^2$ for all $x \in \mathbb{R}^d$. Letting $\|\cdot\|$ denote operator norm and noting that $\|A\| = \sup_x |x^T A x|$ for any real symmetric A, we see that being an OSE as above is equivalent to the following holding for all $U \in \mathbb{R}^{n \times d}$ with orthonormal columns:

$$\mathbb{P}_{\Pi \sim \mathcal{D}}\left(\|(\Pi U)^T(\Pi U) - I\| > \varepsilon\right) < \delta. \quad (1)$$

Sarlós introduced OSEs [19] to provide faster approximate algorithms for least squares regression and low-rank approximation. In these problems, the input is a tall and skinny matrix $A \in \mathbb{R}^{n \times d}$ ($n \gg d$). For regression, we also are given $b \in \mathbb{R}^n$. The goal is to solve some computational problem given A, and naturally the running time depends on both n and d. The basic idea is to instead run the computation on ΠA for Π sampled from an OSE and (1) prove that the quality of solution found is near optimal if ε is small and (2) enjoy faster computation time to find a solution since the dimensionality of the problem is reduced (ΠA is $m \times d$, whereas A is $n \times d$, $m \ll n$).

Key Results

As mentioned above, Sarlós showed how to use OSEs to speed up least squares regression and low-rank approximation. Below we first discuss constructions of OSEs, and then we elaborate on applications.

Constructing OSEs

One OSE is to pick the entries of $\Pi \in \mathbb{R}^{m \times n}$ i.i.d. from a Gaussian distribution with mean zero and variance $1/m$, where $m = \Theta((d + \log(1/\delta))/\varepsilon^2)$. In fact it suffices to pick any "Johnson-Lindenstrauss transform," i.e., a Π which preserves the Euclidean norms of a certain set of $2^{O(d)}$ vectors up to $1 + \varepsilon$ (see [7]). This setting of m is optimal for any OSE [17]. The downside of such constructions is multiplying ΠA then takes time $\Theta(nmd)$, which is in fact worse than the time to solve the problems considered here.

Sarlós remedied this by picking Π from the "Fast Johnson-Lindenstrauss" distribution [1], which improved this time to $O(nd \log n) + \text{poly}(d)/\varepsilon^2$. A related construction, the "Subsampled Randomized Hadamard Transform" (SRHT), with improved bounds for OSEs was analyzed in [14, 21] using matrix concentration inequalities. In this construction, one chooses $\Pi = \sqrt{n/m} \cdot SHD$ where $D \in \mathbb{R}^{n \times n}$ is diagonal with random signs on the diagonal, H is any bounded orthonormal system that supports matrix-vector multiplication in $O(n \log n)$ time (i.e., H should be orthogonal with $\max_{i,j} |H_{i,j}| = O(1/\sqrt{n})$), and S is a sampling matrix with m rows. That is, the rows of S are independent, and each row of S has exactly a single 1 in a uniformly random location and zeroes elsewhere. The works [14, 21] showed one can take $m = O(d \log(d/\delta)/\varepsilon^2)$. Note we can multiply ΠA in time $O(nd \log n + m)$ by individually multiplying Π by each column of A.

Subsequently, Clarkson and Woodruff [7] showed that the Thorup-Zhang sketch [20] provides an OSE with small m. In particular, consider a random Π with independent columns where in each column there is a single nonzero entry placed in a uniformly random location. The value of this nonzero is uniform in $\{-1, 1\}$. Note that with this construction, one can multiply ΠA in time $O(\text{nnz}(A))$, where $\text{nnz}(\cdot)$ counts nonzero entries. They showed this distribution is an OSE for $m = O(d^2 \log^6(d/\varepsilon)/(\varepsilon^2\delta))$. This bound was improved independently in [15, 16] to $m = O(d^2/(\varepsilon^2\delta))$ via the moment method (see also an observation of Nguyễn [12, Remark 6.4]). Note also a valid OSE is the product of the SRHT with this construction, yielding m as for the SRHT and with multiplication time $O(\text{nnz}(A) + d^3 \log(d/(\varepsilon\delta))/\varepsilon^2)$ to apply to A.

Nelson and Nguyễn [16] analyzed the "Sparse Johnson-Lindenstrauss Transform" (SJLT) of [12] in the context of OSEs. In particular, they showed one can choose an OSE with $m = O(d \log^6(d/\delta)/\varepsilon^2)$ and $s = O(\log^3(d/\delta)/\varepsilon)$ nonzero entries per column. See also [5]. Note ΠA can be computed in time $O(s \cdot \text{nnz}(A))$. One could also choose $m = O(d^{1+\gamma}/\varepsilon^2), s = O(1/\varepsilon)$ for any fixed constant $0 < \gamma < 1$, in which case $\delta = 1/d^c$ for any desired constant $c > 0$. A conjecture of [16] is that $m = O((d + \log(1/\delta))/\varepsilon^2), s = O(\log(d/\delta)/\varepsilon)$ suffices.

Applying OSEs

Least Squares Regression

The input is $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^n$. The goal is to compute

$$x^* = \text{argmin}_{x \in \mathbb{R}^d} \|Ax - b\|_2.$$

By optimality of x^*, Ax^* must be the projection of b onto the column span of A. Write the singular value decomposition (SVD) $A = U\Sigma V^T$, where $U \in \mathbb{R}^{n \times r}, V \in \mathbb{R}^{d \times r}$ have orthonormal columns, and r is the rank of A. Also, $\Sigma \in \mathbb{R}^{r \times r}$ is diagonal with strictly positive entries on the diagonal (the "singular values" of A). Then the column spans of U and of A are identical, and so $Ax^* = UU^T b$ is the desired projection. Thus, we can choose $x^* = V\Sigma^{-1}U^T b$. Alternatively one can write $x^* = (A^T A)^+ A^T b$, where the pseudoinverse of a matrix B with SVD LDW^T is $B^+ = WD^{-1}L^T$.

Simply computing $A^T A$ in the formula for x^* naively takes $O(nd^2)$ time (or $O(nd^{\omega-1})$ if using fast matrix multiplication). Note the following observation.

Observation 1 *Let E be the subspace spanned by b and the columns of A. Let Π be an ε-subspace embedding for E, and write $\tilde{x} = argmin_x \|\Pi Ax - \Pi b\|_2$. Then*

$$\|A\tilde{x} - b\|_2 \leq \sqrt{\frac{1+\varepsilon}{1-\varepsilon}} \cdot \|Ax^* - b\|_2$$

Proof By optimality of $\tilde{x}$, $\|\Pi A\tilde{x} - \Pi b\|_2 \leq \|\Pi Ax^* - \Pi b\|_2$. Furthermore, since $A\tilde{x} - b, Ax^* - b \in E$, we have $(1-\varepsilon)\|A\tilde{x} - b\|_2^2 \leq \|\Pi A\tilde{x} - \Pi b\|_2^2$ and $\|\Pi Ax^* - \Pi b\|_2^2 \leq (1+\varepsilon)\|Ax^* - b\|_2^2$. The claim follows.

The above observation informs us that minimizing $\|\Pi Ax - \Pi b\|_2$ for a subspace embedding Π is sufficient to obtain a high-quality solution to the original problem. Using an OSE with m rows, one can compute $\tilde{x}$ in $O(md^2)$ time (or faster using fast matrix multiplication).

Sarlós also provided another method of using OSEs for least squares regression. First we provide a definition.

Definition 2 We call a distribution $\mathcal{D}$ over $\mathbb{R}^{m\times n}$ an (ε, δ)-AMM_F distribution if, for every pair of matrices A, B each with n rows,

$$\mathbb{P}_{\Pi\sim\mathcal{D}}(\|(\Pi A)^T(\Pi B) - A^T B\| > \varepsilon\|A\|_F\|B\|_F) < \delta$$

where $\|A\|_F = (\sum_{i,j} A_{i,j}^2)^{1/2}$ is the Frobenius norm of A. ("AMM" here stands for "approximate matrix multiplication.")

The work [10] was the first to propose using AMM_F and (non-oblivious) subspace embeddings in the context of low-rank approximation. Sarlós showed that any Johnson-Lindenstrauss (JL) distribution also provides AMM_F, but with δ increased by some factor involving the dimensions of A, B. This factor was removed for random sign matrices in [8] and later for a fairly general class of JL distributions in [12, Theorem 6.2]. We state the relevant definition and theorem for this general class.

Definition 3 ([12]) We say a distribution $\mathcal{D}$ over $\mathbb{R}^{m\times n}$ has (ε, δ, p)-*JL moments* if for any $x \in \mathbb{R}^n$ of unit Euclidean norm,

$$\mathbb{E}_{\Pi\sim\mathcal{D}} \left|\|\Pi x\|_2^2 - 1\right|^p < \varepsilon^p\delta.$$

Theorem 1 ([12]) *Given $\varepsilon, \delta \in (0, 1/2)$, let $\mathcal{D}$ be any distribution with the (ε, δ, p)-JL moment property for some $p \geq 2$. Then $\mathcal{D}$ is a $(3\varepsilon, \delta)$-AMM_F distribution.*

For constant δ, for example, it thus follows from [20] that the Thorup-Zhang sketch with $m = O(1/\varepsilon^2)$ provides AMM_F. Sarlós then proved the following.

Theorem 2 *Suppose $\tilde{x} = argmin\|\Pi Ax - \Pi b\|_2$ where the distribution Π is drawn from is (1) an $(O(1), \delta)$-OSE for d-dimensional subspaces (with a distortion parameter independent of ε), and (2) a $(\sqrt{\varepsilon/d}, \delta)$-$AMM_F$ distribution. Then with probability $1 - 2\delta$,*

$$\|A\tilde{x} - b\|_2 \leq (1 + O(\varepsilon))\|Ax^* - b\|_2.$$

The above theorem combined with [16] allows, for example, picking Π as the SJLT with $m = O(d^{1+\gamma} + d/\varepsilon)$, $s = O(1)$ for any constant $0 < \gamma < 1$ to achieve $(1+\varepsilon)$-multiplicative error for least squares regression.

Low-Rank Approximation

The input is $A \in \mathbb{R}^{n\times d}$ and positive integer k, and the goal is to compute

$$A_k = \underset{B:\text{rank}(B)\leq k}{argmin} \|A - B\|_F.$$

Given the SVD $A = U\Sigma V^T$, the Schmidt approximation theorem (later rediscovered as the Eckart-Young theorem) yields that $A_k = U\Sigma_k V^T$, where Σ_k retains only the k largest elements of Σ and zeroes out the rest. Up to terms logarithmic in dimension and precision, the SVD can be computed in time $nd^{\omega-1}$ [9] where ω is the exponent of square matrix multiplication.

Thus, low-rank approximation can be solved in the same time bound.

A scheme based on OSEs and AMM_F was given by Sarlós. For matrices B, S, let $\mathrm{Proj}_{S,k}(B)$ denote the best rank-k approximation to B in the column span of S. Equivalently, it is the best rank-k approximation to the matrix formed by projecting each column of B to the column span of S. Sarlós' theorem is as follows.

Theorem 3 ([19]) *Let Π be drawn from a distribution which is (1) an $(O(1), \delta)$-OSE for k-dimensional subspaces and (2) a $(\sqrt{\varepsilon/k}, \delta)$-$\mathrm{AMM}_F$ distribution. Then with probability $1-2\delta$,*

$$\|A - \mathrm{Proj}_{A\Pi^T,k}(A)\|_F \leq (1+O(\varepsilon))\|A - A_k\|_F.$$

The above theorem has led to low-rank approximation algorithms with $(1+\varepsilon)$-multiplicative error running in time $O(\mathrm{nnz}(A)) + \tilde{O}(nk^2/\mathrm{poly}(\varepsilon))$ [7,15,16] or even $O(\mathrm{nnz}(A)) + \tilde{O}(nk^{\omega-1}/\mathrm{poly}(\varepsilon))$ [16]. Here $\tilde{O}(\cdot)$ hides logarithmic factors. These algorithms, in these running times, can output a decomposition $L \in \mathbb{R}^{n\times k}, D \in \mathbb{R}^{k\times k}, W \in \mathbb{R}^{d\times k}$ with D diagonal and L, W having orthonormal columns such that

$$\|A - LDW^T\|_F \leq (1+\varepsilon)\|A - A_k\|_F.$$

Other Applications

OSEs have also found other applications, e.g., to approximating leverage scores [11], distributed principle component analysis [13], k-means clustering [6], canonical correlation analysis [3], support vector machines [18], ℓ_p regression [22], ridge regression [14], CUR matrix factorization [4], and streaming approximation of eigenvalues [2]. The reader may investigate these references for more details.

Recommended Reading

1. Ailon N, Chazelle B (2009) The fast Johnson–Lindenstrauss transform and approximate nearest neighbors. SIAM J Comput 39(1):302–322
2. Andoni A, Nguyễn HL (2013) Eigenvalues of a matrix in the streaming model. In: Proceedings of the 24th annual ACM-SIAM symposium on discrete algorithms (SODA), New Orleans, pp 1729–1737
3. Avron H, Boutsidis C, Toledo S, Zouzias A (2013) Efficient dimensionality reduction for canonical correlation analysis. In: Proceedings of the 30th international conference on machine learning (ICML), Atlanta, pp 347–355
4. Boutsidis C, Woodruff DP (2014) Optimal CUR matrix decompositions. In: Proceedings of the 46th ACM symposium on theory of computing (STOC), New York, pp 353–362
5. Bourgain J, Nelson J (2013) Toward a unified theory of sparse dimensionality reduction in Euclidean space. CoRR abs/1311.2542
6. Christos Boutsidis, Anastasios Zouzias, Mahoney MW, Drineas P (2011) Stochastic dimensionality reduction for k-means clustering. CoRR abs/1110.2897
7. Clarkson KL, Woodruff DP (2013) Low rank approximation and regression in input sparsity time. In: Proceedings of the 45th ACM symposium on theory of computing (STOC), Palo Alto, pp 81–90
8. Clarkson KL, Woodruff DP (2009) Numerical linear algebra in the streaming model. In: Proceedings of the 41st ACM symposium on theory of computing (STOC), Bethesda, pp 205–214
9. Demmel J, Dumitriu I, Holtz O (2007) Fast linear algebra is stable. Numer Math 108(1):59–91
10. Drineas P, Mahoney MW, Muthukrishnan S (2006) Subspace sampling and relative-error matrix approximation: column-based methods. In: Proceedings of the 10th international workshop on randomization and computation (RANDOM), Barcelona, pp 316–326
11. Drineas P, Magdon-Ismail M, Mahoney MW, Woodruff DP (2012) Fast approximation of matrix coherence and statistical leverage. J Mach Learn Res 13:3475–3506
12. Kane DM, Nelson J (2014) Sparser Johnson-Lindenstrauss transforms. J ACM 61(1):4
13. Kannan R, Vempala S,Woodruff DP (2014) Principal component analysis and higher correlations for distributed data. In: Proceedings of the 27th annual conference on learning theory (COLT), Barcelona, pp 1040–1057
14. Lu Y, Dhillon P, Foster D, Ungar L (2013) Faster ridge regression via the subsampled randomized Hadamard transform. In: Proceedings of the 26th annual conference on advances in neural information processing systems (NIPS), Lake Tahoe
15. Mahoney MW, Meng X (2013) Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In: Proceedings of the 45th ACM symposium on theory of computing (STOC), Palo Alto, pp 91–100
16. Nelson J, Nguyễn HL (2013) OSNAP: faster numerical linear algebra algorithms via sparser subspace embeddings. In: Proceedings of the 54th annual IEEE symposium on foundations of computer science (FOCS), Berkeley, pp 117–126

17. Nelson J, Nguyễn HL (2014) Lower bounds for oblivious subspace embeddings. In: Proceedings of the 41st international colloquium on automata, languages, and programming, Copenhagen, pp 883–894
18. Paul S, Boutsidis C, Magdon-Ismail M, Drineas P (2013) Random projections for support vector machines. In: Proceedings of the sixteenth international conference on artificial intelligence and statistics (AISTATS), Scottsdale, pp 498–506
19. Sarlós T (2006) Improved approximation algorithms for large matrices via random projections. In: Proceedings of the 47th annual IEEE symposium on foundations of computer science (FOCS), Berkeley, pp 143–152
20. Thorup M, Zhang Y (2012) Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. SIAM J Comput 41(2):293–331
21. Tropp JA (2011) Improved analysis of the subsampled randomized Hadamard transform. Adv Adapt Data Anal 3(1–2, special issue, "Sparse Representations of Data and Images,"):115–126
22. Woodruff DP, Zhang Q (2013) Subspace embeddings and ℓ_p-regression using exponential random variables. In: Proceedings of the 26th annual conference on learning theory (COLT), Princeton, pp 546–567

Obstacle Avoidance Algorithms in Wireless Sensor Networks

Sotiris Nikoletseas[1,2] and Olivier Powell[3]
[1]Computer Engineering and Informatics Department, University of Patras, Patras, Greece
[2]Computer Technology Institute and Press "Diophantus", Patras, Greece
[3]Informatics Department, University of Geneva, Geneva, Switzerland

Keywords

Greedy geographic routing; Routing holes

Years and Authors of Summarized Original Work

2007; Powell, Nikoletseas

Problem Definition

Wireless sensor networks are composed of many small devices called sensor nodes with sensing, computing and radio frequency communication capabilities. Sensor nodes are typically deployed in an ad hoc manner and use their sensors to collect environmental data. The emerging network collectively processes, aggregates and propagates data to regions of interest, e.g., from a region where an event is being detected to a base station or a mobile user. This entry is concerned with the data propagation duty of the sensor network in the presence of obstacles.

For different reasons, including energy conservation and limited transmission range of sensor nodes, information propagation is achieved via multi-hop message transmission, as opposed to single-hop long range transmission. As a consequence, message routing becomes necessary. Routing algorithms are usually situated at the network layer of the protocol stack where the most important component is the (dynamic) communication graph.

Definition 1 (Communication graph) A wireless sensor network is viewed as a graph $G = (V, E)$ where vertexes correspond to sensor nodes and edges represent wireless links between nodes.

Wireless sensor networks have stringent constraints that make classical routing algorithms inefficient, unreliable or even incorrect. Therefore, the specific requirements of wireless sensor networks have to be addressed [2] and geographic routing offers the possibility to design particularly well adapted algorithms.

Geographic Routing

A geographic routing algorithm takes advantage of the fact that sensor nodes are location aware, i.e., they know their position in a coordinate system following the use of a localization protocol [7]. Although likely to introduce a significant overhead, the use of a localization protocol is also likely to be inevitable in many applications where environmental data collected by the sensors

would be useless if not related to some geographical information. For those applications, node location awareness can be assumed to be available for routing purposes at no additional cost.

The Power of Simple Geographic Routing

The early "most forward within range" (MFR) or greedy geographic routing algorithms [14] route messages by maximizing, at each hop, the progress on a projected line towards the destination or, alternatively, minimizing the remaining distance to the message's destination. Both of these greedy heuristics are referred to as *greedy forwarding* (GF). Greedy forwarding is a very appealing routing technique for wireless sensor networks. Among explanations for the attractiveness of GF are the following. (1) GF, as is almost imperatively required, is *fully distributed*. (2) It is *lightweight* in the sense that it induces no topology control overhead. (3) It is *all-to-all* (as opposed to all-to-one). (4) Making no assumptions on the structure of the communication graph, which can be directed, undirected, stable or dynamic (e.g., nodes may be mobile or wireless links may appear and disappear, for example following environmental fluctuation or as a consequence of lower protocol stack layers such as sleep/awake schemes for energy saving purposes), it is *robust*. (5) It is *on-demand*: no routing table or gradient has to be built prior to message propagation. (6) *Efficiency* is featured as messages find short paths to their destination in terms of hop count. (7) It is very *simple* and thus *easy to implement*. (8) It is *memory efficient* in the sense that (8a) the only information stored in the message header is the message's destination and that (8b) it is "ecologically sound" because no "polluting" information is stored on the sensor nodes visited by messages.

Problem Statement

Although very appealing, GF suffers from a major flaw: when a message reaches a local minimum where no further progress towards the destination is possible the routing algorithm fails. There are two major reasons for the occurrence of local minimums: routing holes [1] and obstacles.

Definition 2 The so called *routing holes* are low density regions of the network where no sensor nodes are available for next-hop forwarding.

Even in uniform-randomly deployed networks, routing holes appear as the manifestation of statistical variance of node density. Although increasing as network density diminishes, routing holes have a severe impact on the performance of GF even for very high density networks [12].

Definition 3 A *transmission blocking obstacle* is a region of the network where no sensors are deployed and through which radio signals do not propagate.

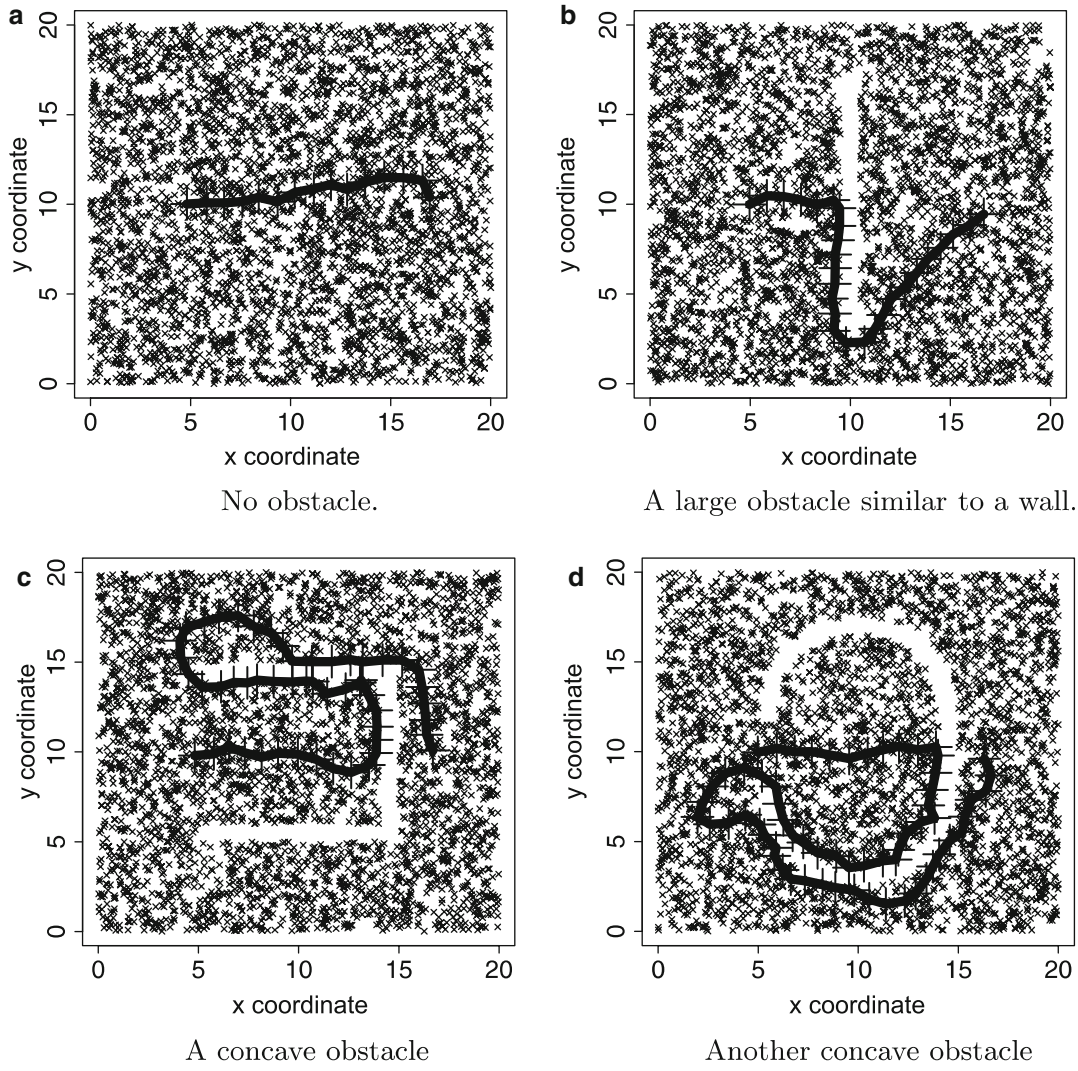
Clearly, large obstacles lying between a message and its destination tend to make GF fail.

The problem reported in this entry is to find a geographic routing algorithm that *maintains the advantages of greedy forwarding* listed in section "Geographic Routing" such as simplicity, light weight, robustness and efficiency while *overcoming its weaknesses*: the inability to escape local minimum nodes created by routing holes and large transmission blocking obstacles such as those seen in Fig. 1.

Problem 1 (Escaping routing holes) The first problem is to route messages out of the many routing holes which are statistically doomed to occur even in dense networks.

Problem 2 (Contouring obstacles) The second problem is to design a protocol capable of routing messages around large transmission blocking obstacles.

Problem 1 can be considered a simplified instance of Problem 2. Lightweight solutions to problem 1 have been previously proposed, usually using limited backtracking [6] or controlled flooding combined with a GF heuristic [4, 13]. However, as shown in [5] where an integrated model for obstacles is proposed and where different algorithms are compared with respect to their obstacle avoidance capability, those solutions do not



Obstacle Avoidance Algorithms in Wireless Sensor Networks, Fig. 1 Typical path followed by GRIC to bypass certain obstacles

satisfactorily solve Problem 2 in the sense that only small and simple obstacles are efficiently bypassed.

Key Results

In [12] a new **g**eographic **r**outi**n**g around obsta**c**les (GRIC) algorithm was proposed to address the problems described in the previous section.

Basic Idea of the Algorithm

In GF, the strategy is to always propagate the message to the neighbor that maximizes progress *towards the destination*. Similarly, GRIC also maximizes progress in a chosen direction. However, this direction is not necessarily the message's destination but an *ideal direction of progress* which has to be computed according to one of two possible strategies: the *inertia mode* or the *rescue mode* described below. Finally, it was found that performance is better in the presence of slightly unstable networks, cf. Result 4,

and thus in the case where the communication graph is very stable it is recommended to use a randomized version of GRIC where nodes about to take a routing decision randomly mark as either passive or active each outbound wireless link of the communication graph. Only active wireless links can be used for message propagation, and link status is re-evaluated each time a new routing decision is taken. Marking links as active with a probability of $p = 0.95$ was found to be a good choice for practical purposes [12].

Inertia Mode

The idea of the inertia mode is that a message should have a strong incentive to go towards its destination but this incentive should be moderated by one to follow the straight ahead direction of current motion *"... like a celestial body in a planet system ..."* [12]. The inertia mode aims at making messages follow closely the perimeter of routing holes and obstacles in order to eventually bypass them and ensure final routing to the destination. To implement the inertia mode, a single additional assumption is made: sensor nodes should be aware of the position of the node from which they receive a message. As an example, this could be done by piggy-backing this 1-hop away routing path history in the message header. Knowing its own position p, the message's destination and the 1-hop away previous position of the message a sensor node can compute the vectors v_{cur} and v_{dst} starting at position p and pointing in the direction of current motion and the direction to the message's destination respectively. The inertia mode defines the ideal direction of progress, v_{idl}, as a vector starting at point p and lying "somewhere in between" v_{cur} and v_{dst}. More precisely, let α be the only angle in $[-\pi, \pi[$ such that v_{dst} is obtained by applying a rotation of angle α to v_{cur}, then v_{idl} is the vector obtained by applying a rotation of angle α' to v_{cur}, where $\alpha' = \text{sign}(\alpha) \cdot \min\left\{\frac{\pi}{6}, |\alpha|\right\}$. Finally, the message is greedily forwarded to the neighbor node maximizing progress in the computed ideal direction of progress v_{idl}.

Rescue Mode

In order to improve overall performance and to bypass complex obstacles, the rescue mode imitates the right-hand rule (RHR) which is a well known wall follower technique to find one's way out of a maze. A high-level description of the RHR component of GRIC is given below while details will be found in [12]. In GRIC, the RHR makes use of a virtual compass and a flag. The virtual compass assigns to v_{cur} a cardinal point value, treating the message's destination as the north. Considering the angle α defined in the previous section, the compass returns a string x-y with x equal to north or south if $|\alpha|$ is smaller or greater than $\frac{\pi}{2}$ respectively, while y is equal to west or east if α is negative or positive respectively. The first time the compass returns a south value, the flag is raised and tagged with the (x, y) value of the compass. Raising the flag means that the message is being routed around an obstacle using the RHR rule if the compass indicates south-west. In the case where the compass indicates south-east, a symmetric case not discussed here for brevity is applied using the left-hand rule (LHR) instead of the RHR. Once the flag is raised, it stays up with its tag unchanged until the compass indicates north, meaning that the obstacle has been bypassed. In fact, a small optimization can be made by lowering the flag only if the compass points to the north-west (in the case of the RHR) and not if it points north-east, but cf. [12] for details. According to the RHR the obstacle's perimeter should be followed closely and kept on the right side of the message's current direction. If ever the compass and the flag's tag disagree, i.e., if the flag is tagged with south-west and the compass returns south-east, it is assumed that the message is turning left too much, that it risks going away from the obstacle and that the RHR is at risk of being violated (a symmetric case applies for the LHR). When this is so, GRIC responds by calling the rescue mode which changes the default way of computing v_{idl}: in rescue mode the message is forced to turn right (or left if the LHR is applied), by defining v_{idl} as the vector obtained by applying to v_{cur} a rotation of angle α'' (instead of α' in inertia mode) where $\alpha'' = -sign(\alpha)(2\pi - |\alpha|)/6$.

O

Main Findings

The performance of GRIC was evaluated through simulations. The main parameters were the presence (or absence) of different shapes of large communication blocking obstacles and the network density which ranged from very low to very high and controls the average degree of the communication graph and the occurrence of routing holes. The main performance metrics were the success rate, i.e., the percentage of messages routed to destination, and the path length. The main findings are that GRIC efficiently, i.e., using short paths, bypasses routing holes and obstacles but that in the presence of hard obstacles, the performance decreases with network density. In Fig. 1. typical routing paths found by GRIC for different obstacle shapes are illustrated, cf. [12] for details on the simulation environment.

Result 1 *In the absence of obstacles, routing holes are bypassed for every network density: The success rate is close to 100 % as long as the source and the destination are connected. Also, routing is efficient in the sense that path lengths are very short.*

Result 2 *Some convex obstacles such as the one in* Fig. 1b *are bypassed with almost 100% success rate and using short paths, even for low densities. When the density gets very low performance diminishes: If the density gets below the critical level guaranteeing the communication graph to be connected with high probability, then the success probability diminishes quickly and successful routings use longer routing paths.*

Result 3 *Some large concave obstacles such as those in* Fig. 1c *and d are efficiently bypassed. However, when facing such obstacles performance becomes more sensitive to network density. The success rate drops and routing paths become longer when the density gets below a certain level depending on the exact obstacle shape.*

Result 4 (Robustness) *Similarly to GF, GRIC is robust to link instability. Furthermore, it was observed that limited link instability has a significantly positive impact on performances. This can be understood as the fact that messages are less likely to enter endless routing loops in a "hot" system than in a "cold" system.*

Applications

Replacement for Greedy Forwarding

Because it makes no compromise with the advantages of GF except the fact that it may be somehow more complicated to implement and because it overcomes GF's main limitations, GRIC can probably replace GF for most routing scenarios including but not exclusively wireless sensor networks. As an example opportunistic-routing strategies [11] could be applied to GRIC rather than to GF.

Wireless Sensor Networks with Large Obstacles

GRIC successfully bypasses large communication blocking obstacles. However, it does so efficiently only if the network density is high enough. This suggests that the obstacle avoidance feature of GRIC may be more useful for dense wireless networks than for sparse networks. Wireless sensor networks are an example of networks which are usually considered to be dense.

Dynamic Networks

There exist some powerful alternatives to GRIC such as the celebrated guaranteed delivery protocols GFG [3], GPSR [8] or GOAFR [10]. Those protocols rely on a planarization phase such as the lazy cross-link detection protocol (CLDP) [9]. LCR implies significant topology maintenance overhead which would be amortized over time if the network is stable enough. On the contrary, if the network is highly dynamic the necessity for frequent updates could make this topology maintenance overhead prohibitive. GRIC may

thus be a preferable choice for dynamic networks where the communication graph is not a stable structure.

Open Problems

(1) Hard concave obstacles such as the one in Fig. 1d are still a challenge for lightweight protocol since in this configuration GRIC's performance is strongly dependent on network density. (2) Low to very low densities are challenging when combined with large obstacles, even when they are "simple" convex obstacles like the one in Fig. 1b. (3) The problem reported in this entry in the case of 3-dimensional networks is open. Inertia may be of some help, however the virtual compass and the right-hand rule seem quite strongly dependant on the 2-dimensional plane. (4) GRIC is not loop free. A mechanism to detect loops or excessively long routing paths would be quite important for practical purposes. (5) The understanding of GRIC could be improved. Analytical results are lacking and new metrics could be considered such as network lifetime, energy consumption or traffic congestion.

Cross-References

▸ Probabilistic Data Forwarding in Wireless Sensor Networks

Recommended Reading

1. Ahmed N, Kanhere SS, Jha S (2005) The holes problem in wireless sensor networks: a survey. SIGMOBILE Mob Comput Commun Rev 9:4–18
2. Al-Karaki JN, Kamal AE (2004) Routing techniques in wireless sensor networks: a survey. Wirel Commun IEEE 11:6–28
3. Bose P, Morin P, Stojmenovic I, Urrutia J (1999) Routing with guaranteed delivery in ad hoc wireless networks. In: Discrete algorithms and methods for mobile computing and communications
4. Chatzigiannakis I, Dimitriou T, Nikoletseas S, Spirakis P (2004) A probabilistic forwarding protocol for efficient data propagation in sensor networks. In: European wireless conference on mobility and wireless systems beyond 3G (EW2004), Barcelona, pp 344–350
5. Chatzigiannakis I, Mylonas G, Nikoletseas S (2006) Modeling and evaluation of the effect of obstacles on the performance of wireless sensor networks. In: 39th ACM/IEEE simulation symposium (ANSS), Los Alamitos. IEEE Computer Society, pp 50–60
6. Chatzigiannakis I, Nikoletseas S, Spirakis P (2005) Smart dust protocols for local detection and propagation. J Mob Netw (MONET) 10:621–635
7. Karl H, Willig A (2005) Protocols and architectures for wireless sensor networks. Wiley, West Sussex
8. Karp B, Kung HT (2000) GPSR: greedy perimeter stateless routing for wireless networks. In: Mobile computing and networking. ACM, New York
9. Kim YJ, Govindan R, Karp B, Shenker S (2006) Lazy cross-link removal for geographic routing. In: Embedded networked sensor systems. ACM, New York
10. Kuhn F, Wattenhofer R, Zhang Y, Zollinger A (2003) Geometric ad-hoc routing: of theory and practice. In: Principles of distributed computing. ACM, New York
11. Lee S, Bhattacharjee B, Banerjee S (2005) Efficient geographic routing in multihop wireless networks. In: MobiHoc'05: proceedings of the 6th ACM international symposium on mobile ad hoc networking and computing. ACM, New York, pp 230–241
12. Powell O, Nikolesteas S (2007) Simple and efficient geographic routing around obstacles for wireless sensor networks. In: WEA 6th workshop on experimental algorithms, Rome. Springer, Berlin
13. Stojmenovic I, Lin X (2001) Loop-free hybrid single-path/flooding routing algorithms with guaranteed delivery for wireless networks. IEEE Trans Parallel Distrib Syst 12:1023–1032
14. Takagi H, Kleinrock L (1984) Optimal transmission ranges for randomly distributed packet radio terminals. IEEE Trans Commun [legacy, pre-1988] 32:46–257

Online Interval Coloring

Leah Epstein
Department of Mathematics, University of Haifa, Haifa, Israel

Keywords

Interval graphs; Online algorithms; Vertex coloring

Years and Authors of Summarized Original Work

1981; Kierstead, Trotter

Problem Definition

Online interval coloring is a graph coloring problem. In such problems the vertices of a graph are presented one by one. Each vertex is presented in turn, along with a list of its edges in the graph, which are incident to previously presented vertices. The goal is to assign colors (which without loss of generality are assumed to be nonnegative integers) to the vertices, so that two vertices which share an edge receive different colors and the total number of colors used (or alternatively, the largest index of any color that is used) is minimized. The smallest number of colors, for which the graph still admits a valid coloring, is called the chromatic number of the graph.

The interval coloring problem is defined as follows. Intervals on the real line are presented one by one, and the online algorithm must assign each interval a color before the next interval arrives, so that no two intersecting intervals receive the same color. The goal is again to minimize the number of colors used to color any interval. The last problem is equivalent to coloring of interval graphs. These are graphs which have a representation (or realization) where each interval represents a vertex and two vertices share an edge if and only if they intersect. It is assumed that the interval graph arrives online together with its realization.

Given an interval graph, denote the size of the largest cardinality clique (complete subgraph) in it by ω. Interval graphs have the special property that in a realization, the set of vertices in a clique have a common point in which they all intersect.

Before discussing the online problem, some properties of interval graphs need to be stated. There exists a simple offline algorithm which produces an optimal coloring of interval graphs. An algorithm applies First Fit, if each time it needs to assign a color to an interval, it assigns a smallest index color which still produces a valid coloring. The optimal algorithm simply considers intervals sorted from left to right by their left end points and applies First Fit. Note that the resulting coloring never uses more than ω colors. Indeed, interval graphs are perfect. A graph G is perfect if any induced subgraph of G, G' (including G), can be colored using $\omega(G')$ colors, where $\omega(G')$ is the size of the largest cardinality clique in G'. (For any graph, ω is a clear lower bound on its chromatic number.)

However, once intervals arrive in an arbitrary order, it is impossible to design an optimal coloring. Consider a simple example where the two intervals $[1, 3]$ and $[6, 8]$ are introduced. If they are colored using two distinct colors, this is already suboptimal, since using the same color for both of them is possible. However, if the sequence of intervals is augmented with $[2, 5]$ and $[4, 7]$, these two new intervals cannot receive the color of the previous intervals or the same color for both new intervals. Thus, three colors are used, even though a valid coloring using two colors can be designed. Note that even if it is known in advance that the input can be colored using exactly two colors, not knowing whether the additional intervals are as defined above, or alternatively, a single interval $[2, 7]$ arrives instead, leads to the usage of three colors instead of only two.

Online coloring is typically hard, which already applies to some simple graph classes such as trees. This is due to the lower bound of $\Omega(\log n)$ (where n is the number of vertices), given by Gyárfás and Lehel [9] on the competitive ratio of online coloring of trees. There are very few classes for which constant bounds are known. One such class is line graphs, for which Bar-Noy, Motwani, and Naor [3] showed that First Fit is 2-competitive (specifically it uses at most $2 \cdot \text{OPT} - 1$ colors, where OPT is the number of colors in an optimal coloring), and this is the best possible bound. This result was later generalized to $k \cdot \text{OPT} - k + 1$ for $(k + 1)$-claw-free graphs by [8] (note that line graphs are 3-claw-free).

Key Results

The paper of Kierstead and Trotter [11] provides a solution to the online interval coloring problem.

They show that the best possible competitive ratio is 3 which is achieved by an algorithm they design. More accurately, the following theorem is proved in the paper.

Theorem 1 *Given an interval graph which is introduced online and presented via its realization, any online algorithm uses at least* $3\omega - 2$ *colors to color the graph, and there exists an algorithm which achieves this bound.*

The algorithm does not need to know ω in advance. Moreover, even though the algorithm is deterministic, it was shown in [13] that the lower bound of 3 on the competitive ratio of online algorithms for interval coloring holds for randomized algorithms as well. Thus, [11] gives a complete solution for the problem.

The main idea of the algorithm is creation of "levels." At the time of arrival of an interval, it is classified into a level as follows. Denote by A_k the union of sets of intervals which currently belong to all levels $1, \ldots, k$. Intervals are classified so that the largest cardinality clique in A_k is of size k. Thus, A_1 is simply a set of non-intersecting intervals. On arrival of an interval, the algorithm finds the smallest k such that the new interval can join level k, without violating the rule above. It can be shown that each level can be colored using two colors by an offline algorithm. Since the algorithm defined here is online, such a coloring cannot be found in general (see example above). However it is shown in [11] that at most three colors are required for each such level, and a coloring using three colors can be found by applying First Fit on each level (with disjoint sets of colors). Moreover, the first level can always be colored using a single color, and ω is equal exactly to the number of levels. Thus a total number of colors, which is at most $3(\omega - 1) + 1 = 3\omega - 2$, is used.

Applications

In this section, both real-world applications of the problem and applications of the methods of Kierstead and Trotter [11] to related problems are discussed.

Many applications arise in various communication networks. The need for connectivity all over the world is rapidly increasing. On the other hand, networks are still composed of very expensive parts. Thus application of optimization algorithms is required in order to save costs.

Consider a network with a line topology that consists of links. Each connection request is for a path between two nodes in the network. The set of requests assigned to a channel must consist of disjoint paths. The goal is to minimize the number of channels (colors) used. A connection request from a to b corresponds to an interval $[a, b]$, and the goal is to minimize the number of required channels to serve all requests.

Another network-related application is that if the requests have constant duration c and all requests have to be served as fast as possible. In this case the colors correspond to time slots, and the total number of colors corresponds to the schedule length. The problem can be described as a scheduling problem as well, and it is clearly of theoretical interest being a natural online graph coloring problem. Two later studies are of possible interest here, both due to their relevance to the original problem and for the usage of related methods.

The applications in networks stated above raise a generalized problem studied in the recent years. In these applications, it is assumed that once a connection request between two points is satisfied, the channel is blocked at least for the duration of this request. An interesting question that was raised by Adamy and Erlebach [1] is the following. Assume that a request consists not only of a requested interval but also from a bandwidth requirement. That is, a customer of a communication channel specifies exactly how much of the channel is needed. Thus, in some cases it is possible to have overlapping requests sharing the same channel. It is required that at every point, the sum of all bandwidth requirements of requests sharing a color cannot exceed the value 1, which is the capacity of the channel. This problem is called *online interval coloring with bandwidth*. In the paper [1], a (large) constant competitive algorithm was designed for the problem. The original interval coloring problem is a special case of this problem

where all bandwidth requests are 1. Note that this problem is a generalization of *bin packing* as well, since bin packing is the special case of the problem where all requests have a common point. Azar et al. [2] designed an algorithm of competitive ratio of at most 10 for this problem. This was done by partitioning the requests into four classes based on their bandwidth requirements and coloring each such class separately. The class of requests with bandwidth in $(\frac{1}{2}, 1]$ was colored using the basic algorithm of [11], since no two such requests colored with one color can overlap. The two other classes, which are $(0, \frac{1}{4}]$ and $(\frac{1}{4}, \frac{1}{2}]$, were colored using adaptations of the algorithm of [11]. Epstein and Levy [6, 7] designed improved lower bounds on the competitive ratio, showing that online interval coloring with bandwidth is harder than online interval coloring.

Another problem related to coloring is the *max coloring* problem [5, 14, 15]. In this problem each interval is given a nonnegative weight. Given a coloring, the weight of a color is the maximum weight of any vertex of this color. The goal is to minimize the sum of weights of the used colors. Note that if all weights are 1, max coloring reduces to the graph coloring problem. Several papers [5, 15], starting with that of Pemmaraju, Raman, and Varadarajan [15], designed algorithms for max coloring that are based on the algorithm of [11] (sometimes as a black box).

Open Problems

Since the paper [11] provided a nice and clean solution to the online interval coloring problem, it does not directly raise open problems. Yet, one related problem is of interest to researchers over the last 30 years, which is the performance of First Fit on this problem. It was shown by Kierstead [10] that First Fit uses at most 40ω colors, thus implying that First Fit has a constant competitive ratio. The quest after the exact competitive ratio was never completed. The best current published results are an upper bound of 10ω by [15] and a lower bound of 4.4ω by Chrobak and Slusarek [4]. See [16] for recent developments. It particular, it is mentioned that a lower bound of $4.99999\omega - C$ (for a fixed $C > 0$) was proved by Kierstead and Trotter in 2004, later improved to a lower bound of $(5-\varepsilon)\omega$ for any $\varepsilon > 0$, implying a lower bound of 5 on the competitive ratio of First Fit [12]. It is interesting to note that for online interval coloring with bandwidth, First Fit has an unbounded competitive ratio [1].

Cross-References

▸ Graph Coloring

Recommended Reading

1. Adamy U, Erlebach T (2003) Online coloring of intervals with bandwidth. In: Proceedings of the first international workshop on approximation and online algorithms (WAOA2003), Budapest, Hungary, pp 1–12
2. Azar Y, Fiat A, Levy M, Narayanaswamy NS (2006) An improved algorithm for online coloring of intervals with bandwidth. Theor Comput Sci 363(1):18–27
3. Bar-Noy A, Motwani R, Naor J (1992) The greedy algorithm is optimal for on-line edge coloring. Inf Process Lett 44(5):251–253
4. Chrobak M, Ślusarek M (1988) On some packing problems relating to dynamical storage allocation. RAIRO J Inf Theory Appl 22:487–499
5. Epstein L, Levin A (2012) On the max coloring problem. Theor Comput Sci 462(1):23–38
6. Epstein L, Levy M (2005) Online interval coloring and variants. In: Proceedings of the 32nd international colloquium on automata, languages and programming (ICALP2005), Lisbon, Portugal, pp 602–613
7. Epstein L, Levy M (2008) Online interval coloring with packing constraints. Theor Comput Sci 407(1–3):203–212
8. Epstein L, Levin A, Woeginger GJ (2011) Graph coloring with rejection. J Comput Syst Sci 77(2):439–447
9. Gyárfás A, Lehel J (1991) Effective on-line coloring of P_5-free graphs. Combinatorica 11(2):181–184
10. Kierstead HA (1988) The linearity of first-fit coloring of interval graphs. SIAM J Discret Math 1(4):526–530
11. Kierstead HA, Trotter WT (1981) An extremal problem in recursive combinatorics. Congr Numer 33:143–153
12. Kierstead HA, Smith DA, Trotter WT (2013) Manuscript. https://math.la.asu.edu/~halk/Publications/biwall6.pdf
13. Leonardi S, Vitaletti A (1998) Randomized lower bounds for online path coloring. In: Proceedings of

the second international workshop on randomization and approximation techniques in computer science (RANDOM'98), Barcelona, Spain, pp 232–247
14. Nonner T (2011) Clique clustering yields a PTAS for max-coloring interval graphs. In: Proceedings of the 38th international colloquium on automata, languages and programming (ICALP2011), Zurich, Switzerland, pp 183–194
15. Pemmaraju S, Raman R, Varadarajan KS (2011) Max-coloring and online coloring with bandwidths on interval graphs. ACM Trans Algorithms 7(3):35
16. Trotter WT (2014) Current research problems: first fit colorings of interval graphs. http://people.math.gatech.edu/~trotter/rprob.html

Online Learning and Optimization

Yaoliang Yu
Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA, USA

Keywords

Convex programming; Hannan consistency; Online learning; Regret; Subgradient descent

Years and Authors of Summarized Original Work

2003; Zinkevich

Problem Definition

Suppose we are going to invest in a stock market. Our neighbor, for mysterious reasons, happens to know how the market evolves. But he cannot change his portfolio (proportions of holding stocks) once committed (to avoid being caught by regulators, say). On the other hand, we, the normal investor, do not have any inside information but can sell and buy at will. If we and our prescient neighbor invest the same amount of money, is there a (computationally feasible) way for us to perform comparably well to our neighbor, without knowing his investing strategy? Surprisingly (as contrary to our real-life experience perhaps), the answer is yes, and we will see it through the lens of online learning. Disclaimer: The reader is at his own risk if he decides to practice the beautiful theoretical results we describe below.

The online learning problem is best described as a multi-round two-person game between the "learner" and the "environment," following the protocol:

The Online Learning Protocol

For $t = 1, \ldots, T$
Learner predicts $x_t \in D$;
Environment responds with a cost function $f_t : D \to \mathbb{R}$;
Learner suffers an immediate cost $f_t(x_t)$;
Learner learns some information of f_t.

Through the multi-round interactions with the environment, the learner tries to learn the behavior of the environment so as to minimize its cumulative cost in the time horizon $t \in [1, T]$, where we could allow the game to continue indefinitely, i.e., $T = \infty$.

The online learning framework is particularly relevant in real applications where (1) *sequential* decisions are needed, (2) *average* good performance is desired, and (3) the process is too *complicated* to be modeled statistically. In our stock example above, the learner will be us (normal stock holder), and the environment will be the market. Each day we submit our portfolio x_t, carefully constructed based on the past information and perhaps also mingled with some randomness (coin tosses for luck). The market responds with rises and falls of the stock prices, represented as the cost function f_t. We suffer the loss $f_t(x_t)$ and learn something about the market (e.g., f_t), and the life moves on to the next day. (If it feels more comfortable, one can negate f and call it *gain*. We shall not do this, because "a true warrior faces her bleak life bravely.") Our adventure ends at day T, which is prefixed. (For $T = \infty$, the adventure never ends.) Of course, the goal is to earn *on average* as much money as possible; it is OK if we lose occasionally. Also, for an average person (us), it is perhaps too

complicated to have a clear idea what is exactly going on in that stock market. As mentioned, we would like to compete against our "prescient neighbor." This is formalized as the regret below.

To evaluate the performance of the learner, the following notion of *regret* plays a central role:

$$\mathsf{R}_T(x) := \sum_{t=1}^{T} \Big(f_t(x_t) - f_t(x) \Big), \ \forall x \in C \subseteq D. \tag{1}$$

Intuitively, the learner compares itself with the baseline (e.g., the "prescient" neighbor) that *constantly* predicts $x \in C$ in each round. We are interested in bounding the learner's regret with respect to the "best" competitor in the set C (although our notation drops the dependence on C):

$$\mathsf{R}_T := \sup_{x \in C} \mathsf{E}(\mathsf{R}_T(x)), \tag{2}$$

where the expectation $\mathsf{E}(\cdot)$ is taken with respect to any internal randomization the learner or the environment might use. The learner is said to be (Hannan) consistent if

$$\frac{\mathsf{R}_T}{T} \to 0, \text{ as } T \to \infty, \text{ i.e., } \mathsf{R}_T = o(T).$$

In other words, the learner performs, on average, as well as the best constant competitor in the long run.

We adopted the notion of regret *not* because we believe a constant (unchanging) predictor is the best strategy for our problem. Instead, the regret should be interpreted as a bare minimum requirement: If there does exist a constant predictor that performs reasonably well on our task, it would be unacceptable if our algorithm is not even on par with it. More often than not, we would like to do *better* than any constant predictor, but this can be highly nontrivial (either computationally or statistically).

We have allowed the learner to operate on a larger set D than its "competitors" (which are restricted to C). Of course this buys the learner some advantage, which sometimes is necessary for consistency, particularly when C is a nonconvex set. For instance, consider the game where the sets $C = D = \{0, 1\}$ and the cost functions

$$f_t(x) = \begin{cases} 1, & \text{if } x = x_t \\ 0, & \text{otherwise} \end{cases}. \tag{3}$$

Recall that in our online learning protocol, we have no control on how the environment reacts. In the very worst case, the environment may appear to be completely "hostile." For instance, the cost function f_t in (3) is thus defined to make the learner always suffer unit cost in each round. On the other hand, the best constant competitor in C suffers cost at most $T/2$ in T rounds. Hence, $\frac{\mathsf{R}_T}{T} \geq \frac{1}{2}$ for all T, meaning that any learner that follows our protocol cannot be consistent. The lesson is, of course, that we cannot compete under a very adversarial environment. However, if we allow the learner to *randomize* its decisions and correspondingly pay *expected* cost, then it is again possible to devise consistent learners for this game [8], provided that the environment is oblivious, i.e., it does not adapt to the learner's randomization, thus constraining its "hostility." Intuitively, randomization and averaging *smooth* out the possible worst-case (but oblivious) reactions of the environment. This is also equivalent to allowing the learner to operate on $D = [0, 1]$, the convex hull of $C = \{0, 1\}$. Indeed, for binary x we can interpret the cost function in (3) as $f_t(x) = |x - y_t|$, where in the worse case the environment could happen to choose $y_t = 1 - x_t$ from the set C. In the randomized setting, the learner first picks $x \in D$, the convex hull of C, and then chooses 1 with probability x and 0 otherwise. Provided that the environment still chooses (however adversarial) $y_t \in C$, the *expected* cost the learner suffers is again $f_t(x) = |x - y_t|$, but this time extended to the convex domain D. The claim that there exists a consistent learner under this randomized setting follows from Theorem 2 below. Intuitively, now the learner sits in the middle ($x = 1/2$) and leans toward the better constant predictor fast enough.

The previous example shows that consistency may not always be achievable. Consequently, the

interesting questions in online learning include (but are not limited to) the following:

- Identifying settings under which consistency can be achieved
- Determining the correct order of the regret tending to infinity
- Devising computationally efficient and order optimal learners

These questions heavily depend on what the learner can learn in each round. For instance, in the full information setting, the learner observes the entire cost function f_t; in the bandit setting, the learner only observes its incurred cost $f_t(x_t)$, while in the partial monitoring setting, the learner only observes some quantity related to its cost. The geometry of the decision set D and the competitor set C, as well as the structural property (such as convexity, smoothness, etc.) of the cost functions, also play a significant role. In the next section, we will consider a special case where a particularly simple algorithm known as online gradient descent suffices to achieve the optimal regret. For more complete and thorough discussions, please refer to the excellent book [3] and surveys [2, 8].

Online Convex Programming

We further simplify our online learning protocol as follows:

Online Convex Programming (on the real line)

- $D \subseteq \mathbb{R}$ is a closed convex set, with $r = \max_{x,y \in D} |x - y| < \infty$;
- $\forall t \leq T$, f_t is convex and differentiable on some open set containing D;
- The gradient is uniformly bounded: $\sup_{x \in D, t \leq T} |\nabla f_t(x)| \leq M < \infty$;
- The learner gets to observe $\nabla f_t(x_t)$ in round t.

The third condition is satisfied if each f_t is M-Lipschitz continuous, i.e.,

$$\forall x, y \in D,\ |f_t(x) - f_t(y)| \leq M \cdot |x - y|, \tag{4}$$

while the last condition is certainly met if the cost function f_t is revealed to the learner in each round. Under this setting, Zinkevich [9] first analyzed the online learner that simply follows the (projected) gradient update:

$$\forall t \geq 1,\ x_{t+1} = \mathsf{P}_D(x_t - \eta_t \nabla f_t(x_t)), \tag{5}$$

where $\eta_t \geq 0$ is a small step size that we determine later and

$$\mathsf{P}_D(x) = \operatorname*{argmin}_{y \in D} |x - y|, \tag{6}$$

is the (Euclidean) projection of x onto the closed set D, i.e., the closest point in D to x. The projection is needed since the learner's prediction x_{t+1} is restricted to the decision set D.

Before we analyze the regret of the above online gradient algorithm, let us first observe that

$$\begin{aligned} \mathsf{R}_T(x) &= \sum_{t=1}^{T} \Big(f_t(x_t) - f_t(x) \Big) \\ &\leq \sum_{t=1}^{T} \Big(\nabla f_t(x_t) \cdot (x_t - x) \Big) \\ &\leq M \sum_{t=1}^{T} |x_t - x|, \end{aligned} \tag{7}$$

where the first inequality follows from the convexity of f_t. Interestingly, the right-hand side is the worst-case regret for the special case where each f_t is a linear function, say, $w_t x_t$ for some $|w_t| \leq M$. In other words, we could have restricted the game to linear cost functions, instead of the seemingly more general convex functions.

The regret of an online learner can be bounded by analyzing its progress with respect to some *potential* function. Here we choose the familiar quadratic potential. Note that for any $x \in C \subseteq D$, clearly $\mathsf{P}_D(x) = x$; hence,

O

$$
\begin{aligned}
|x_{t+1} - x|^2 &= |\mathsf{P}_D(x_t - \eta_t \nabla f_t(x_t)) - \mathsf{P}_D(x)|^2 \\
&\le |x_t - \eta_t \nabla f_t(x_t) - x|^2 \\
&\le |x_t - x|^2 - 2\eta_t \nabla f_t(x_t) \\
&\quad \cdot (x_t - x) + \eta_t^2 M^2 \\
&\le |x_t - x|^2 - 2\eta_t (f_t(x_t) \\
&\quad - f_t(x)) + \eta_t^2 M^2, \qquad (8)
\end{aligned}
$$

where the first inequality follows from the 1-Lipschitz continuity of the projection $\mathsf{P}_D(\cdot)$ and the last inequality is due to the convexity of f_t. Dividing (8) by $2\eta_t$, summing the indices from $t = 1$ to $t = T$, and rearranging, we have

$$
\begin{aligned}
\mathsf{R}_T(x) &= \sum_{t=1}^{T} \Big(f_t(x_t) - f_t(x) \Big) \\
&\le \sum_{t=1}^{T} \frac{1}{2\eta_t} (|x_t - x|^2 - |x_{t+1} - x|^2) \\
&\quad + M^2 \sum_{t=1}^{T} \frac{\eta_t}{2} \qquad (9) \\
&\le \frac{1}{2\eta_1} |x_1 - x|^2 + \sum_{t=2}^{T} \left(\frac{1}{2\eta_t} - \frac{1}{2\eta_{t-1}} \right) \\
&\quad |x_t - x|^2 + M^2 \sum_{t=1}^{T} \frac{\eta_t}{2}. \qquad (10)
\end{aligned}
$$

Setting the step size η_t properly leads to our key results, summarized in the next section.

Key Results

If the horizon T is finite and known in advance, then we can use a constant step size $\eta_t \equiv \eta$. Optimizing with respect to $\eta \ge 0$ from (10) yields

Theorem 1 (e.g., [8, 9]) *Let $\eta_t \equiv \eta = \frac{c}{M\sqrt{T}}$ for some constant $c > 0$; then the online gradient learner achieves sublinear regret*

$$
\mathsf{R}_T(x) \le M\sqrt{T} \frac{c^2 + |x - x_1|^2}{2c}
$$

$$
\le M\sqrt{T} \frac{c^2 + r^2}{2c} \qquad (11)
$$

for the online convex programming problem.

If the horizon is not know in advance, inspired by the step size in Theorem 1, we can try setting $\eta_t = \frac{c}{M\sqrt{t}}$. Note that η_t is decreasing with respect to t. Continuing from (10):

$$
\mathsf{R}_T(x) \le r^2 \left(\frac{1}{2\eta_1} + \sum_{t=2}^{T} \left(\frac{1}{2\eta_t} - \frac{1}{2\eta_{t-1}} \right) \right)
$$

$$
+ cM \sum_{t=1}^{T} \frac{1}{2\sqrt{t}}.
$$

Using integration, $\sum_{t=1}^{T} \frac{1}{2\sqrt{t}} \le \int_0^T \frac{1}{2\sqrt{t}} \mathrm{d}t \le \sqrt{T}$. Thus, we have proved

Theorem 2 (Zinkevich [9]) *Let $\eta_t = \frac{c}{M\sqrt{t}}$ for some constant $c > 0$; then the online gradient learner achieves sublinear regret (simultaneously for all T)*

$$
\mathsf{R}_T(x) \le M\sqrt{T} \frac{2c^2 + r^2}{2c} \qquad (12)
$$

for the online convex programming problem.

Comparing to Theorem 1, we only lose a constant 2 in Theorem 2, but the result now holds simultaneously for all T – a property sometimes called *anytime*. Theorems 1 and 2 not only imply the consistency of the online gradient learner but also demonstrate that $\mathsf{R}_T = \mathrm{O}(\sqrt{T})$, since the right-hand sides of (11) and (12) are independent of the competitor x. In fact, this rate is optimal, i.e., there exists an instantiation where no learner (efficient or not) can do better; see, e.g., [6]. Thanks to the convexity assumption on f_t (and the decision set D), the online gradient algorithm can be efficiently implemented if the gradient ∇f_t and the projection $\mathsf{P}_D(\cdot)$ can be efficiently computed.

Doubling Trick When the horizon T is not known in advance, we can also use the doubling trick, which divides the time into exponentially increasing phases

$$\bigcup_{i=1}^{\lceil \log_2(T+1) \rceil} \{2^{i-1}, \ldots, 2^i - 1\},$$

and on the ith phase, we use the constant step size $\eta_i = \mathrm{O}(1/\sqrt{2^{i-1}})$ suggested in Theorem 1. The overall regret is bounded by

$$\sum_{i=1}^{\lceil \log_2(T+1) \rceil} \mathrm{O}(\sqrt{2^{i-1}}) = \frac{\sqrt{2}}{\sqrt{2}-1} \mathrm{O}(\sqrt{T+1}).$$

So asymptotically we only lose a factor of $\frac{\sqrt{2}}{\sqrt{2}-1} \approx 3.41$.

Other Rates It is possible to tighten the regret rate if the cost functions are more "regular." Intuitively, this means the environment is more constrained hence can only be less adversarial. Indeed, if $f_t - \frac{\sigma}{2}|\cdot|^2$ is convex, namely, f_t is σ-strongly convex, Hazan et al. [6] showed that the online gradient learner equipped with a smaller step size $\eta_t \propto \frac{1}{\sigma t}$ suffers only logarithmic regret $\mathrm{O}(\log(T))$ – an exponential improvement compared to Theorem 2. Just like the time horizon, it is possible to achieve the same logarithmic regret without knowing the parameter σ; see [1]. Similarly, if f_t is so-called exponentially concave, a similar logarithmic regret can be achieved using a second-order Newton-type learner [6].

Extension to High Dimensions The above analysis easily extends to high dimensions. In fact, Theorems 1 and 2 hold in any abstract Hilbert space, with virtually the same proof (provided that we replace the absolute value with the Hilbert norm). The cost functions f_t need not be differentiable either; picking an arbitrary subgradient in the subdifferential $\partial f_t(x_t)$ would suffice.

Extension to Composite Functions The regret can be extended to include a penalty function g as follows:

$$\mathsf{R}_T(x) = \sum_{t=1}^{T} (f_t(x_t) + g(x_t) - f_t(x) - g(x)). \tag{13}$$

Our previous definition in (1) corresponds to the setting where $g(x) = 0$ iff $x \in D$ (otherwise the regret is set to ∞). We could simply treat $f_t + g$ as a whole and apply the online gradient algorithm without any modification. A different approach, resulting in a similar regret bound, upgrades the projection to the proximity operator (of g):

$$\mathsf{P}_g^\eta(x) = \operatorname*{argmin}_y \tfrac{1}{2\eta}|x-y|^2 + g(y), \tag{14}$$

where $\eta > 0$ is the step size to be chosen appropriately. The latter approach is not only more general but also leads to more *structured* intermediate predictions [4]. For instance, if $g(x) = \sum_i |x_i|$ is the ℓ_1 norm, then $[\mathsf{P}_g^\eta(x)]_i = \mathrm{sign}(x_i) \cdot \max\{|x_i| - \eta, 0\}$, which would be exactly zero if $|x_i|$ is small and η is large. In contrast, if we apply online gradient descent directly to $f_t + g$, we would almost never get sparse intermediate predictions.

Without Projections The online gradient learner is computationally efficient only when the projection $\mathsf{P}_D(\cdot)$ in (6) (or more generally the proximity operator in (14)) can be efficiently implemented. In some applications, this is unfortunately not the case. Instead, Hazan and Kale [5] proposed a different learner that bypasses the projection step. Basically, the learner iteratively finds the vertexes of the decision set D and then takes suitable convex combinations of them to make progress.

Connection to Stochastic Optimization The regret bound in Theorem 2 is closely related to some results in stochastic optimization, for the following problem [7]:

$$\inf_{x \in D} f(x), \text{ where } f(x) := \mathsf{E}_\xi(F(x,\xi)), \tag{15}$$

and ξ is some random variable. The stochastic (sub)gradient method is a popular iterative algorithm for optimizing (15). In each iteration, it randomly draws an independent sample ξ_t and follows the projected (sub)gradient update:

$$x_{t+1} = \mathsf{P}_D(x_t - \eta_t \nabla_x F(x_t, \xi_t)),$$

O

for some small step size $\eta_t \geq 0$. The similarity to the online gradient learner is apparent once we identify $f_t(x) := F(x, \xi_t)$. Thus, the regret bound in Theorem 2 implies

$$\begin{aligned} \mathrm{O}\left(\frac{1}{\sqrt{T}}\right) &= \sup_{x \in D} \frac{1}{T} \mathsf{E}\Big[\sum_{t=1}^{T} \big(f_t(x_t) - f_t(x)\big)\Big] \\ &= \sup_{x \in D} \frac{1}{T} \mathsf{E}\Big[\sum_{t=1}^{T} \big(F(x_t, \xi_t) - F(x, \xi_t)\big)\Big] \\ &= \mathsf{E}\left(\frac{1}{T}\sum_{t=1}^{T} f(x_t)\right) - \inf_{x \in D} f(x) \\ &\geq \mathsf{E}\left(f\left(\frac{1}{T}\sum_{t=1}^{T} x_t\right)\right) - \inf_{x \in D} f(x), \end{aligned}$$

provided that the random sample ξ_t is independent of x_t and $F(\cdot, \xi)$ is convex for (almost) every realization of ξ. In other words, the ergodic mean $\frac{1}{T}\sum_{t=1}^{T} x_t$ approaches, in expectation, the infimum in (15) at the rate $\mathrm{O}(1/\sqrt{T})$.

Cross-References

▸ Multi-armed Bandit Problem

Recommended Reading

1. Bartlett PL, Hazan E, Rakhlin A (2007) Adaptive online gradient descent. In: Platt JC, Koller D, Singer Y, Roweis ST (eds) Advances in neural information processing systems 20 (NIPS). Curran Associates, Inc., Vancouver, pp 257–269
2. Bubeck S, Cesa-Bianchi N (2012) Regret analysis of stochastic and nonstochastic multi-armed bandit problems. Found Trends Mach Learn 5(1):1–122
3. Cesa-Bianchi N, Lugosi G (2006) Prediction, learning, and games. Cambridge University Press, New York
4. Duchi JC, Shalev-Shwartz S, Singer Y, Tewari A (2010) Composite objective mirror descent. In: Kalai AT, Mohri M (eds) The 23rd conference on learning theory (COLT). Haifa, pp 14–26
5. Hazan E, Kale S (2012) Projection-free online learning. In: Langford J, Pineau J (eds) The 29th international conference on machine learning (ICML), Edinburgh. Omnipress, pp 521–528
6. Hazan E, Agarwal A, Kale S (2007) Logarithmic regret algorithms for online convex optimization. Mach Learn 69:169–192
7. Nemirovski A, Juditsky A, Lan G, Shapiro A (2009) Robust stochastic approximation approach to stochastic programming. SIAM J Optim 19(4):1574–1609
8. Shalev-Shwartz S (2011) Online learning and online convex optimization. Found Trends Mach Learn 4(2):107–194
9. Zinkevich M (2003) Online convex programming and generalized infinitesimal gradient approach. In: Fawcett T, Mishra N (eds) The 20th international conference on machine learning (ICML), Washington. AAAI Press, pp 928–936

Online List Update

Shahin Kamali
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Keywords

Competitive analysis; Data compression; Online computation; Self-adjusting lists

Years and Authors of Summarized Original Work

1985; Sleator, Tarjan

Problem Definition

List update is one of the classic problems in the context of online computation. The main motivation for the study of the problem is self-adjusting lists. Consider a linear list which represents a dictionary abstract data type. There are three elementary operations in the dictionary, namely, insertion, deletion, and lookup (search). To perform these operations on an item x, an algorithm needs to search for x, i.e., examine the list items, one by one, to find x. For the case of an insertion, all items should be sequentially checked to ensure that the inserted item is not already in the list. A deletion also requires finding the item that is being deleted. In this manner, all operations can be translated into a sequence of lookups or *accesses* to the items in the list. To access an item at index i, an algorithm examines

i items and therefore incurs an access cost of i. Immediately after the access, the algorithm can move the accessed item to any position closer to the front of the list at no extra cost; this is called a *free exchange*. It is also possible to exchange any two consecutive items at a cost of 1 through a *paid exchange*. The objective is to organize the list, using free and paid exchanges, so that the total cost (for accesses and paid exchanges) is minimized.

The list update is naturally an online problem, i.e., at the time of accessing an item, it is not clear what items will be requested in the future. An online algorithm has to take its decision without any knowledge about the forthcoming requests. For example, Move-To-Front (MTF) is a well-known list update algorithm which moves an accessed item to front using a free exchange. In taking its decision, MTF does not rely on any information about future requests. Among other classic list update algorithms, we might mention Transpose (TRANS) and Frequency Count (FC). After accessing an item, TRANS moves it one step closer to the front, i.e., it exchanges the position of x with its preceding item. FC maintains the list in a way that more frequent items appear closer to the front. In doing so, it maintains a counter for each item x which indicates the number of previous requests to x.

Competitive analysis is the standard method for the study and classification of list update algorithms. An algorithm is said to be c-competitive if the cost of serving any request sequence never exceeds c times the optimal cost of an *offline* algorithm OPT which knows the entire sequence in advance. More precisely, an algorithm $\mathcal{A}$ is c-competitive if $\mathcal{A}(\sigma) \leq c\,\mathrm{OPT}(\sigma) + b$ for any sequence σ. Here, $\mathcal{A}(\sigma)$ and $\mathrm{OPT}(\sigma)$ respectively denote the costs of $\mathcal{A}$ and OPT for serving σ, and c and b are constants.

Key Results

List update algorithms were initially studied in regard to their typical behavior on sequences that follow probability distributions. The average cost ratio of an algorithm $\mathcal{A}$ is the ratio between the expected cost of $\mathcal{A}$ for a random sequence and the cost of an optimal offline algorithm which arranges items in nonincreasing order by probability. Under this setting, the ratio achieved by FC is 1 [14], while that of MTF is $\pi/2$ [7]. Moreover, there are distributions in which TIMESTAMP has a better ratio than MTF [14]. These results indicate that FC and TIMESTAMP are better than MTF. However, in practice, MTF has an advantage over the other algorithms. This is partially because the input sequences do not necessarily follow a fixed probability distribution.

In their seminal paper, Sleator and Tarjan proved that MTF is 2-competitive, while TRANS and FC do not achieve a constant competitive ratio [15]. At the same time, for sufficiently long lists, no algorithm can achieve a competitive ratio better than 2. For a while, MTF was the only algorithm with optimal competitive ratio until Albers introduced the TIMESTAMP algorithm [1]. After accessing an item x, TIMESTAMP inserts x in front of the first item y that is before x in the list and is requested at most once since the last request for x. If there is no such item y, or if this is the first access to x, TIMESTAMP does not reorganize the list. TIMESTAMP is also 2-competitive [1].

Randomized Algorithms

No randomized list update algorithm can be better than 2-competitive against adaptive adversaries. However, there are randomized algorithms with better competitive ratios against oblivious adversaries. Reingold et al. introduced a randomized algorithm called BIT which assigns a bit to each item x and initially sets it, uniformly and randomly, to be 0 or 1. At the time of an access to an element x, the bit of x is complemented, and if it becomes 1, the algorithm moves x to the front. BIT has a competitive ratio of 1.75 [13]. Albers et al. proposed a hybrid algorithm, called COMB, which randomly selects between TIMESTAMP and BIT strategies [4]. Upon a request to an item, the algorithm applies BIT strategy with probability 0.8 and TIMESTAMP with probability 0.2. COMB has a competitive ratio of 1.6 [4] which is the best among existing algorithms. Teia proved that no randomized algorithm can be better than 1.5-competitive [16].

Locality of Reference

Real-life sequences usually exhibit locality of reference which implies that the currently requested item is more likely to be requested again. One model of locality is *concave analysis* in which the sequences are consistent with a concave function f so that the number of distinct requests in any window of size τ is at most $f(\tau)$. MTF is the unique optimal solution under *bijective analysis* for sequences that have locality of reference with respect to concave analysis [6]. Bijective analysis is an alternative to competitive analysis that directly compares two algorithms based on their worst-case and average-case behavior (see [6] for details).

Inspired by the concave analysis, Dorrigiv et al. [8] defined the *nonlocality* of a sequence σ of length n, denoted by $\hat{\lambda}(\sigma)$, as $\sum_{i=1}^{n} d_i$ in which d_i is the number of distinct items requested since the last request to the ith item in σ. For the first request to an item, d_i is equal to the length l of the list. For any sequence σ, the cost of any online algorithm is at least $\hat{\lambda}(\sigma)$, while MTF has the same cost of $\hat{\lambda}(\sigma)$. The cost of TIMESTAMP is at least $2\hat{\lambda}(\sigma)$, and TRANS and FC both have a cost of at least $l/2\times\hat{\lambda}(\sigma)$ [8]. These results imply an advantage for MTF when sequences have high locality.

Albers and Lauer defined an alternative locality model which assigns a value $\lambda \in [0,1]$ for each sequence [2]. The larger values for λ imply a higher locality. Using this notion of locality, the competitive ratio of MTF is at most $\frac{2}{1+\lambda}$, i.e., for sequences with high locality, MTF is 1-competitive. The ratio of TIMESTAMP does not improve on request sequences satisfying λ locality, i.e., it remains 2-competitive. The same holds for algorithm COMB, i.e., it remains 1.6-competitive. However, for the algorithm BIT the competitive ratio improves to $\min\{1.75, \frac{2+\lambda}{1+\lambda}\}$.

Applications

As mentioned earlier, the basic application of the list update is in maintaining self-adjusting lists. Martínez and Roura [11], and also Munro [12], observed that a complete rearrangement of items which precede an item at position i is proportional to i rather than i^2. For example, accessing the item at the end of a list and reversing the list can be done in linear time, while under the standard model, it has a quadratic cost. In the MRM model, after an access to an item at index i, the preceding items can be arranged free of charge. It is known that any online algorithm has a competitive ratio of $\Omega(l/\lg l)$ for a list of length l under the MRM model [11, 12], i.e., under this practical setting of the problem, no online algorithm can be competitive (see [9] for details).

List update is widely used for compression purposes. Consider each character of a text as an item in the list and the text as the input sequence. A compression algorithm writes an arbitrary initial configuration in the compressed file, as well as the access costs of a list update algorithm $\mathcal{A}$ for serving each character. In the decompression phase, the algorithm starts from the same initial configuration and follows the steps of the algorithm by reading the access cost written in the compressed file. In order to enhance the performance of the compression schemes, the Burrows-Wheeler Transform (BWT) can be applied to the input string to increase the locality of the input. The bzip2 compression program which applies MTF after BWT transform outperforms the widely used gzip program by more than 5 % on the standard Canterbury corpus.

To theoretically study the list update problem in the context of compression, it is better to assume the cost of accessing an item at index i is $\Theta(\log i)$ rather than $\Theta(i)$. This is because, when an item is accessed in the ith position, the value of i is written as a binary code rather than unary. Sleator and Tarjan show that MTF is 2-competitive if the access cost is a convex function. On the other hand, some algorithms are competitive when the access cost is linear and noncompetitive when the access cost is $\Theta(\log i)$. However, it is not the case for MTF and it has been shown that MTF is 2-competitive when the access cost is logarithmic [10]. In other words,

MTF is useful for compression, even for the sequences (files) generated by an adversary.

Open Problems

The competitive ratio of the best randomized algorithm lies in the range [1.5, 1.6]. Closing this gap is an important direction for future research. Almost all existing algorithms have the *projective property* which informally means that the relative position of any two items in the lists maintained by these algorithms only depend on the requests to these items. Projective algorithms are analyzed under the partial cost model where the cost of accessing an item at index i is $i - 1$. It is known that no online algorithm with the projective property can achieve a competitive ratio better than 1.6 under the partial cost model [5]. Hence, to introduce an algorithm with competitive ratio better than 1.6 of BIT, one needs to deviate from the projective property.

Reingold et al. introduced another model, called *d-paid exchange model*, in which the cost of paid exchanges is scaled up by a value $d \geq 1$, while free exchanges are not allowed [13]. Under this model, no deterministic algorithm can be better than 3-competitive [13], while the best existing algorithm is 4.56-competitive (reported in [3]). For the particular case of $d = 1$, the best lower and upper bound are, respectively, 3 and 4 (MTF is 4-competitive). Closing these gaps is another direction for future research.

Cross-References

- Alternative Performance Measures in Online Algorithms
- Burrows-Wheeler Transform
- Online Paging and Caching

Recommended Reading

1. Albers S (1998) Improved randomized on-line algorithms for the list update problem. SIAM J Comput 27:682–693
2. Albers S, Lauer S (2008) On list update with locality of reference. In: Proceedings of the 35th international colloquium on automata, languages, and programming (ICALP), Reykjavik. Lecture notes in computer science, vol 5125. Springer, pp 96–107
3. Albers S, Westbrook J (1996) Self-organizing data structures. In: Online algorithms: the state of the art, Dagstuhl. Lecture notes in computer science, vol 1442. Springer, pp 13–51
4. Albers S, von Stengel B, Werchner R (1995) A combined BIT and TIMESTAMP algorithm for the list update problem. Inf Process Lett 56(3):135–139
5. Ambühl C, Gärtner B, von Stengel B (2013) Optimal lower bounds for projective list update algorithms. ACM Trans Algorithms 9(4):31
6. Angelopoulos S, Dorrigiv R, López-Ortiz A (2008) List update with locality of reference. In: Proceedings of the 8th Latin American theoretical informatics symposium (LATIN), Búzios. Lecture notes in computer science, vol 4957. Springer, pp 399–410
7. Chung FRK, Hajela DJ, Seymour PD (1988) Self-organizing sequential search and Hilbert's inequality. J Comput Syst Sci 36(2):148–157
8. Dorrigiv R, Ehmsen MR, López-Ortiz A (2009) Parameterized analysis of paging and list update algorithms. In: Proceedings of the 7th international workshop on approximation and online algorithms (WAOA), Copenhagen. Lecture notes in computer science, vol 5893. Springer, pp 104–115
9. Kamali S, López-Ortiz A (2013) A survey of algorithms and models for list update. In: Space-efficient data structures, streams, and algorithms, Waterloo. Lecture notes in computer science, vol 8066. Springer, pp 251–266
10. Kamali S, López-Ortiz A (2014) Better compression through better list update algorithms. In: Proceedings of the 23rd data compression conference (DCC), Snowbird, pp 372–381
11. Martínez C, Roura S (2000) On the competitiveness of the move-to-front rule. Theor Comput Sci 242(1–2):313–325
12. Munro JI (2000) On the competitiveness of linear search. In: Proceedings of the 8th European symposium on algorithms (ESA), Saarbrücken. Lecture notes in computer science, vol 1879. Springer, pp 338–345
13. Reingold N, Westbrook J, Sleator DD (1994) Randomized competitive algorithms for the list update problem. Algorithmica 11:15–32
14. Rivest R (1976) On self-organizing sequential search heuristics. Commun ACM 19:63–67
15. Sleator D, Tarjan RE (1985) Amortized efficiency of list update and paging rules. Commun ACM 28:202–208
16. Teia B (1993) A lower bound for randomized list update algorithms. Inf Process Lett 47:5–9

Online Load Balancing of Temporary Tasks

Leah Epstein
Department of Mathematics, University of Haifa, Haifa, Israel

Keywords

Online scheduling of temporary tasks

Years and Authors of Summarized Original Work

1994; Azar, Broder, Karlin
1997; Azar, Kalyanasundaram, Plotkin, Pruhs, Waarts

Problem Definition

Load balancing of temporary tasks is an online problem. In this problem, arriving tasks (or jobs) are to be assigned to processors, which are also called machines. In this entry, deterministic online load balancing of temporary tasks with unknown duration is discussed. The input sequence consists of departures and arrivals of tasks. If the sequence consists of arrivals only, the tasks are called permanent. Events happen one by one, so that the next event appears after the algorithm completes dealing with the previous event.

Clearly, the problem with temporary tasks is different from the problem with permanent tasks. One such difference is that for permanent tasks, the maximum load is always achieved in the end of the sequence. For temporary tasks, this is not always the case. Moreover, the maximum load may be achieved at different times for different algorithms.

In the most general model, there are m machines $1,\ldots,m$. The information of an arriving job j is a vector p_j of length m, where p_j^i is the load or size of job j if it is assigned to machine i. As stated above, each job is to be assigned to a machine before the next arrival or departure. The load of a machine i at time t is denoted by L_i^t and is the sum of the loads (on machine i) of jobs which are assigned to machine i that arrived by time t and did not depart by this time. The goal is to minimize the maximum load of any machine over all times t. This machine model is known as *unrelated machines* (see [3] for a study of the load-balancing problem of permanent tasks on unrelated machines). Many more specific models were defined. In the sequel, a few such models are described.

For an algorithm $\mathcal{A}$, denote its cost by $\mathcal{A}$ as well. The cost of an optimal offline algorithm that knows the complete sequence of events in advance is denoted by OPT. Load balancing is studied in terms of the (absolute) competitive ratio. The competitive ratio of $\mathcal{A}$ is the infimum $\mathcal{R}$ such that for any input, $\mathcal{A} \leq \mathcal{R} \cdot \mathrm{OPT}$. If the competitive ratio of an online algorithm is at most $\mathcal{C}$, it is also called $\mathcal{C}$-competitive.

Uniformly related machines [3, 12] are machines with speeds associated with them; thus, machine i has speed s_i, and the information that a job j needs to provide upon its arrival is just its size, or the load that it incurs on a unit speed machine, which is denoted by p_j. Then, let $p_j^i = p_j/s_i$. If all speeds are equal, this results in *identical machines* [13].

Restricted assignment [8] is a model where each job may be run only on a subset of the machines. A job j is associated with running time, which is the time to run it on any of its permitted machines M_j. Thus, if $i \in M_j$, then $p_j^i = p_j$, and otherwise, $p_j^i = \infty$.

Key Results

The known results in all four models are surveyed below.

Identical Machines

Interestingly, the well-known algorithm of Graham [13], List Scheduling, which is defined for identical machines, is valid for temporary tasks as well as permanent tasks. This algorithm greedily assigns a new job to the least loaded machine.

The competitive ratio of this algorithm is $2-1/m$, which is best possible (see [5]). Note that the competitive ratio is the same as for permanent tasks, but for permanent tasks, it is possible to achieve a competitive ratio which does not tend to 2 for large m, see, e.g., [11].

Uniformly Related Machines

The situation for uniformly related machines is not very different. In this case, the algorithms of Aspnes et al. [3] and of Berman et al. [12] cannot be applied as they are, and some modifications are required. The algorithm of Azar et al. [7] has competitive ratios of at most 20, and it is based on the general method introduced in [3]. The algorithm of [3] keeps a guess value λ, which is an estimation of the cost of an optimal offline algorithm OPT. An invariant that must be kept is $\lambda \leq 2OPT$. At each step, a procedure is applied for some value of λ (which can be initialized as the load of the first job on the fastest machine). The procedure for a given value of λ is applied until it fails, and some job cannot be assigned while satisfying all conditions. The procedure is designed so that if it fails, then it must be the case that $OPT > \lambda$, the value of λ is doubled, and the procedure is reinvoked for the new value, ignoring all assignments that were done for small values of λ. This method is called doubling and results in an algorithm with a competitive ratio which is at most four times the competitive ratio achieved by the procedure. The procedure for a given λ acts as follows. Let c be a target competitive ratio for the procedure. The machines are sorted according to speed. Each job is assigned to the first machine in the sorted order such that the job is assignable to it. A job j arriving at time t is assignable to machine i if $p_j/s_i \leq \lambda$ and $L_i^{t-1} + p_j/s_i \leq c\lambda$. It is shown in [7] that $c = 5$ allows the algorithm to succeed in the assignment of all jobs (i.e., to have at least one assignable machine for each job) as long as $OPT \leq \lambda$. Note that the constant c for permanent tasks used in [3] is 2. As for lower bounds, it is shown in [7] that the competitive ratio $\mathcal{R}$ of any algorithm satisfies $\mathcal{R} \geq 3 - o(1)$. The upper bound has been improved to $6 + 2\sqrt{5} \approx 10.47$ by Bar-Noy et al. [9].

Restricted Assignment

As for restricted assignment, temporary tasks make this model much more difficult than permanent tasks. The competitive ratio $O(\log m)$ which is achieved by a simple greedy algorithm (see [8]) does not hold in this case. In fact, the competitive ratio of this algorithm becomes $\Omega(m^{\frac{2}{3}})$ [4]. Moreover, in the same paper, a lower bound of $\Omega\sqrt{m}$ on the competitive ratio of any algorithm was shown. The construction was quite involved; however, Ma and Plotkin [14] gave a simplified construction which yields the same result.

The construction of [14] selects a value p, which is the largest integer that satisfies $p + p^2 \leq m$. Clearly, $p = \Theta\left(\sqrt{m}\right)$. The lower bound uses two sets of machines, p machines which are called "the small group" and p^2 machines which are called "the large group." The construction consists of p^2 phases, each of which consists of p jobs and is dedicated to one machine in the large group. In phase i, job k of this phase can run either on the k-th machine of the small group or the i-th machine of the large group. After this arrival, only one of these p jobs does not depart. An optimal offline algorithm assigns all jobs in each phase to the small group except for the one job that will not depart. Thus, when the construction is completed, it has one job on each machine of the large group. The maximum load ever achieved by OPT is 1. However, the algorithm does not know at each phase which job will not depart. If no job is assigned to the small group in phase i, then the load of machine i becomes p. Otherwise, a job that the algorithm assigns to the small group is chosen as the one that will not depart. In this way, after p phases, a total load of p^2 is accumulated on the small group, which means that at least one machine there has load p. This completes the construction.

An alternative algorithm called ROBIN HOOD was designed in [7]. This algorithm keeps a lower bound on OPT, which is the maximum between the following two functions. The first one is the maximum average machine load over time. The second is the maximum job size that has ever arrived. Denote this lower bound at time t (after

O

t events have happened) by B^t. A machine i is called rich at time t if $L_i^t \geq \sqrt{m}B^t$. Otherwise, it is called poor. The windfall time of a rich machine i at time t is the time t' such that i is poor at time $t'-1$ and rich at times $t', \ldots, t$, i.e., the last time that machine i became rich. Clearly, machines can become poor due to an update of B^t or departure of jobs. A machine can become rich due to arrival of jobs that are assigned to it.

The algorithm assigns a job j to a poor machine in $M(j)$ if such a machine exists. Otherwise, j is assigned to the machine in $M(j)$ with the most recent windfall time. The analysis makes use of the fact that at most $\sqrt{m}$ machines can be rich simultaneously.

Note that for small values of m ($m \leq 5$), the competitive ratio of the greedy algorithm is still best possible, as shown in [1]. In this paper, it was shown that these bounds are $(m+3)/2$ for $m = 3, 4, 5$. It is not difficult to see that for $m = 2$, the best bound is 2.

Unrelated Machines

The most extreme difference occurs for unrelated machines. Unlike the case of permanent tasks, where an upper bound of $O(\log m)$ can be achieved [3], it was shown in [2] that any algorithm has a competitive ratio of $\Omega(m/\log m)$. Note that a trivial algorithm, which assigns each job to the machine where it has a minimum load, has a competitive ratio of at most m [3].

Applications

In [10], a hierarchical model was studied. This is a special case of restricted assignment where for each job j, $M(j)$ is a prefix of the machines. They showed that even for temporary tasks, an algorithm of constant competitive ratio exists for this model.

In [6], which studied resource augmentation in load balancing, temporary tasks were considered as well. Resource augmentation is a type of analysis where the online algorithm is compared to an optimal offline algorithm which has less machines.

Open Problems

Small gaps still remain for both uniformly related machines and for unrelated machines. For unrelated machines, it could be interesting to find if there exists an algorithm of competitive ratio $o(m)$ or whether the simple algorithm stated above has optimal competitive ratio (up to a multiplicative factor).

Cross-References

▸ List Scheduling

Recommended Reading

1. Armon A, Azar Y, Epstein L, Regev O (2003) On-line restricted assignment of temporary tasks with unknown durations. Inf Process Lett 85(2):67–72
2. Armon A, Azar Y, Epstein L, Regev O (2003) Temporary tasks assignment resolved. Algorithmica 36(3):295–314
3. Aspnes J, Azar Y, Fiat A, Plotkin S, Waarts O (1997) On-line load balancing with applications to machine scheduling and virtual circuit routing. J ACM 44:486–504
4. Azar Y, Broder AZ, Karlin AR (1994) On-line load balancing. Theor Comput Sci 130:73–84
5. Azar Y, Epstein L (2004) On-line load balancing of temporary tasks on identical machines. SIAM J Discret Math 18(2):347–352
6. Azar Y, Epstein L, van Stee R (2000) Resource augmentation in load balancing. J Sched 3(5):249–258
7. Azar Y, Kalyanasundaram B, Plotkin S, Pruhs K, Waarts O (1997) On-line load balancing of temporary tasks. J Algorithms 22(1):93–110
8. Azar Y, Naor J, Rom R (1995) The competitiveness of on-line assignments. J Algorithms 18:221–237
9. Bar-Noy A, Freund A, Naor J (2000) New algorithms for related machines with temporary jobs. J Sched 3(5):259–272
10. Bar-Noy A, Freund A, Naor J (2001) On-line load balancing in a hierarchical server topology. SIAM J Comput 31:527–549
11. Bartal Y, Fiat A, Karloff H, Vohra R (1995) New algorithms for an ancient scheduling problem. J Comput Syst Sci 51(3):359–366
12. Berman P, Charikar M, Karpinski M (2000) On-line load balancing for related machines. J Algorithms 35:108–121
13. Graham RL (1966) Bounds for certain multiprocessing anomalies. Bell Syst Tech J 45:1563–1581

14. Ma Y, Plotkin S (1997) Improved lower bounds for load balancing of tasks with unknown duration. Inf Process Lett 62:31–34

Online Node-Weighted Problems

Debmalya Panigrahi
Department of Computer Science, Duke University, Durham, NC, USA

Keywords

Network design; Node-weighted graphs; Online algorithms; Primal dual algorithms

Years and Authors of Summarized Original Work

2011; Naor, Panigrahi, Singh
2013; Hajiaghayi, Liaghat, Panigrahi
2014; Hajiaghayi, Liaghat, Panigrahi

Problem Definition

We are given an undirected graph $G = (V, E)$ offline, where node v has a given weight w_v. Initially, the output graph $H \subseteq G$ is the empty graph. In the generic online Steiner network design problem, each online step has a connectivity request C_i and the online algorithm must augment the output graph H to meet the new request. We will consider the following problems in this domain:

- *Steiner tree.* Each connectivity request C_i comprises a new vertex $t_i \in V$ (called a *terminal*) that must be connected in H to all previous terminals. (The first terminal t_0 is often called the *root* and the constraint C_i can then be restated as connecting terminal t_i to the root.)
- *Steiner forest.* Each connectivity request C_i comprises a new vertex pair (s_i, t_i) (called a *terminal pair*) that must be connected in H.
- *Group Steiner tree.* Each connectivity request C_i comprises a new set (group) of vertices $T_i \subseteq V$ (called a *terminal group*). The first terminal group T_0 is a single vertex r called the *root*. At least one vertex in each terminal group must be connected in H to the root.
- *Group Steiner forest.* Each connectivity request C_i comprises a new pair of sets (groups) of vertices (S_i, T_i) (called a *terminal group pair*). For each terminal group pair, at least one vertex in S_i must be connected in H to at least one vertex in T_i.
- *Prize-collecting Steiner tree (resp., Prize-collecting Steiner forest).* Each connectivity request comprises a new terminal t_i (resp., a new terminal pair (s_i, t_i)) and a penalty $\pi_i > 0$; the algorithm must either *pay the penalty* π_i or augment graph H to connect terminal t_i to the root (resp., augment graph H to connect the terminal pair (s_i, t_i)).

In the (group) Steiner tree and (group) Steiner forest problems, the objective is to minimize the total weight (i.e., sum of weights of vertices) of graph H. In the prize-collecting versions of these problems, the objective is to minimize the sum of the total weight of H and the sum of penalties paid by the algorithm.

Key Results

The following theorem was obtained by Naor et al. [7] for the online node-weighted Steiner tree problem.

Theorem 1 *There is a randomized online algorithm for the node-weighted Steiner tree problem that has a competitive ratio of* $O(\log^2 k \log n)$ *and runs in polynomial time.*

This was the first result to obtain a polylogarithmic competitive ratio for the online node-weighted Steiner tree problem. The competitive ratio for this problem was later improved to $O(\log k \log n)$ (see [5]), which is tight up to constants.

The lower bound follows from the observation that the online set cover problem is a special case of the online node-weighted Steiner tree problem. For the online set cover problem, a lower bound of $\Omega\left(\frac{\log m \log n}{\log\log m + \log\log n}\right)$ for deterministic algorithms was obtained by Alon et al. [1], where m is the number of sets and n is the number of elements. This was later improved and extended to a lower bound of $\Omega(\log m \log n)$ for randomized algorithms by Korman [6]. An online set cover instance can be encoded as an online node-weighted Steiner tree instance where the terminals are the elements and the nonterminals are the sets. This encoding yields a lower bound of $\Omega(\log k \log n)$ for the online node-weighted Steiner tree problem and its generalizations discussed below.

In addition to the Steiner tree problem, Naor et al. [7] also considered the online node-weighted Steiner forest problem and the online node-weighted group Steiner tree problem. In fact, they obtained the following theorem for the online node-weighted group Steiner forest problem which generalizes both these problems.

Theorem 2 *There is a randomized online algorithm for the node-weighted group Steiner forest problem that has a competitive ratio polylogarithmic in n and k and runs in quasi-polynomial time.*

For edge-weighted graphs, the same competitive ratio was obtained with a polynomial-time algorithm.

Subsequent to this work, Hajiaghayi et al. [4] investigated the online node-weighted Steiner forest problem and obtained the first polynomial-time algorithm with a polylogarithmic competitive ratio.

Theorem 3 *There is a randomized online algorithm for the node-weighted Steiner forest problem that has a competitive ratio of* $O(\log^2 k \log n)$ *and runs in polynomial time.*

The competitive ratio is tight up to a logarithmic factor owing to the online set cover lower bound described above. For graphs with an excluded minor (such as planar graphs), they gave an improved competitive ratio of $O(\log n)$ for this problem, which is tight up to constants. Moreover, the result can be extended to all $\{0, 1\}$-proper functions which were introduced by Goemans and Williamson [3] to capture a broad range of connectivity problems and extended later to node-weighted graphs by Demaine et al. [2].

For the prize-collecting variants of the online node-weighted Steiner tree and Steiner forest problems, Hajiaghayi et al. [5] gave the first algorithms with a polylogarithmic competitive ratio by showing that these problems can be reduced to the fractional versions of their non prize-collecting variants while losing only a logarithmic factor in the competitive ratio. This led to the following results.

Theorem 4 *There is a randomized online algorithm for the prize-collecting node-weighted Steiner tree problem that has a competitive ratio of* $O(\log k \log^2 n)$. *For the node-weighted prize-collecting Steiner forest problem, there is a randomized online algorithm that has a competitive ratio of* $O(\log^2 k \log^2 n)$. *Both these algorithms run in polynomial time.*

Corresponding results for edge-weighted graphs were previously known [8].

Applications

Online node-weighted Steiner problems have broad applications in designing communication networks where the clientele grows over time.

Open Problems

Suppose we are given a node-weighted undirected graph $G = (V, E)$. In the online edge-connectivity (resp., vertex connectivity) version of the survivable network design problem (SNDP), the online connectivity requirement C_i comprises a pair of terminals (s_i, t_i) and an integer *requirement* $r_i > 0$. The online algorithm must augment the output graph H so that there

are r_i edge-disjoint (resp., node-disjoint) paths between s_i and t_i in H. The objective is to minimize the total weight of H.

An interesting open problem is to obtain an algorithm with competitive ratio $O(r_{\max}^{\alpha} \log^{\beta} n)$ for any constants α, β for the online node-weighted SNDP problem with either edge or vertex connectivity requirements, where $r_{\max} = \max_i r_i$.

Experimental Results

No experimental results are known.

Cross-References

- Generalized Steiner Network
- Steiner Forest
- Steiner Trees

Recommended Reading

1. Alon N, Awerbuch B, Azar Y, Buchbinder N, Naor J (2009) The online set cover problem. SIAM J Comput 39(2):361–370
2. Demaine ED, Hajiaghayi MT, Klein PN (2009) Node-weighted steiner tree and group steiner tree in planar graphs. In: ICALP (1), Rhodes, pp 328–340
3. Goemans MX, Williamson DP (1995) A general approximation technique for constrained forest problems. SIAM J Comput 24(2):296–317
4. Hajiaghayi MT, Liaghat V, Panigrahi D (2013) Online node-weighted steiner forest and extensions via disk paintings. In: FOCS, Berkeley, pp 558–567
5. Hajiaghayi MT, Liaghat V, Panigrahi D (2014) Near-optimal online algorithms for prize-collecting steiner problems. In: ICALP (1), Copenhagen, pp 576–587
6. Korman S (2005) On the use of randomization in the online set cover problem. M.S. thesis, Weizmann Institute of Science
7. Naor J, Panigrahi D, Singh M (2011) Online node-weighted steiner tree and related problems. In: FOCS, Palm Springs, pp 210–219
8. Qian J, Williamson DP (2011) An $O(\log n)$-competitive algorithm for online constrained forest problems. In: ICALP (1), Zurich, pp 37–48

Online Paging and Caching

Neal E. Young
Department of Computer Science and Engineering, University of California, Riverside, CA, USA

Keywords

Caching; Competitive analysis; Competitive ratio; k-server problem; Least-recently-used; Online algorithms; Paging

Years and Authors of Summarized Original Work

1985–2013; multiple authors

Synonyms

Caching; File caching; Paging; Weighted caching; Weighted paging

Problem Definition

A *file-caching* problem instance specifies a cache size k (a positive integer) and a sequence of requests to files, each with a *size* (a positive integer) and a *retrieval cost* (a nonnegative number). The goal is to maintain the cache to satisfy the requests while minimizing the retrieval cost. Specifically, for each request, if the file is not in the cache, one must retrieve it into the cache (paying the retrieval cost) and remove other files to bring the total size of files in the cache to k or less. *Weighted caching* or *weighted paging* is the special case when each file size is 1. *Paging* is the special case when each file size and each retrieval cost is 1 (then the retrieval cost is the number of *cache misses*, and the *fault rate* is the average retrieval cost per request).

An algorithm is *online* if its response to each request is independent of later requests. In practice this is generally necessary. Standard worst-case analysis is not meaningful for online

algorithms – any algorithm will have some input sequence that forces a retrieval for every request. Yet worst-case analysis can be done meaningfully as follows. An algorithm is $c(h,k)$-*competitive* if on *any* sequence σ the total (expected) retrieval cost incurred by the algorithm using a cache of size k is at most $c(h,k)$ times the *minimum* cost to handle σ with a cache of size h (plus a constant independent of σ). Then the algorithm has *competitive ratio* $c(h,k)$. The study of competitive ratios is called *competitive analysis*. (In the larger context of approximation algorithms for combinatorial optimization, this ratio is commonly called the *approximation ratio*.)

Algorithms. Here are definitions of a number of caching algorithms; first is LANDLORD. LANDLORD gives each file "credit" (equal to its cost) when the file is requested and not in cache. When necessary, LANDLORD reduces all cached file's credits proportionally to file size, then evicts files as they run out of credit.

File-caching algorithm LANDLORD
Maintain real value credit$[f]$ with each file f (credit$[f] = 0$ if f is not in the cache).
When a file g is requested:
1. **if** g is not in the cache:
2. **until** the cache has room for g:
3. **for each** cached file f: decrease credit$[f]$ by $\Delta \cdot$ size$[f]$,
4. where $\Delta = \min_{f \in \text{cache}} \text{credit}[f]/\text{size}[f]$.
5. Evict from the cache any subset of the zero-credit files f.
6. Retrieve g into the cache; set credit$[g] \leftarrow$ cost(g).
7. **else** Reset credit$[g]$ anywhere between its current value and cost(g).

For weighted caching, file sizes equal 1. GREEDY DUAL is LANDLORD for this special case. BALANCE is the further special case obtained by leaving credit unchanged in line 7.

For paging, files sizes and costs equal 1. FLUSH-WHEN-FULL is obtained by evicting *all* zero-credit files in line 5; FIRST-IN-FIRST-OUT is obtained by leaving credits unchanged in line 7 and evicting the file that entered the cache earliest in line 5; LEAST-RECENTLY-USED is obtained by raising credits to 1 in line 7 and evicting the least-recently requested file in line 5. The MARKING algorithm is obtained by raising credits to 1 in line 7 and evicting a *random* zero-credit file in line 5. (LANDLORD generalizes to arbitrary covering problems with submodular costs as described in [10].)

Key Results

This entry focuses on competitive analysis of paging and caching strategies as defined above. Competitive analysis has been applied to many problems other than paging and caching, and much is known about other methods of analysis (mainly empirical or average case) of paging and caching strategies, but these are outside scope of this entry.

Paging

In a seminal paper, Sleator and Tarjan showed that LEAST-RECENTLY-USED, FIRST-IN-FIRST-OUT, and FLUSH-WHEN-FULL are $\frac{k}{k-h+1}$-competitive [13]. Sleator and Tarjan also showed that this competitive ratio is the best possible for any deterministic online algorithm. Fiat et al. showed that the MARKING algorithm is $2H_k$-competitive and that no randomized online algorithm is better than H_k-competitive [6]. Here $H_k = 1 + 1/2 + \cdots + 1/k \approx 0.58 + \ln k$. McGeoch and Sleator gave an optimal H_k-competitive randomized online paging algorithm [12].

Weighted Caching

For weighted caching, Chrobak et al. showed that the deterministic online BALANCE algorithm is k-competitive [4]. Young showed that GREEDY DUAL is $\frac{k}{k-h+1}$-competitive and that GREEDY DUAL is a primal-dual algorithm – it generates a solution to the linear-programming dual which proves the near-optimality of the primal solution [14]. Bansal et al., resolving a long-standing open problem, used the primal-dual framework to

give an $O(\log k)$-competitive randomized algorithm for weighted caching [2].

File Caching

When each cost equals 1 (the goal is to minimize the *number* of retrievals), or when each file's cost equals the file's size (the goal is to minimize the total number of *bytes* retrieved), Irani gave $O(\log^2 k)$-competitive randomized online algorithms [7].

For general file caching, Irani and Cao showed that a restriction of LANDLORD is k-competitive [3]. Independently, Young showed that LANDLORD is $\frac{k}{k-h+1}$-competitive [15].

Other Theoretical Models

Practical performance can be better than the worst case studied in competitive analysis. Refinements of the model have been proposed to increase realism. Borodin et al. [1], to model locality of reference, proposed the *access-graph* model (see also [8, 9]). Koutsoupias and Papadimitriou proposed the *comparative ratio* (for comparing classes of online algorithms directly) and the *diffuse-adversary model* (where the adversary chooses requests probabilistically subject to restrictions) [11]. Young showed that any $\frac{k}{k-h+1}$-competitive algorithm is also *loosely* $O(1)$-competitive: for any fixed $\varepsilon, \delta > 0$, on any sequence, for all but a δ-fraction of cache sizes k, the algorithm either is $O(1)$-competitive or pays at most ε times the sum of the retrieval costs [15].

Analyses of Deterministic Algorithms

Here is a competitive analysis of GREEDY DUAL for weighted caching.

Theorem 1 GREEDY DUAL *is* $\frac{k}{k-h+1}$*-competitive for weighted caching.*

Proof Here is an amortized analysis (in the spirit of Sleator and Tarjan, Chrobak et al., and Young; see [14] for a different primal-dual analysis). Define potential

$$\Phi = (h-1) \cdot \sum_{f \in \text{GD}} \text{credit}[f] + k \cdot \sum_{f \in \text{OPT}} \Big(\text{cost}(f) - \text{credit}[f]\Big),$$

where GD and OPT denote the current caches of GREEDY DUAL and OPT (the optimal off-line algorithm that manages the cache to minimize the total retrieval cost), respectively. After each request, GREEDY DUAL and OPT take (some subset of) the following steps in order.

OPT evicts a file f: Since $\text{credit}[f] \le \text{cost}(f)$, Φ cannot increase.

OPT retrieves requested file g: OPT pays $\text{cost}(g)$; Φ increases by at most $k\ \text{cost}(g)$.

GREEDY DUAL decreases $\text{credit}[f]$ **for all $f \in$ GD:** The cache is full and the requested file is in OPT but not yet in GD. So $|\text{GD}| = k$ and $|\text{OPT} \cap \text{GD}| \le h-1$. Thus, the total decrease in Φ is $\Delta[(h-1)|\text{GD}| - k\,|\text{OPT} \cap \text{GD}|] \ge \Delta[(h-1)k - k(h-1)] = 0$.

GREEDY DUAL evicts a file f: Since $\text{credit}[f] = 0$, Φ is unchanged.

GREEDY DUAL retrieves requested file g and sets $\text{credit}[g]$ **to** $\text{cost}(g)$**:** GREEDY DUAL pays $c = \text{cost}(g)$. Since g was not in GD but is in OPT, $\text{credit}[g] = 0$ and Φ decreases by $-(h-1)c + k\,c = (k-h+1)c$.

GREEDY DUAL resets $\text{credit}[g]$ **between its current value and** $\text{cost}(g)$**:** Since $g \in$ OPT and $\text{credit}[g]$ only increases, Φ decreases.

So, with each request: (1) when OPT retrieves a file of cost c, Φ increases by at most kc; (2) at no other time does Φ increase; and (3) when GREEDY DUAL retrieves a file of cost c, Φ decreases by at least $(k-h+1)c$. Since initially $\Phi = 0$ and finally $\Phi \ge 0$, it follows that GREEDY DUAL's total cost times $k-h+1$ is at most OPT's cost times k.

Extension to File Caching

Although the proof above easily extends to LANDLORD, it is more informative to analyze LANDLORD via a *general reduction* from file caching to weighted caching:

Corollary 1 LANDLORD *is* $\frac{k}{k-h+1}$*-competitive for file caching.*

Proof Let W be any deterministic c-competitive weighted-caching algorithm. Define file-caching algorithm F_W as follows. Given request sequence σ, F_W simulates W on weighted-caching sequence σ' as follows. For each file f, break f

into size(f) "pieces" $\{f_i\}$ each of size 1 and cost cost(f)/size(f). When f is requested, give a batch $(f_1, f_2, \ldots, f_s)^{N+1}$ of requests for pieces to W. Take N large enough so W has all pieces $\{f_i\}$ cached after the first sN requests of the batch.

Assume that W *respects equivalence*: after each batch, for every file f, all or none of f's pieces are in W's cache. After each batch, make F_W update its cache correspondingly to $\{f : f_i \in \text{cache}(W)\}$. F_W's retrieval cost for σ is at most W's retrieval cost for σ', which is at most c OPT(σ'), which is at most c OPT(σ). Thus, F_W is c-competitive for file caching.

Now, observe that GREEDY DUAL can be made to respect equivalence. When GREEDY DUAL processes a batch of requests $(f_1, f_2, \ldots, f_s)^{N+1}$ resulting in retrievals, for the last s requests, make GREEDY DUAL set credit$[f_i] =$ cost(f_i) = cost$(f)/s$ in line 7. In general, restrict GREEDY DUAL to raise credits of equivalent pieces f_i equally in line 7. After each batch the credits on equivalent pieces f_i will be the same. When GREEDY DUAL evicts a piece f_i, make GREEDY DUAL evict all other equivalent pieces f_j (all will have zero credit).

With these restrictions, GREEDY DUAL respects equivalence. Finally, taking W to be GREEDY DUAL above, F_W is LANDLORD.

Analysis of the Randomized MARKING Algorithm.

Here is a competitive analysis of the MARKING algorithm

Theorem 2 *The* MARKING *algorithm is* $2H_k$*-competitive for paging.*

Proof Given a paging request sequence σ, partition σ into contiguous *phases* as follows. Each phase starts with the request after the end of the previous phase and continues as long as possible subject to the constraint that it should contain requests to at most k distinct pages. (Each phase starts when the algorithm runs out of zero-credit files and reduces all credits to zero.)

Say a request in the phase is *new* if the item requested was not requested in the previous phase. Let m_i denote the number of new requests in the ith phase. During phases $i-1$ and i, $k + m_i$ distinct files are requested. OPT has at most k of these in cache at the start of the $i-1$st phase, so it will retrieve at least m_i of them before the end of the ith phase. So OPT's total cost is at least $\max\{\sum_i m_{2i}, \sum_i m_{2i+1}\} \geq \sum_i m_i/2$.

Say a non-new request is *redundant* if it is to a file with credit 1 and nonredundant otherwise. Each new request costs the MARKING algorithm 1. The jth nonredundant request costs the MARKING algorithm at most $m_i/(k-j+1)$ in expectation because, of the $k-j+1$ files that if requested would be nonredundant, at most m_i are not in the cache (and each is equally likely to be in the cache). Thus, in expectation MARKING pays at most $m_i + \sum_{j=1}^{k-m_i} m_i/(k-j+1) \leq m_i H_k$ for the phase and at most $H_k \sum_i m_i$ total.

Applications

Variants of GREEDY DUAL and LANDLORD have been incorporated into file-caching software such as Squid [5].

Open Problems

None to report.

Experimental Results

For a study of competitive ratios on practical inputs, see, for example, [3, 5, 14].

Cross-References

- Algorithm DC-TREE for k-Servers on Trees
- Alternative Performance Measures in Online Algorithms
- Online List Update
- Price of Anarchy
- Work-Function Algorithm for k-Servers

Recommended Reading

1. Borodin A, Irani S, Raghavan P, Schieber B (1995) Competitive paging with locality of reference. J Comput Syst Sci 50(2):244–258. Elsevier

2. Buchbinder N, Naor J (2009) Online primal-dual algorithms for covering and packing. Math Oper Res 34(2):270–286. INFORMS
3. Cao P, Irani S (1997) Cost-aware WWW proxy caching algorithms. In: USENIX symposium on internet technologies and systems, Monterey, vol 12(97), pp 193–206
4. Chrobak M, Karloff H, Payne T, Vishwanathan S (1991) New results on server problems. SIAM J Discret Math 4(2):172–181
5. Dilley J, Arlitt M, Perret S (1999) Enhancement and validation of Squid's cache replacement policy. Technical report HPL-1999-69, Hewlett-Packard Laboratories, also in 4th International Web Caching Workshop
6. Fiat A, Karp RM, Luby M, McGeoch LA, Sleator DD, Young NE (1991) Competitive paging algorithms. J Algorithms 12:685–699
7. Irani S (2002) Page replacement with multi-size pages and applications to web caching. Algorithmica 33(3):384–409
8. Irani S, Karlin AR, Phillips S (1996) Strongly competitive algorithms for paging with locality of reference. SIAM J Comput 25(3):477–497. SIAM
9. Karlin AR, Phillips SJ, Raghavan P (2000) Markov paging. SIAM J Comput 30(3):906–922
10. Koufogiannakis C, Young NE (2013) Greedy Δ-approximation algorithm for covering with arbitrary constraints and submodular cost. Algorithmica 66(1):113–152
11. Koutsoupias E, Papadimitriou C (2000) Beyond competitive analysis. SIAM J Comput 30(1):300–317
12. McGeoch L, Sleator D (1991) A strongly competitive randomized paging algorithm. Algorithmica 6(6):816–825
13. Sleator D, Tarjan RE (1985) Amortized efficiency of list update and paging rules. Commun ACM 28:202–208
14. Young NE (1994) The k-server dual and loose competitiveness for paging. Algorithmica 11:525–541
15. Young NE (2002) On-line file caching. Algorithmica 33(3):371–383

Online Preemptive Scheduling on Parallel Machines

Jiří Sgall
Computer Science Institute, Charles University, Prague, Czech Republic

Keywords

Makespan; Online scheduling; Preemption

Years and Authors of Summarized Original Work

2009; Ebenlendr, Jawor, Sgall
2010; Ebenlendr
2011; Ebenlendr, Sgall

Problem Definition

We consider an online version of the classical problem of preemptive scheduling on uniformly related machines.

We are given m machines with *speeds* $s_1 \geq s_2 \geq \cdots \geq s_m$ and a sequence of jobs, each described by its *processing time* (length). The actual time needed to process a job with length p on a machine with speed s is p/s. In the *preemptive version*, each job may be divided into several pieces, which can be assigned to different machines in disjoint time slots. (A job may be scheduled in several time slots on the same machine, and there may be times when a partially processed job is not running at all.) The objective is to find a schedule of all jobs in which the *maximal completion time (makespan)* is minimized.

In the *online problem*, jobs arrive one by one and the algorithm needs to assign each incoming job to some time slots on some machines, without any knowledge of the jobs that arrive later. This problem, also known as list scheduling, was first studied by Graham [8] for identical machines (i.e., $s_1 = \cdots = s_m = 1$), without preemption. In the preemptive version, upon the arrival of a job, its complete assignment at all times must be given and the algorithm is not allowed to change this assignment later. In other words, the online nature of the problem is in the order of the input sequence, and it is not related to possible preemptions and the time in the schedule.

Key Results

The main result is an optimal online algorithm RatioStretch for preemptive scheduling on uniformly related machines [4]. RatioStretch

achieves the best possible competitive ratio not only in the general case but also for any number of machines and any particular combination of machine speeds. Although RatioStretch is deterministic, its competitive ratio matches the best competitive ratio of any randomized algorithm. This proves that randomization does not help for this variant of preemptive scheduling.

For any fixed set of speeds, the competitive ratio of the algorithm RatioStretch can be computed by solving a linear program. However, its worst-case value over all speed combinations is not known. Nevertheless, using the fact that there exists an e-competitive randomized algorithm [5], it is possible to conclude that RatioStretch also achieves the ratio of at most $e \approx 2.718$. The best lower bound shows that RatioStretch (and thus any algorithm) is not better than 2.112-competitive, by providing an explicit numerical instance on 200 machines [3].

Key Techniques

The idea of the algorithm RatioStretch is fairly natural. Suppose that the algorithm is given a ratio R which we are trying to achieve. For each arriving job, RatioStretch computes the optimal makespan for jobs that have arrived so far and runs the incoming job as slow as possible so that it finishes at R times the computed optimal makespan. There are many ways of creating such a schedule given the flexibility of preemptions. RatioStretch chooses a particular one based on the notion of a *virtual machine* from [5]. Given a schedule, the ith virtual machine at each time corresponds to the ith fastest real machine that is idle. (In particular, before the first job, the virtual machines are the real machines.) This assignment of the real machines to the virtual machines can vary at different times in the schedule. Due to preemption, a virtual machine can be thought of and used as a single machine with changing speed. The key idea of RatioStretch is to schedule each job on two adjacent virtual machines.

If RatioStretch fails on some input for a given R, it is possible to use the lower bound technique from [7] and show that there is no R-competitive algorithm. This implies that if the algorithm knows the optimal competitive ratio R, it never fails and thus it is R-competitive.

It remains to find the optimal competitive ratio R. Since the lower bound technique from [7] results in a linear condition, one can show that R can be computed by a linear program for each combination of speeds.

Semi-online Scheduling

The algorithm RatioStretch can be extended to *semi-online* scenarios [6]. This term encompasses situations where some partial information about the input is given to the scheduler in advance. Already Graham [9] studied a semi-online variant of scheduling on identical machines: he proved that if the jobs are presented in non-increasing order of their processing times, the approximation ratio of list scheduling decreases from 2 to $4/3$. Since then numerous semi-online models of scheduling have been studied; typical examples include (sequences of) jobs with decreasing processing times, jobs with bounded processing times, sequences with known total processing time of jobs, and so on. Most of these models can be viewed as online algorithms on a restricted set of input sequences.

RatioStretch can be generalized so that it is optimal for any chosen semi-online restriction. This means not only the cases listed above – the restriction can be given as an arbitrary set of sequences that are allowed as inputs. Again, for any semi-online restriction, RatioStretch achieves the best possible approximation ratio for any number of machines and any particular combination of machine speeds; it is deterministic, but its approximation ratio matches the best possible approximation ratio of any randomized algorithm. This result also provides a clear separation between the design of the algorithm and the analysis of the optimal approximation ratio. While the algorithm is always the same, the analysis of the optimal ratio depends on the studied restrictions.

For typical semi-online restrictions, the optimal ratio can again be computed by linear programs (with machine speeds as parameters). Then we can study the relations between the optimal approximation ratios for different semi-online

restrictions and give some bounds for a large number of machines by analysis of these linear programs. One interesting result is that the overall ratio with known sum of processing times is the same as in the purely online case – even though for a small fixed number of machines, knowing the sum provides a significant advantage.

Some basic restrictions form an inclusion chain: the inputs where the first job has the maximal processing time (which is equivalent to known maximal processing time) include the inputs with non-increasing processing times, which in turn include the inputs with all jobs of equal processing time. The restriction to non-increasing processing times gives the same approximation ratio as when all jobs have equal processing times, even for any particular combination of speeds. The overall approximation ratio of these two equivalent problems is at most 1.52. For known maximal processing time of a job, there exists a computer-generated hard instance with approximation ratio 1.88 with 120 machines. Thus, restricting the jobs to be non-increasing helps the algorithm much more than just knowing the maximal processing time of a job. This is very different from identical machines, where knowing the maximal processing time is equally powerful as knowing that all the jobs are equal; see [10].

Small Number of Machines

For two, three, and sometimes four machines, it is possible to give an exact formula for the competitive ratio for any speed combination [2,3]. This is a fairly routine task which can be simplified (but not completely automated) using standard mathematical software. Once the solution is known, verification amounts to checking the given primal and dual solutions for the linear program.

Open Problems

The main remaining open problem is to develop techniques for determining or bounding the overall competitive ratio of the optimal algorithm RatioStretch. In particular, it would be interesting to obtain a tight bound in the online case.

It is also open if similar techniques can be used for the non-preemptive problem. In this case, the currently best algorithms were obtained by a doubling approach. This means that a competitive algorithm is designed for the case when the optimum is approximately known in advance, and then, without this knowledge, it is used in phases with geometrically increasing guesses of the optimum. Such an approach probably cannot lead to an optimal algorithm for this type of scheduling problems. The best lower and upper bounds for non-preemptive scheduling on uniformly related machines are 2.438 and 5.828 for deterministic algorithms (see [1]) and 2 and 4.311 for randomized algorithms (see [1, 7]). Thus, it is still open whether randomized algorithms are better than deterministic.

Cross-References

- Approximation Schemes for Makespan Minimization
- List Scheduling
- Online Load Balancing of Temporary Tasks

Recommended Reading

1. Berman P, Charikar M, Karpinski M (2000) On-line load balancing for related machines. J. Algorithms 35:108–121
2. Ebenlendr T (2010) Semi-online preemptive scheduling: study of special cases. In: Proceedings of 8th international conference on parallel processing and applied mathematics (PPAM 2009), part II, Wroclaw. Lecture notes in computer science, vol 6068. Springer, pp 11–20
3. Ebenlendr T (2011) Combinatorial algorithms for online problems: semi-online scheduling on related machines. Ph.D. thesis, Charles University, Prague
4. Ebenlendr T, Jawor W, Sgall J (2009) Preemptive online scheduling: optimal algorithms for all speeds. Algorithmica 53:504–522
5. Ebenlendr T, Sgall J (2004) Optimal and online preemptive scheduling on uniformly related machines. In: Proceedings of 21st symposium on theoretical aspects of computer science (STACS), Montpellier. Lecture notes in computer science, vol 2996. Springer, pp 199–210
6. Ebenlendr T, Sgall J (2011) Semi-online preemptive scheduling: one algorithm for all variants. Theory Comput Syst 48:577–613

O

7. Epstein L, Sgall J (2000) A lower bound for on-line scheduling on uniformly related machines. Oper Res Lett 26:17–22
8. Graham RL (1966) Bounds for certain multiprocessing anomalies. Bell Syst Tech J 45: 1563–1581
9. Graham RL (1969) Bounds on multiprocessing timing anomalies. SIAM J Appl Math 17: 263–269
10. Seiden S, Sgall J, Woeginger GJ (2000) Semi-online scheduling with decreasing job sizes. Oper Res Lett 27:215–221

Optimal Crowdsourcing Contests

Balasubramanian Sivan
Microsoft Research, Redmond, WA, USA

Keywords

All-pay auctions; Bayesian Nash equilibrium; Contests

Years and Authors of Summarized Original Work

2012; Chawla, Hartline, Sivan

Problem Definition

With the ever-increasing reach of the Internet, crowdsourcing contests have become an increasingly convenient alternative for completing tasks, compared to traditional hire-and-pay methods. There are several websites dedicated to providing users a platform for creating their own crowdsourcing contests. For instance, Taskcn.com allows users to post tasks, collect submissions from registered users, and provide a monetary reward to the best submission. The reach of crowdsourcing is far beyond tedious/labor-intensive tasks. Netflix, for instance, issued a million-dollar contest for developing a collaborative filtering algorithm to predict user ratings for films, instead of hiring an in-house research team to develop this. The Indian Government used a crowdsourcing contest to pick a new symbol for its rupee currency.

The Questions

In designing a crowdsourcing contest, a principal, with a preallocated sum of monetary reward in hand, seeks to identify the format of the contest that optimizes the quality of the best submission. For instance, Topcoder.com issues 2/3 of the reward to the best submission and 1/3 of the reward to the second-best submission. Is this the format best suited to optimize the best submission? Or should the entire award be given to the winner? More generally, should the precise division of rewards be even announced prior to the contest, or should they be announced only as a function of the quality of the submissions received? In a different direction, crowdsourcing contests make several people to expend efforts in producing submissions, but often only the best submission is put to use. How much effort is getting "burnt" in this process compared to conventional hire-and-pay?

The Model

Formally, let there be n contestants, and let the monetary reward be normalized to \$1. Contestants enter their submissions which are ranked according to their qualities. Agent i's submission quality p_i is a function of their skill v_i and their effort e_i, given by $p_i = v_i \cdot e_i$. The skill v_i can be interpreted as the rate at which agent i can do useful work. The contest designer can observe only the submission qualities p_i's and not the skills v_i's. However, the distribution F from which the v_i's are drawn (independently) is common knowledge to all contestants and the contest designer. Every contestant's goal is to maximize their utility, namely, their reward minus the effort they expended. If x_i is the probability that agent i gets the reward, then their utility is given by $x_i - e_i = x_i - \frac{p_i}{v_i}$.

We model crowdsourcing contests as all-pay auctions, following the contest architecture literature [4, 6]. In an all-pay auction with n bidders, a seller auctions a good that bidder i values at v_i. The value v_i is private to bidder i, but the distribution F from which v_i's are independently drawn is common knowledge to the seller and the bidders. The seller solicits sealed bids from the agents, and all bidders agree to pay their

bids regardless of which bidder gets the good (corresponding to all contestants losing their effort irrespective of which contestant wins the contest). Which agent gets the good depends on the allocation rule of the auction. Given the rules of the auction, each bidder aims to maximize his utility. If x_i is the probability that agent i receives the good, then agent i maximizes his utility of $v_i x_i - p_i$. Note that this utility is precisely v_i times the utility of a contestant in the crowdsourcing contest defined in the previous paragraph. From agent i's perspective, v_i is just a constant. Thus, the incentives in the contest and the all-pay auction are identical. Thus, designing a contest to maximize the quality of the best submission, namely, maximize $\max_i p_i$, is the same as designing an all-pay auction to maximize the maximum payment. Thus, we have an all-pay auction design problem where the objective is not the traditional one of maximizing revenue $\left(\sum_i p_i\right)$ but requires maximizing $\max_i p_i$.

We assume that the space of possible valuations V is an interval and the density $f(\cdot)$ of the value distribution is nonzero everywhere in this interval.

Bayesian Nash Equilibrium

In an all-pay auction it is not strategic for an agent to bid his true value v: the probability he wins the good is at most 1 (in which case he gets a value v), where he is sure to lose his bid. Thus, agents submit bids smaller than their true value. An agent's bidding function $b_i(\cdot)$ maps their true value to bids. A profile of bidding functions $(b_1(\cdot), \ldots, b_n(\cdot))$ is a Bayesian Nash equilibrium (BNE) if the bidding functions are mutual best responses, i.e., if values are drawn from F and other agents bid according to their bidding functions, agent i weakly prefers following his own bidding strategy $b_i(\cdot)$ over submitting any other bid. For a given outcome $x_i(\mathbf{v})$, let $x_i(v_i) = \mathbf{E}_{\mathbf{v}_{-i}}[x_i(\mathbf{v})]$, and let $p_i(v_i) = \mathbf{E}_{\mathbf{v}_{-i}}[p_i(\mathbf{v})]$.

We will appeal to the following result from [2] that shows that for most of the all-pay auctions that we discuss, there exists a unique Bayesian Nash equilibrium, and it is also symmetric. That is, in the unique BNE, all agents have the same bidding function.

Theorem 1 ([2]) *In the all-pay auction parameterized by a reserve price and a nonincreasing sequence of rewards $a_1, \ldots, a_n$, where the agents whose bids meet the reserve are assigned to the rewards in decreasing order of bids, a symmetric BNE exists and is the unique equilibrium.*

Key Results

We now present the key results concerning the design of optimal crowdsourcing contests (from [3]).

Rank-Based-Reward Contests

Consider the class of contests that predetermine the division of rewards into fractions $a_1, \ldots, a_n$, s.t. $\sum_i a_i = 1$, $a_k \geq a_{k+1}$, and $a_i \geq 0$. That is, agents are ordered by submission qualities and the ith best submission receives a_i fraction of the reward. In this notation, Topcoder's contest will be $a_1 = 2/3$, $a_2 = 1/3$, and $a_k = 0$ for $k > 2$. The first key result is that if the goal is to maximize the maximum payment, the optimal all-pay auction is to award the good completely to the highest bidder. In contest language, the optimal contest is a winner-takes-all contest. Note that by Theorem 1 this contest format has a unique BNE.

Theorem 2 *When the contestant skills are distributed i.i.d., the optimal rank-based-reward contest is a winner-takes-all contest.*

Optimal Symmetric Contest

Is there an even better contest in the larger space of contests? Suppose we allow contests that announce rewards as a function of agents' submission qualities, what is the optimal contest? We focus on the class of symmetric contests and optimize over their symmetric equilibria. For a large class of distributions, including distributions that satisfy the monotone hazard rate property (e.g., uniform, normal, exponential), the optimal auction will turn out to have a unique equilibrium that is also symmetric.

Theorem 3 *When the contestant skills are distributed i.i.d. from a distribution that satisfies the monotone hazard rate condition, the optimal*

symmetric contest is highest-submission-wins contest subject to a minimum submission quality.

Proof (Sketch) We prove this result through an argument that mirrors Myerson's revenue optimal auction argument [8]. Recall that in auction theory terms, we have to prove that the optimal auction is a highest-bidder-wins auction subject to a minimum bid reserve. Writing out the expression for the expected maximum payment and using the characterization of BNE payments in terms of allocation, we realize that the expected maximum payment is just the expected virtual welfare. That is, let $\phi(\cdot)$ be a distribution-dependent transformation that is applied to each agent's value v_i to obtain $\phi(v_i) = v_i F(v_i)^{n-1} - \frac{1-F(v_i)^{n-1}}{nf(v_i)}$. The expected virtual welfare of an outcome is just $\mathbf{E}_{\mathbf{v}}\left[\sum_i \phi(v_i)x_i(\mathbf{v})\right]$. If this is the quantity to maximize, it is immediate that the optimal outcome is to allocate completely to the agent with the highest virtual value subject to the highest virtual value being nonnegative. If the virtual value transformation were a strictly increasing function (whenever it is positive), the bidder with the highest value, and hence also the highest bid because of our focus on symmetric equilibria, will also be the bidder with the highest virtual value. Thus the highest-bidder-wins auction subject to a minimum bid reserve will implement the desired outcome. Now, for the distribution-dependent transformation $\phi(\cdot)$ to be strictly increasing, it is enough for the distribution to satisfy the monotone hazard rate condition. Finally, this contest has a unique BNE from Theorem 1.

Theorem 4 *For any setting with i.i.d. values, the optimal symmetric contest is defined by a minimum submission quality and a subset of submission qualities called forbidden qualities that has the following format: the contest solicits submissions and rounds them down to the nearest non-forbidden quality; it then distributes the reward equally among the highest submissions subject to the submissions being above the minimum submission quality.*

Proof (Sketch) Continuing with the proof of Theorem 3, if the virtual value transformation were not increasing, allocating to the highest virtual value is no more a BNE outcome. In this case, the transformation ϕ is "ironed" to obtain a nondecreasing function $\bar{\phi}(\cdot)$, such that the expected maximum payment is equal to the expected ironed virtual surplus. To optimize this quantity, the outcome should be to allocate completely to the agent with the highest ironed virtual value subject to it being nonnegative. In case of a tie, all agents with the highest ironed virtual value get equal allocations. Such an allocation will result in a discontinuous allocation function and hence a discontinuous payment function. That is, some payments are forbidden. Correspondingly to ensure that some bids are forbidden, the auction explicitly says that bids in certain regions will be rounded down so that no rational agent will bid inside that interval. This explains the format of the optimal contest specified in the theorem.

Utilization Ratio of Crowdsourcing

In a crowdsourcing contest, which is like an all-pay auction, every agent's submission is collected, but only the best submission is used. In contrast, in conventional contracting, which is like first- or second-price auctions, only the winner makes any submission at all, and thus there is no underutilization. One way of measuring the amount of work that actually gets utilized in crowdsourcing as opposed to getting "burnt" is to study the ratio of the maximum payment and the sum of all payments in an all-pay auction. It turns out that the utilization ratio in a large class of contests is at least a $1/2$.

Theorem 5 *In any highest-submission-wins contest with a minimum submission quality, the quality of the best submission is at least half of the sum total of the qualities of all the submissions.*

Related Work

Other objectives that have been studied in contest design include maximizing the sum of submission qualities instead of the maximum submission quality [5–7] and maximizing the sum of submission qualities less the normalized reward [1]. The rank-based-reward result in

Theorem 2 is quite robust and continues to hold in many of these other models as well. Moldovanu and Sela [7] study multi-round contests and show that there are situations where it is better to split contestants into two divisions and to have a final among the divisional winners. DiPalantino and Vojnovic [4] study crowdsourcing websites as a matching market. Yang et al. [9] and DiPalantino and Vojnovic [4] study contestant behavior from contest website Taskcn.com and observe that experienced contestants strategize well.

Open Problems

Multi-round Contests

The optimality result discussed here is restricted to single-round contests. If one were allowed to do a tournament-style multi-round contest, what is the optimal contest in this large class of contests? How significant is the difference in objective value when one is allowed to organize more than one round of contest? How does the objective value grow with the number of rounds?

Cross-References

- Algorithmic Mechanism Design
- Competitive Auction
- Generalized Vickrey Auction

Recommended Reading

1. Archak N, Sundararajan A (2009) Optimal design of crowdsourcing contests. In: International conference on information systems (ICIS), Phoenix
2. Chawla S, Hartline JD (2013) Auctions with unique equilibria. In: Proceedings of the fourteenth ACM conference on electronic commerce (EC '13), New York. ACM, pp 181–196
3. Chawla S, Hartline JD, Sivan B (2012) Optimal crowdsourcing contests. In: SODA, Kyoto, pp 856–868
4. DiPalantino D, Vojnovic M (2009) Crowdsourcing and all-pay auctions. In: Proceedings of the 10th ACM conference on electronic commerce (EC '09), Stanford, pp 119–128
5. Minor D (2011) Increasing effort through rewarding the best less. Mansucript. http://www.kellogg.northwestern.edu/faculty/minor/Papers/Increasing%20Effort%20%28with%20Figures%29.pdf
6. Moldovanu B, Sela A (2001) The optimal allocation of prizes in contests. Am Econ Rev 91(3):542–558
7. Moldovanu B, Sela A (2006) Contest architecture. J Econ Theory 126(1):70–97
8. Myerson R (1981) Optimal auction design. Math Oper Res 6:58–73
9. Yang J, Adamic LA, Ackerman MS (2008) Crowdsourcing and knowledge sharing: strategic user behavior on taskcn. In: Proceedings of the 9th ACM conference on electronic commerce (EC '08), Chicago, pp 246–255

Optimal Probabilistic Synchronous Byzantine Agreement

Juan Garay
Bell Laboratories, Murray Hill, NJ, USA

Keywords

Byzantine generals problem; Distributed consensus

Years and Authors of Summarized Original Work

1988; Feldman, Micali

Problem Definition

The Byzantine agreement problem (BA) is concerned with multiple processors (parties, "players") all starting with some initial value, agreeing on a common value, despite the possible disruptive or even malicious behavior of some them. BA is a fundamental problem in fault-tolerant distributed computing and secure multi-party computation.

The problem was introduced by Pease, Shostak and Lamport in [17], who showed that the number of faulty processors must be less than a third of the total number of processors for the problem to have a solution. They also presented a protocol matching this bound, which

requires a number of communication rounds proportional to the number of faulty processors – exactly $t + 1$, where t is the number of faulty processors. Fischer and Lynch [10] later showed that this number of rounds is necessary in the worst-case run of any deterministic BA protocol. Furthermore, the above assumes that communication takes place in synchronous rounds. Fischer, Lynch and Patterson [11] proved that no completely asynchronous BA protocol can tolerate even a single processor with the simplest form of misbehavior – namely, ceasing to function at an arbitrary point during the execution of the protocol ("crashing").

To circumvent the above-mentioned lower bound on the number of communication rounds and impossibility result, respectively, researchers beginning with Ben-Or [1] and Rabin [18], and followed by many others (e.g., [3, 5]) explored the use of randomization. In particular, Rabin showed that linearly resilient BA protocols in expected *constant* rounds were possible, provided that all the parties have access to a "common coin" (i.e., a common source of randomness) – essentially, the value of the coin can be adopted by the non-faulty processors in case disagreement at any given round is detected, a process that is repeated multiple times. This line of research culminated in the unconditional (or information-theoretic) setting with the work of Feldman and Micali [9], who showed an efficient (i.e., polynomial-time) probabilistic BA protocol tolerating the maximal number of faulty processors (Karlin and Yao, Probabilistic lower bounds for the byzantine generals problem, unpublished manuscript showed that the maximum number of faulty processors for probabilistic BA is also $t < \frac{n}{3}$, where n is the total number of processors.) that runs in expected constant number of rounds. The main achievement of the Feldman–Micali work is to show how to obtain a shared random coin with constant success probability in the presence of the maximum allowed number of misbehaving parties "from scratch".

Randomization has also been applied to BA protocols in the computational (or cryptographic) setting and for weaker failure models. See [6] for an early survey on the subject.

Notations

Consider a set $\mathcal{P} = \{P_1, P_2, \cdots, P_n\}$ of processors (probabilistic polynomial-time Turing machines) out of which t, $t < n$ may not follow the protocol, and even collude and behave in arbitrary ways. These processors are called *faulty*; it is useful to model the faulty processors as being coordinated by an adversary, sometimes called a *t-adversary*.

For $1 \leq i \leq n$, let b_i, $b_i \in \{0, 1\}$ denote party P_i's initial value. The work of Feldman and Micali considers the problem of designing a probabilistic BA protocol in the model defined below.

System Model

The processors are assumed to be connected by point-to-point private channels. Such a network is assumed to be synchronous, i.e., the processors have access to a global clock, and thus the computation of all processors can proceed in a lock-step fashion. It is customary to divide the computation of a synchronous network into *rounds*. In each round, processors send messages, receive messages, and perform some local computation.

The t-adversary is computationally unbounded, *adaptive* (i.e., it chooses which processors to corrupt on the fly), and decides on the messages the faulty processors send in a round depending on the messages sent by the non-faulty processors in all previous rounds, including the current round (this is called a *rushing* adversary).

Given the model above, the goal is to solve the problem stated in the ▸ Byzantine Agreement; that is, for every set of inputs and any behavior of the faulty processors, to have the non-faulty processors output a common value, subject to the additional condition that if they all start the computation with the same initial value, then that should be the output value. The difference with respect to the other entry is that, thanks to randomization, BA protocols here run in expected constant rounds.

Problem 1 (BA)
INPUT: Each processor P_i, $1 \leq i \leq n$, has bit b_i.
OUTPUT: Eventually, each processor P_i, $1 \leq i \leq n$, outputs bit d_i satisfying the following two conditions:

- Agreement: *For any two non-faulty processors P_i and P_j, $d_i = d_j$.*
- Validity: *If $b_i = b_j = b$ for all non-faulty processors P_i and P_j, then $d_i = b$ for all non-faulty processors P_i.*

In the above definition input and output values are from $\{0, 1\}$. This is without loss of generality, since there is a simple two-round transformation that reduces a multi-valued agreement problem to the binary problem [19].

Key Results

Theorem 1 *Let $t < \frac{n}{3}$. Then there exists a polynomial-time BA protocol running in expected constant number of rounds.*

The number of rounds of the Feldman–Micali BA protocol is expected constant, but there is no bound in the worst case; that is, for every r, the probability that the protocol proceeds for more than r rounds is very small, yet greater than 0 – in fact, equal to $2^{-O(r)}$. Further, the non-faulty processors may not terminate in the same round. (Indeed, it was shown by Dwork and Moses [7] that at least $t + 1$ rounds are necessary for simultaneous termination. In [13], Goldreich and Petrank combine "the best of both worlds" by showing a BA protocol running in expected constant number of rounds which always terminates within $t + O(\log t)$ rounds.)

The Feldman–Micali BA protocol assumes synchronous rounds. As mentioned above, one of the motivations for the use of randomization was to overcome the impossibility result due to Fischer, Lynch and Paterson [11] of BA in asynchronous networks, where there is no global clock, and the adversary is also allowed to schedule the arrival time of a message sent to a non-faulty processor (of course, faulty processors may not send any message(s)). In [8], Feldman mentions that the Feldman–Micali BA protocol can be modified to work on asynchronous networks, at the expense of tolerating $t < \frac{n}{4}$ faults. In [4], Canetti and Rabin present a probabilistic asynchronous BA protocol for $t < \frac{n}{3}$ that differs from the Feldman–Micali approach in that it is a Las Vegas protocol – i.e., it has non-terminating runs, but when it terminates, it does so in constant expected rounds.

Applications

There exists a one-to-one correspondence, possibility- and impossibility-wise between BA in the unconditional setting as defined above and a formulation of the problem called the "Byzantine generals" by Lamport, Shostak and Pease [15], where there is a distinguished source among the parties sending a value, call it b_s, and the rest of the parties having to agree on it. The Agreement condition remains unchanged; the Validity condition becomes

- VALIDITY: If the source is non-faulty, then $d_i = b_s$ for all non-faulty processors P_i.

A protocol for this version of the problem realizes a functionality called a "broadcast channel" on a network with only point-to-point connectivity. Such a tool is very useful in the context of cryptographic protocols and secure multi-party computation [12]. Probabilistic BA is particularly relevant here, since it provides a constant-round implementation of the functionality. In this respect, without any optimizations, the reported actual number of expected rounds of the Feldman–Micali BA protocol is at most 57. Recently, Katz and Koo [14] presented a probabilistic BA protocol with an expected number of rounds at most 23.

BA has many other applications. Refer to the ▸ Byzantine Agreement, as well as to, e.g., [16] for further discussion of other application areas.

Cross-References

- ▸ Asynchronous Consensus Impossibility
- ▸ Atomic Broadcast
- ▸ Byzantine Agreement
- ▸ Randomized Energy Balance Algorithms in Sensor Networks

Recommended Reading

1. Ben-Or M (1983) Another advantage of free choice: completely asynchronous agreement protocols. In: Proceedings of the 22nd annual ACM symposium on the principles of distributed computing, pp 27–30
2. Ben-Or M, El-Yaniv R (2003) Optimally-resilient interactive consistency in constant time. Distrib Comput 16(4):249–262
3. Bracha G (1987) An O(log n) expected rounds randomized Byzantine generals protocol. J Assoc Comput Mach 34(4):910–920
4. Canetti R, Rabin T (1993) Fast asynchronous Byzantine agreement with optimal resilience. In: Proceedings of the 25th annual ACM symposium on the theory of computing, San Diego, 16–18 May 1993, pp 42–51
5. Chor B, Coan B (1985) A simple and efficient randomized Byzantine agreement algorithm. IEEE Trans Softw Eng SE-11(6):531–539
6. Chor B, Dwork C (1989) Randomization in Byzantine agreement. Adv Comput Res 5:443–497
7. Dwork C, Moses Y (1990) Knowledge and common knowledge in a Byzantine environment: crash failures. Inf Comput 88(2):156–186, Preliminary version in TARK'86
8. Feldman P (1988) Optimal algorithms for Byzantine agreement. PhD thesis, MIT
9. Feldman P, Micali S (1997) An optimal probabilistic protocol for synchronous Byzantine agreement. SIAM J Comput 26(4):873–933, Preliminary version in STOC'88
10. Fischer MJ, Lynch NA (1982) A lower bound for the time to assure interactive consistency. Inf Process Lett 14(4):183–186
11. Fischer MJ, Lynch NA, Paterson MS (1985) Impossibility of distributed consensus with one faulty processor. J ACM 32(2):374–382
12. Goldreich O (2001/2004) Foundations of cryptography, vols 1 and 2. Cambridge University Press, Cambridge
13. Goldreich O, Petrank E (1990) The best of both worlds: guaranteeing termination in fast randomized agreement protocols. Inf Process Lett 36(1):45–49
14. Katz J, Koo C (2006) On expected constant-round protocols for Byzantine agreement. In: Proceedings of advances in cryptology–CRYPTO 2006, Santa Barbara. Springer, Berlin/Heidelberg/New York, pp 445–462
15. Lamport L, Shostak R, Pease M (1982) The Byzantine generals problem. ACM Trans Program Lang Syst 4(3):382–401
16. Lynch N (1996) Distributed algorithms. Morgan Kaufmann, San Francisco
17. Pease M, Shostak R, Lamport L (1980) Reaching agreement in the presence of faults. J ACM 27(2):228–234
18. Rabin M (1983) Randomized Byzantine generals. In: Proceedings of the 24th annual IEEE symposium on foundations of computer science, pp 403–409
19. Turpin R, Coan BA (1984) Extending binary Byzantine agreement to multivalued Byzantine agreement. Inf Process Lett 18(2):73–76

Optimal Stable Marriage

Robert W. Irving
School of Computing Science, University of Glasgow, Glasgow, UK

Keywords

Optimal stable matching

Years and Authors of Summarized Original Work

1987; Irving, Leather, Gusfield

Problem Definition

The classical *stable marriage problem* (SM), first studied by Gale and Shapley [5], is introduced in ▸ Stable Marriage. An instance of SM comprises a set $\mathcal{M} = \{m_1, \ldots, m_n\}$ of n men and a set $\mathcal{W} = \{w_1, \ldots, w_n\}$ of n women and for each person a *preference list*, which is a total order over the members of the opposite sex. A man's (respectively woman's) preference list indicates his (respectively her) strict order of preference over the women (respectively men). A matching M is a set of n man-woman pairs in which each person appears exactly once. If the pair (m, w) is

in the matching M, then m and w are *partners* in M, denoted by $w = M(m)$ and $m = M(w)$. Matching M is stable if there is no man m and woman w such that m prefers w to $M(m)$ and w prefers m to $M(w)$.

The key result established in [5] is that at least one stable matching exists for every instance of SM. In general, there may be many stable matchings, so the question arises as to what is an appropriate definition for the "best" stable matching and how such a matching may be found.

Gale and Shapley described an algorithm to find a stable matching for a given instance of SM. This algorithm may be applied either from the men's side or from the women's side. In the former case, it finds the so-called *man-optimal* stable matching, in which each man has the best partner, and each woman the worst partner, that is possible in any stable matching. In the latter case, the *woman-optimal* stable matching is found, in which these properties are interchanged. For some instances of SM, the man-optimal and woman-optimal stable matchings coincide, in which case this is the unique stable matching. In general, however, there may be many other stable matchings between these two extremes. Knuth [13] was first to show that the number of stable matchings can grow exponentially with n.

Because of the imbalance inherent, in general, in the man-optimal and woman-optimal solutions, several other notions of optimality in SM have been proposed.

A stable matching M is *egalitarian* if the sum

$$\sum_i r(m_i, M(m_i)) + \sum_j r(w_j, M(w_j))$$

is minimized over all stable matchings, where $r(m, w)$ represents the rank, or position, of w in m's preference list and similarly for $r(w, m)$. An egalitarian stable matching incorporates an optimality criterion that does not overtly favor the members of one sex – though it is easy to construct instances having many stable matchings in which the unique egalitarian stable matching is in fact the man (or woman) optimal.

A stable matching M is *minimum regret* if the value $\max(r(p, M(p)))$ is minimized over all stable matchings, where the maximum is taken over all persons p. A minimum-regret stable matching involves an optimality criterion based on the least happy member of the society, but again, minimum regret can coincide with man optimal or woman optimal in some cases, even when there are many stable matchings.

A stable matching is *rank maximal* (or *lexicographically maximal*) if, among all stable matchings, the largest number of people have their first choice partner and, subject to that, the largest number have their second choice partner and so on.

A stable matching M is *sex equal* if the difference

$$\left| \sum_i r(m_i, M(m_i)) - \sum_j r(w_j, M(w_j)) \right|$$

is minimized over all stable matchings. This definition is an explicit attempt to ensure that one sex is treated no more favorably than the other, subject to the overriding criterion of stability.

In the *weighted* stable marriage problem (WSM), each person has, as before, a strictly ordered preference list, but the entries in this list have associated costs or weights – $wt(m, w)$ represents the weight associated with woman w in the preference list of man m and likewise for $wt(w, m)$. It is assumed that the weights are strictly increasing along each preference list.

A stable matching M in an instance of WSM is *optimal* if

$$\sum_i wt(m_i, M(m_i)) + \sum_j wt(w_j, M(w_j))$$

is minimized over all stable matchings.

A stable matching M in an instance of WSM is *balanced* if

$$\max\left(\sum_i wt(m_i, M(m_i)), \sum_j wt(w_j, M(w_j)) \right)$$

is minimized over all stable matchings.

These same forms of optimality may be defined in the more general context of the stable marriage problem with incomplete preference lists (SMI); see ▸ Stable Marriage for a formal definition of this problem.

Again as described in ▸ Stable Marriage, the *stable roommates* problem (SR) is a non-bipartite generalization of SM, also introduced by Gale and Shapley [5]. In contrast to SM, an instance of SR may or may not admit a stable matching. Irving [9] gave the first polynomial-time algorithm to determine whether an SR instance admits a stable matching and if so to find one such matching.

There is no concept of man or woman optimal in the SR context, and nor is there any analogue of sex-equal or balanced matching. However, the other forms of optimality introduced above can be defined also for instances of SR and WSR (weighted stable roommates).

A comprehensive treatment of many aspects of the stable marriage problem, as of 1989, appears in the monograph of Gusfield and Irving [8], and a more recent detailed exposition is given by Manlove [14].

Key Results

The key to providing efficient algorithms for the various kinds of optimal stable matching is an understanding of the algebraic structure underlying an SM instance and the discovery of methods to exploit this structure. Knuth [13] attributes to Conway the observation that the set of stable matchings for an SM instance forms a distributive lattice under a natural dominance relation. Irving and Leather [10] characterized this lattice in terms of the so-called rotations – essentially minimal differences between lattice elements – which can be efficiently computed directly from the preference lists. The rotations form a natural partial order, the *rotation poset*, and there is a one-to-one correspondence between the stable matchings and the closed subsets of the rotation poset.

Building on these structural results, Gusfield [6] gave a $O(n^2)$ algorithm to find a Minimum-regret stable matching, improving an earlier $O(n^4)$ algorithm described by Knuth [13] and attributed to Selkow. Irving et al. [11] showed how the application of network flow methods to the rotation poset yields efficient algorithms for egalitarian and rank-maximal stable matchings as well as for an optimal stable matching in WSM. These algorithms have complexities $O(n^4)$, $O(n^5 \log n \log n)$ and $O(n^4 \log n)$, respectively. Subsequently, by using an interpretation of a stable marriage instance as an instance of 2-SAT and with the aid of a faster network flow algorithm exploiting the special structure of networks representing SM instances, Feder [3, 4] reduced these complexities to $O(n^3)$, $O(n^{3.5})$, and $O(\min(n, \sqrt{K})n^2 \log(K/n^2 + 2))$, respectively, where K is the weight of an optimal solution.

By way of contrast, and perhaps surprisingly, the problems of finding a sex-equal stable matching for an instance of SM and of finding a balanced stable matching for an instance of WSM have been shown to be NP-hard [2, 12].

The following theorem summarizes the current state of knowledge regarding the various flavors of optimal stable matching in SM and WSM.

Theorem 1 *For an instance of SM:*

1. *A minimum-regret stable matching can be found in $O(n^2)$ time.*
2. *An egalitarian stable matching can be found in $O(n^3)$ time.*
3. *A rank-maximal stable matching can be found in $O(n^{3.5})$ time.*
4. *The problem of finding a sex-equal stable matching is NP-hard.*

For an instance of WSM:

1. *An optimal stable matching can be found in $O(\min(n, \sqrt{K})n^2 \log(K/n^2+2))$ time, where K is the weight of an optimal solution.*

2. *The problem of finding a balanced stable matching is NP-hard, but can be approximated within a factor of 2 in $O(n^2)$ time.*

Among related problems that can also be solved efficiently by exploitation of the rotation structure of an instance of SM are the following [6]:

- All stable pairs, i.e., pairs that belong to at least one stable matching, can be found in $O(n^2)$ time.
- All stable matchings can be enumerated in $O(n^2 + kn)$ time, where k is the number of such matchings.

Results analogous to those of Theorem 1 are known for the more general SMI problem. In the case of the stable roommates problem (SR), some of these problems appear to be harder, as summarized in the following theorem.

Theorem 2 *For an instance of SR:*

1. *A minimum-regret stable matching can be found in $O(n^2)$ time [7].*
2. *The problem of finding an egalitarian stable matching is NP-hard. It can be approximated in polynomial time within a factor of α if and only if minimum vertex cover can be approximated within α [1, 2].*

For an instance of WSR (weighted stable roommates):

1. *The problem of finding an optimal stable matching is NP-hard, but can be approximated within a factor of 2 in $O(n^2)$ time [3].*

Applications

The best known and most important applications of stable matching algorithms are in centralized matching schemes in the medical and educational domains. ▸ Hospitals/Residents Problem includes a summary of some of these applications.

Open Problems

There remains the possibility of improving the complexity bounds for some of the optimization problems discussed and for finding better polynomial-time approximation algorithms for the various NP-hard problems.

Cross-References

▸ Hospitals/Residents Problem
▸ Ranked Matching
▸ Stable Marriage and Discrete Convex Analysis
▸ Stable Marriage with Ties and Incomplete Lists
▸ Stable Partition Problem

Recommended Reading

1. Feder T (1989) A new fixed point approach for stable networks and stable marriages. In: Proceedings of 21st ACM symposium on theory of computing, Seattle, May 1989. ACM, New York, pp 513–522
2. Feder T (1991) Stable networks and product graphs. Ph.D. thesis, Stanford University
3. Feder T (1992) A new fixed point approach for stable networks and stable marriages. J Comput Syst Sci 45:233–284
4. Feder T (1994) Network flow and 2-satisfiability. Algorithmica 11:291–319
5. Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69:9–15
6. Gusfield D (1987) Three fast algorithms for four problems in stable marriage. SIAM J Comput 16(1):111–128
7. Gusfield D (1988) The structure of the stable roommate problem: efficient representation and enumeration of all stable assignments. SIAM J Comput 17(4):742–769
8. Gusfield D, Irving RW (1989) The stable marriage problem: structure and algorithms. MIT, Cambridge
9. Irving RW (1985) An efficient algorithm for the stable roommates problem. J Algorithms 6:577–595
10. Irving RW, Leather P (1986) The complexity of counting stable marriages. SIAM J Comput 15(3):655–667
11. Irving RW, Leather P, Gusfield D (1987) An efficient algorithm for the "optimal stable" marriage. J ACM 34(3):532–543
12. Kato A (1993) Complexity of the sex-equal stable marriage problem. Jpn J Ind Appl Math 10:1–19
13. Knuth DE (1976) Mariages stables. Les Presses de L'Université de Montréal, Montréal
14. Manlove DF (2013) Algorithmics of matching under preferences. World Scientific, Singapore

Optimal Triangulation

Tiow-Seng Tan
School of Computing, National University of Singapore, Singapore, Singapore

Keywords

Edge-flip; Edge-insertion; Max-min height; Max-min length; Minimum weight; Min-max angle; Min-max eccentricity; Min-max length; Min-max slope

Years and Authors of Summarized Original Work

1972; Lawson
1992; Edelsbrunner, Tan, Waupotitsch
1993; Bern, Edelsbrunner, Eppstein, Mitchell, Tan
1993; Edelsbrunner, Tan

Problem Definition

Let S be a set of n *points* or *vertices* in $\mathbb{R}^2$. An *edge* is a closed line segment connecting two points. Let E be a collection of edges determined by vertices of S. The graph $\mathcal{G} = (S, E)$ is a *plane geometric graph* if (i) no edge contains a vertex other than its endpoints, that is, $ab \cap S = \{a, b\}$ for every edge $ab \in E$, and (ii) no two edges *cross*, that is, $ab \cap cd \in \{a, b\}$ for every two edges $ab \neq cd$ in E. A *triangulation* of S is a plane geometric graph $\mathcal{T} = (S, E)$ with E being maximal. Here maximality means that edges in E bound the convex hull of S, i.e., the smallest convex set in $\mathbb{R}^2$ that contains S, and subdivide its interior into disjoint faces bounded by triangles.

A plane geometric graph $\mathcal{G} = (S, E)$ can be augmented with an edge set E' so that it is a triangulation $\mathcal{T} = (S, E \cup E')$, referred to as a triangulation of $\mathcal{G}$. In this case, E is the set of *constraining edges* if it is not empty. Some triangulations of $\mathcal{G}$ are classified as *optimal triangulations* according to various shape criteria. Many of these criteria are defined as *max-min*, short for maximizing the minimum, or *min-max*, short for minimizing the maximum. The first quantifier is over all possible triangulations of $\mathcal{G}$ while the second one is over all measures (e.g., angles) μ of triangles of a triangulation. For example, in the case of a min-max μ criterion, we define the *measure* of a triangulation $\mathcal{A}$ as $\mu(\mathcal{A}) = \max\{\mu(t) : t \text{ is a triangle of } \mathcal{A}\}$. If $\mathcal{A}$ and $\mathcal{B}$ are two triangulations of $\mathcal{G}$, then $\mathcal{B}$ is called an *improvement* of $\mathcal{A}$ if either $\mu(\mathcal{B}) < \mu(\mathcal{A})$ or $\mu(\mathcal{B}) = \mu(\mathcal{A})$ and the set of triangles t of $\mathcal{B}$ with $\mu(t) = \mu(\mathcal{B})$ is a proper subset of that of $\mathcal{A}$. Triangulation $\mathcal{A}$ is *optimal* for μ, i.e., a min-max μ triangulation of $\mathcal{G}$, if there exists no improvement of $\mathcal{A}$. Hence, the computational problem addressed here is to find a specific optimal triangulation for a given plane geometric graph $\mathcal{G}$.

Key Results

There are a few algorithmic paradigms or approaches to solve the optimal triangulation problems in $\mathbb{R}^2$.

The Edge-Flip Approach

The most notable one is the edge-flip approach [11] to solve the max-min angle triangulation problem of a point set S. Given a triangulation $\mathcal{A}$ of $\mathcal{G} = (S, \emptyset)$, edge-flip is a local optimization method that operates on two adjacent triangles whose union forms a convex polygon. It replaces (or flips) the edge bd shared by triangles abd and cdb with the edge ac when the smallest angle of these triangles is smaller than that of abc and acd. In effect, an edge-flip replaces two existing triangles with two new triangles to (possibly) obtain an improvement of $\mathcal{A}$. By repeating the edge-flip until no such an edge bd exists, the algorithm produces a specific max-min angle triangulation of S, known as the Delaunay triangulation, in

$O(n^2)$ time. Besides being a max-min angle triangulation [3], Delaunay triangulation is also the min-max circumscribed circle and the min-max smallest enclosing circle [2] triangulation. Note that other approaches exist to compute the Delaunay triangulation more efficiently in $\Theta(n \log n)$ time [3].

The Edge-Insertion Approach

The edge-insertion approach is considered as an extension of the edge-flip approach, to replace one or more edges in each operation. The basic idea is to iteratively improve a current triangulation $\mathcal{A}$ by an *edge-insertion* step which adds an appropriate, new edge say qs to $\mathcal{A}$, deleting edges in $\mathcal{A}$ that cross qs and re-triangulating the resulting polygons to the left and the right of qs. In other words, the method starts by constructing an arbitrary triangulation $\mathcal{A}$ of $\mathcal{G}$ and then subsequently applies the edge-insertion steps until no further improvement exists. Same as in the case of edge-flip, this does not work for all measures μ as some may lead to suboptimal solutions. The approach is known to be applicable if the conditions of the so-called *Cake-Cutting Lemma*, which guarantees the existence of an improvement, are fulfilled; see [1, 5] for details. The next theorem summarizes the results obtained by the edge-insertion approach.

Theorem 1 *For a plane geometric graph $\mathcal{G} = (S, E)$ of $n = |S|$ vertices:*

1. *A min-max angle triangulation of $\mathcal{G}$ can be computed in time $O(n^2 \log n)$ and storage $O(n)$.*
2. *A max-min height triangulation of $\mathcal{G}$ can be computed in time $O(n^2 \log n)$ and storage $O(n)$.*
3. *A min-max eccentricity triangulation of $\mathcal{G}$ can be computed in time $O(n^3)$ and storage $O(n^2)$.*
4. *A min-max slope triangulation of $\mathcal{G}$ can be computed in time $O(n^3)$ and storage $O(n^2)$.*

Let us go through those measures mentioned in the theorem. The *height* of a triangle is the minimum distance from a vertex to the opposite edge. The *eccentricity* of a triangle is the infimum over all distances between the center of the circumcircle of the triangle and points in its closure. To define the slope of a triangle, the triangulation is given as a 2D projection of a 2.5D piecewise-linear surface where each vertex of S has a third coordinate, and the slope of each triangle is its slope in $\mathbb{R}^3$.

The Subgraph Approach

The subgraph approach constructs a desired optimal triangulation by first computing a substructure of the optimal triangulation and then completes the computation by solving the smaller problems defined by the substructure. This approach works when (i) the substructure can be computed efficiently and (ii) the substructure subdivides the problem into smaller problems such as polygons that can be solved efficiently. For instance, the approach has successfully solved the min-max length triangulation problem using a substructure called relative neighborhood graph [4]. Here the length of a triangle is the length of its longest edge.

Theorem 2 *A min-max length triangulation of a set of n points in $\mathbb{R}^2$ can be constructed in $O(n^2)$ time and storage.*

Note that the theorem is formulated with reference to a set of n points instead of the general plane geometric graph. In fact, this theorem is valid for the latter provided the minimization condition is defined over all edges (of triangles) including those constraining edges. In both cases, the correctness of the theorem follows from the fact that every point set S in $\mathbb{R}^2$ has a min-max length triangulation $\mathrm{mlt}(S)$ such that $\mathrm{rng}(S) \cup \mathrm{ch}(S) \subseteq \mathrm{mlt}(S)$ where $\mathrm{rng}(S)$ is the relative neighborhood graph of S and $\mathrm{ch}(S)$ is the set of edges bounding the convex hull of S. Since $\mathrm{rng}(S)$ and $\mathrm{ch}(S)$ can each be computed in $O(n \log n)$ time, and $\mathrm{rng}(S) \cup \mathrm{ch}(S)$ is a connected graph of S, the min-max length triangulation problem can be solved by first constructing $\mathrm{rng}(S) \cup \mathrm{ch}(S)$ and then computing an optimal triangulation within each polygon defined by edges of $\mathrm{rng}(S) \cup \mathrm{ch}(S)$. The latter is solvable in

$O(n^2)$ time. Besides Euclidean metric, Theorem 2 can be extended to general normed metrics as stated in the next theorem.

Theorem 3 *Let S be a set of n points in $\mathbb{R}^2$ equipped with a normed metric. Given the relative neighborhood graph, a min-max length triangulation of S can be constructed in time $O(n^2)$.*

Examples of normed metrics are the ℓ_p-metrics, for $p = 1, 2, 3, \ldots$, and the so-called A-metric used in VLSI applications. Note that the relative neighborhood graph under the ℓ_p-metrics can be computed in $O(n \log n)$ time. As for the other normed metrics, a relative neighborhood graph can be constructed in time $O(n^3)$ with a trivial approach to test all $\binom{n}{2}$ edges, each in time $O(n)$.

We note that min-max length is currently the only nontrivial length criterion known to be solvable in polynomial time. The max-min length triangulation problem for an input point set is shown to be NP-complete, while the same problem for a convex polygon is known to be solvable in linear time [6]. Another related problem is to find the minimum weight triangulation of $\mathcal{G}$, where the weight of a triangulation is the sum of length of its edges. This problem is proven to be NP-hard [12].

Applications

Triangulation is a prominent meshing method that decomposes a domain into a collection of triangles. Such decomposition is used in many areas of engineering and scientific applications such as physics simulation, visualization, approximation theory, numerical analysis, computer-aided geometric design, etc. It is desirable to obtain an optimal triangulation, often with respect to the angle, edge length, aspect ratio, etc., as the quality of the subsequent computation depends on the shapes of the triangles. Two popular techniques that greatly depend on such optimal triangulations are finite element analysis and surface interpolation; see, for example, the survey in [7].

Open Problems

There are a few other interesting measures one can define over a triangulation, such as area, aspect ratio, and vertex degree. The min-max area and max-min area triangulation problems for a point set are still open, though the special case of a convex polygon can be solved in polynomial time [10]. The problem to triangulate a plane geometric graph with degree at most seven is known to be NP-complete [8], and the min-max degree problem for an arbitrary biconnected plane geometric graph is also NP-complete [9]. Its general problem without any constraining edges is still open.

URLs to Code and Data Sets

A version of the edge-insertion approach was implemented by Roman Waupotitsch. It is known to be available at: ftp://ftp.ncsa.uiuc.edu/SGI/MinMaxer/

Cross-References

▸ Minimum Weight Triangulation

Recommended Reading

1. Bern M, Edelsbrunner H, Eppstein D, Mitchell S, Tan TS (1993) Edge insertion for optimal triangulations. Discret Comput Geom 10(1):47–65
2. D'Azevedo E, Simpson R (1989) On optimal interpolation triangle incidences. SIAM J Sci Stat Comput 10(6):1063–1075
3. Edelsbrunner H (2000) Triangulations and meshes in computational geometry. Acta Numer 2000 9:133–213
4. Edelsbrunner H, Tan TS (1993) A quadratic time algorithm for the minmax length triangulation. SIAM J Comput 22(3):527–551
5. Edelsbrunner H, Tan TS, Waupotitsch R (1992) An $O(n^2 \log n)$ time algorithm for the minmax angle triangulation. SIAM J Sci Stat Comput 13(4):994–1008
6. Fekete SP (2012) The complexity of maxmin length triangulation. CoRR abs/1208.0202

7. Ho-Le K (1988) Finite element mesh generation methods: a review and classification. Comput Aided Des 20(1):27–38
8. Jansen K (1993) One strike against the min-max degree triangulation problem. Comput Geom 3(2):107–120
9. Kant G, Bodlaender HL (1997) Triangulating planar graphs while minimizing the maximum degree. Inf Comput 135(1):1–14
10. Keil JM, Vassilev TS (2006) Algorithms for optimal area triangulations of a convex polygon. Comput Geom Theory Appl 35(3):173–187
11. Lawson CL (1972) Transforming triangulations. Discret Math 3(4):365–372
12. Mulzer W, Rote G (2008) Minimum-weight triangulation is NP-hard. J ACM 55(2):11:1–11:29

Optimal Two-Level Boolean Minimization

Robert P. Dick
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

Keywords

Logic minimization; Quine–McCluskey algorithm; Tabular method

Years and Authors of Summarized Original Work

1952; Quine
1955; Quine
1956; McCluskey

Problem Definition

Find a minimal sum-of-products expression for a Boolean function. Consider a Boolean algebra with elements *False* and *True*. A Boolean function $f(y_1, y_2, \ldots, y_n)$ of n Boolean input variables specifies, for each combination of input variable values, the function's value. It is possible to represent the same function with various expressions. For example, the first and last expressions in Table 1 correspond to the same function. Assuming access to complemented input variables, straightforward implementations of these expressions would require two *AND* gates and an *OR* gate for $\left(\overline{a} \wedge \overline{b}\right) \vee (\overline{a} \wedge b)$ and only a wire for $\overline{a}$. Although the implementation efficiency depends on target technology, in general terser expressions enable greater efficiency. Boolean minimization is the task of deriving the tersest expression for a function. Elegant and optimal algorithms exist for solving the variant of this problem in which the expression is limited to two levels, i.e., a layer of *AND* gates followed by a single *OR* gate or a layer of *OR* gates followed by a single *AND* gate.

Key Results

This survey will start by introducing the Karnaugh Map visualization technique, which will be used to assist in the subsequent explanation of the Quine–McCluskey algorithm for two-level Boolean minimization. This algorithm is optimal for its constrained problem variant. It is one of the fundamental algorithms in the field of computer-aided design and forms the basis or inspiration for many solutions to more general variants of the Boolean minimization problem.

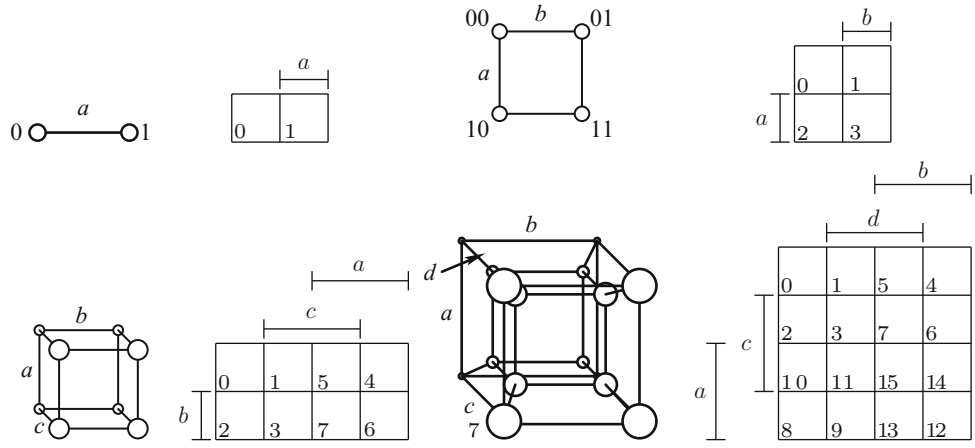
Karnaugh Maps

Karnaugh Maps [4] provide a method of visualizing adjacency in Boolean space. A Karnaugh Map is a projection of an n-dimensional hypercube onto a two-dimensional surface such that adjacent points in the hypercube remain adjacent in the two-dimensional projection. Figure 1 illustrates Karnaugh Maps of 1, 2, 3, and 4 variables: a, b, c, and d.

A *literal* is a single appearance of a complemented or uncomplemented input variable in a Boolean expression. A product term or *implicant* is the Boolean product, or *AND*, of one or more literals. Every implicant corresponds to the repeated balanced bisection of Boolean space, or of the corresponding Karnaugh Map, i.e., an

Optimal Two-Level Boolean Minimization, Table 1 Equivalent representations with different implementation complexities

Expression	Meaning in English	Boolean Logic Identity
$\overline{a} \wedge \overline{b} \vee \overline{a} \wedge b$	not a and not b or not a and b	Distributivity
$\overline{a} \wedge (\overline{b} \vee b)$	Not a and either not b or b	Complements
$\overline{a} \wedge True$	Not a and *True*	Boundeness
$\overline{a}$	not a	



Optimal Two-Level Boolean Minimization, Fig. 1 Boolean function spaces from one to four dimensions and their corresponding Karnaugh Maps

implicant is a rectangle in a Karnaugh Map with width m and height n where $m = 2^j$ and $n = 2^k$ for arbitrary nonnegative integers j and k, e.g., the ovals in Fig. 2(ii–v). An *elementary implicant* is an implicant in which, for each variable of the corresponding function, the variable or its complement appears, e.g., the circles in Fig. 2(ii). Implicant A *covers* implicant B if every elementary implicant in B is also in A.

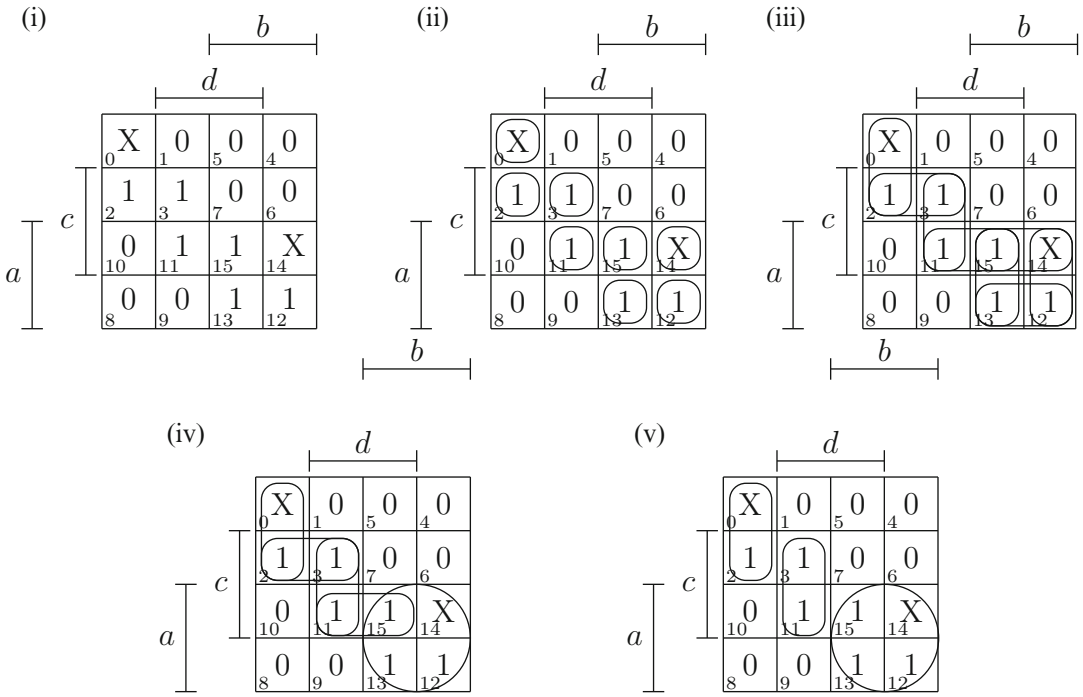
Prime implicants are implicants that are not covered by any other implicants, e.g., the ovals and circle in Fig. 2(iv). It is unnecessary to consider anything but prime implicants when seeking a minimal function representation because, if non-prime implicants could be used to cover some set of elementary implicants, there is guaranteed to exist a prime implicant that covers those elementary implicants and contains fewer literals. One can draw the largest implicants covering each elementary implicant and covering no positions for which the function is *False*, thereby using Karnaugh Maps to identify prime implicants. One can then manually seek a compact subset of prime implicants covering all elementary implicants in the function.

This Karnaugh Map-based approach is effective for functions with few inputs, i.e., those with low dimensionality. However, representing and manipulating Karnaugh Maps for functions of many variables is challenging. Moreover, the Karnaugh Map method provides no clear set of rules to follow when selecting a minimal subset of prime implicants to implement a function.

The Quine–McCluskey Algorithm

The Quine–McCluskey algorithm provides a formal, optimal way of solving the two-level Boolean minimization problem. W. V. Quine laid the essential theoretical groundwork for optimal two-level logic minimization [7, 8]. However, E. J. McCluskey first proposed a precise algorithm to fully automate the process [6]. Both are built upon the ideas of M. Karnaugh [4].

The Quine–McCluskey method has two phases: (1) produce all prime implicants and (2) select a minimal subset of prime implicants



Optimal Two-Level Boolean Minimization, Fig. 2 (**i**) Karnaugh Map of function $f(a,b,c,d)$, (**ii**) elementary implicants, (**iii**) second-order implicants, (**iv**) prime implicants, and (**v**) a minimal cover

covering the function. In the first phase, the elementary implicants of a function are iteratively combined to produce implicants with fewer literals. Eventually, all prime implicants are thus produced. In the second phase, a minimal subset of prime implicants covering the on-set elementary implicants is selected using unate covering [5].

The Quine–McCluskey method may be illustrated using an example. Consider the function indicated by the Karnaugh Map in Fig. 2(i) and the truth table in Table 2. For each combination of Boolean input variable values, the function $f(a,b,c,d)$ is required to output a 0 (*False*), a 1 (*True*), or has no requirements. The lack of requirements is indicated with an X, or don't-care symbol.

Expanding implicants as much as possible will ultimately produce the prime implicants. To do this, combine on-set and don't-care elementary implicants using the reduction theorem ($\overline{a}b \vee ab = b$) shown in Table 1. The elementary implicants are circled in Fig. 2(ii) and listed in the second column of Table 3. In this figure, 0s indicate complemented variables, and 1s indicate uncomplemented variables, e.g., 0010 corresponds to $\overline{a}\overline{b}c\overline{d}$. It is necessary to determine all possible combinations of implicants. It is impossible to combine nonadjacent implicants, i.e., those that differ in more than one variable. Therefore, it is not necessary to consider combining any pair of implicants with a number of uncomplemented variables differing by any value other than 1. This

Optimal Two-Level Boolean Minimization, Table 2 Truth table of function $f(a,b,c,d)$

Elementary implicant (a,b,c,d)	Function value (a,b,c,d)	Elementary implicant	Function value
0000	X	1000	0
0001	0	1001	0
0010	1	1010	0
0011	1	1011	1
0100	0	1100	1
0101	0	1101	1
0110	0	1110	X
0111	0	1111	0

Optimal Two-Level Boolean Minimization, Table 3
Identifying prime implicants

Number of ones	Elementary Implicant (a, b, c, d)	Second-order Implicant	Third-order Implicant
0	0000 ✓	00X0	
1	0010 ✓	001X	
2	0011 ✓	X011	11XX
	1100 ✓	110X ✓	
		11X0 ✓	
3	1011 ✓	1X11	
	1101 ✓	11X1 ✓	
	1110 ✓	111X ✓	
4	1111 ✓		

Optimal Two-Level Boolean Minimization, Table 4
Solving unate covering problem to select minimal cover

Requirements (elementary implicants)	Resources (prime implicants)				
	00X0	001X	X011	1X11	11XX
0010	✓	✓			
0011		✓	✓		
1011			✓	✓	
1100					✓
1101					✓
1111				✓	✓

fact can be exploited by organizing the implicants based on the number of ones they contain, as indicated by the first column in Table 3. All possible combinations of implicants in adjacent subsets are considered. For example, consider combining 0010 with 0011, which results in 001X or $\overline{a}\overline{b}c$, and also consider combining 0010 with 1100, which is impossible due to differences in more than one variable. Whenever an implicant is successfully merged, it is marked. These marked implicants are clearly not prime implicants because the implicants they produced cover them and contain fewer literals. Note that marked implicants should still be used for subsequent combinations. The merged implicants in the third column of Table 3 correspond to those depicted in Fig. 2(iii).

After all combinations of elementary implicants have been considered, and successful combinations listed in the third column, this process is repeated on the second-order merged implicants in the third column, producing the implicants in the fourth column. Implicants that contain don't-care marks in different locations may not be combined. This process is repeated until a column yielding no combinations is arrived at. The unmarked implicants in Table 3 are the prime implicants, which correspond to the implicants depicted in Fig. 2(iv).

After a function's prime implicants have been identified, it is necessary to select a minimal subset that covers the function. The problem can be formulated as unate covering. As shown in Table 4, label each column of a table with a prime implicant; these are resources that may be used to fulfill the requirements of the function. Label each row with an elementary implicant from the on-set; these rows correspond to requirements. Do not add rows for don't cares. Don't cares impose no requirements, although they were useful in simplifying prime implicants. Mark each row–column intersection for which the elementary implicant corresponding to the row is covered by the prime implicant corresponding to the column. If a column is selected, all the rows for which the column contains marks are *covered*, i.e., those requirements are satisfied. The goal is to cover all rows with a minimal-cost subset of columns. McCluskey defined minimal cost as having a minimal number of prime implicants, with ties broken by selecting the prime implicants containing the fewest literals. The most appropriate cost function depends on the implementation technology. One can also use a similar formulation with other cost functions, e.g., minimize the total number of literals by labeling each column with a cost corresponding to the number of literals in the corresponding prime implicant.

One can use a number of heuristics to accelerate solution of the unate covering problem, e.g., neglect rows that have a superset of the marks of any other row, for they will be implicitly covered and neglect columns that have a subset of the marks of any other column if their costs are as high, for the other column is at least as useful. One can easily select columns as long as there

exists a row with only one mark because the marked column is required for a valid solution. However, there exist problem instances in which each row contains multiple two marks. In the worst case, the best existing algorithms are required to make tentative decisions, determine the consequences, and then backtrack and evaluate alternative decisions.

The unate covering problem appears in many applications. It is $\mathcal{NP}$-complete [5], even for the instances arising during two-level minimization [9]. Its use in the Quine–McCluskey method predates its categorization as an $\mathcal{NP}$-complete problem by 16 years. A detailed treatment of this problem would go well beyond the scope of this entry. However, Gimpel [3] as well as Coudert and Madre [2] provide good starting points for further reading.

Some families of logic functions have optimal two-level representations that grow in size exponentially in the number of inputs, but have more compact multilevel implementations. These families are frequently encountered in arithmetic, e.g., a function indicating whether the number of on inputs is odd. Efficient implementation of such functions requires manual design or multilevel minimization [1].

Applications

Digital computers are composed of precisely two things: (1) implementations of Boolean logic functions and (2) memory elements. The Quine–McCluskey method is used to permit efficient implementation of Boolean logic functions in a wide range of digital logic devices, including computers. The Quine–McCluskey method served as a starting point or inspiration for most currently used logic minimization algorithms. Its direct use is contradicted when functions are not amenable to efficient two-level implementation, e.g., many arithmetic functions.

Cross-References

▸ Greedy Set-Cover Algorithms

Recommended Reading

1. Brayton RK, Hachtel GD, Sangiovanni-Vincentelli AL (1990) Multilevel logic synthesis. Proc IEEE 78(2):264–300
2. Coudert O, Madre JC (1995) New ideas for solving covering problems. In: Proceedings of the design automation conference, San Francisco, pp 641–646
3. Gimpel JF (1965) A reduction technique for prime implicant tables. IEEE Trans Electron Comput 14(4):535–541
4. Karnaugh M (1953) The map method for synthesis of combinational logic circuits. Trans AIEE Commun Electron 72:593–599
5. Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (eds) Complexity of computer computations. Plenum Press, New York, pp 85–103
6. McCluskey EJ (1956) Minimization of Boolean functions. Bell Syst Tech J 35(6):1417–1444
7. Quine WV (1952) The problem of simplyfying truth functions. Am Math Mon 59(8):521–531
8. Quine WV (1955) A way to simplify truth functions. Am Math Mon 62(9):627–631
9. Umans C, Villa T, Sangiovanni-Vincentelli AL (2006) Complexity of two-level logic minimization. IEEE Trans Comput-Aided Des Integr Circuits Syst 25(7):1230–1246

Orienteering Problems

Nitish Korula
Google Research, New York, NY, USA

Keywords

k-Stroll; k-TSP; k-MST; Orienteering; Routing with deadlines or time windows; Vehicle routing

Years and Authors of Summarized Original Work

2004; Bansal, Blum, Chawla, Meyerson
2007; Blum, Chawla, Karger, Lane, Meyerson, Minkoff
2011; Nagarajan, Ravi
2012; Chekuri, Korula, Pál

Problem Definition

The Orienteering problem and its variants are in the large class of vehicle routing problems, also containing the traveling salesperson problem (TSP), in which the goal is to find a short route that visits several potential destinations. Typically, the input is represented by a graph $G(V, E)$ with an associated length function $\ell: E \rightarrow R^+$, where each destination is a vertex $v \in V$, and an edge $e = (u, v)$ has length $\ell(e)$ representing the distance between u and v or the time it takes to travel between them. Unlike TSP, where the goal is to find a short tour visiting all vertices, Orienteering and its variants typically involve finding short walks that visit many vertices; having to choose the set of vertices to visit adds additional complexity to the problem.

In Orienteering, we are given a bound on the maximum length of the walk (also referred to as a budget), and the goal is to visit as many vertices as possible. A closely related problem is k-Stroll; here, we are given an integer k, and the goal is to find a walk that is as short as possible, subject to visiting at least k vertices. For both these problems, the walks are allowed to traverse an edge multiple times; the length of a walk W is $\sum_{e \in W} \ell(e)$. Hence, w.l.o.g., one can assume that the input graph is complete (by working with its metric completion) or, equivalently, that the input is represented by a metric.

We focus mainly on the "point-to-point" versions of these problems, in which the start and end vertices of the walk are also specified; here, the goal is to find a short walk from the specified start vertex to the specified end vertex that visits many other vertices. One can also consider the variant in which only the start vertex is specified (and the algorithm can choose where to end the walk) or the one in which neither the start nor the end vertex is specified. These are referred to as the "rooted" and "unrooted" variants, respectively. We define the problems formally below.

Problem 1 (Orienteering)

INPUT: Graph $G(V, E)$, with an associated length function $\ell: E \rightarrow R^+$, start and end vertices $s, t \in V$, and a budget/length bound L.
OUTPUT: An s-t walk of total length at most L.
OBJECTIVE: Maximize the number of distinct vertices in the walk.

Problem 2 (k-Stroll)

INPUT: Graph $G(V, E)$, with an associated length function $\ell: E \rightarrow R^+$, start and end vertices $s, t \in V$, and an integer k.
OUTPUT: An s-t walk containing at least k distinct vertices.
OBJECTIVE: Minimize the total length of the walk.

Orienteering and k-Stroll are "dual" problems. They are equivalent in terms of exact solvability; a polynomial-time optimal algorithm for one can be used to obtain a polynomial-time optimal algorithm for the other. However, this is not true from the standpoint of approximability; an α-approximation for one does not immediately imply an α-approximation for the other.

Orienteering with time windows (Orient-TW) is a generalization of Orienteering in which each vertex v has an associated time interval or window $[R(v), D(v)]$, and a vertex is considered "visited" (i.e., is counted toward the objective) only if the total length of the walk from the start vertex up to v is in the range $[R(v), D(v)]$. (For intuition, if the length of an edge is interpreted as the time taken to traverse it, then the vertex is counted if the time at which it is visited falls within its time window.) $R(v)$ and $D(v)$ are referred to as the release time and deadline of vertex v, respectively. A special case of this problem (sometimes called orienteering with deadlines or even deadline-TSP) is when $R(v) = 0$ for all vertices v.

Problem 3 (Orienteering with Time Windows)

INPUT: Graph $G(V, E)$, with an associated length function $\ell: E \rightarrow R^+$, start and end vertices $s, t \in V$, a budget/length bound L, and a time interval $[R(v), D(v)]$ for each vertex $v \in V$.
OUTPUT: An s-t walk of total length at most L.
OBJECTIVE: Maximize the number of distinct vertices in the walk that are visited during

their time intervals. A vertex is counted as visited only if the walk visits v at a time $t \in [R(v), D(v)]$; we assume it takes ℓ units of time to cross an edge of length ℓ.

See [9] for an overview and applications of many vehicle routing problems related to Orienteering and its variants.

Key Results

Orienteering is both NP-hard and APX-hard [3]; the same applies for k-Stroll, as a generalization of TSP. Therefore, we focus primarily on approximability.

Undirected Graphs

Arkin et al. [1] gave a 2-approximation for rooted Orienteering in the Euclidean plane. Chen and Har-Peled [7] improved this to a PTAS for the plane and higher-dimensional Euclidean metrics.

For general undirected graphs (i.e., symmetric metrics), the first constant-factor approximation for *rooted* Orienteering was given by Blum et al. [3]. They obtained many of the key insights used in subsequent papers, reducing Orienteering to k-Stroll. Blum et al. [3] showed that an α-approximation to k-Stroll gives a $1 + \lceil \frac{3\alpha}{2} - \frac{1}{2} \rceil$-approximation for rooted Orienteering. This was improved by Bansal et al. [2] to $\lceil \frac{3\alpha}{2} - \frac{1}{2} \rceil$ even for the harder point-to-point variant. Since there is a $2 + \varepsilon$-approximation for k-Stroll due to [4], this gives a 3-approximation for Orienteering. Chekuri et al. [6] reduced Orienteering to a *bicriteria* version for k-Stroll; this gives the current best $2+\varepsilon$-approximation for Orienteering, matching the ratio for k-Stroll.

A key challenge in Orienteering is the hard constraint on the total length of the walk L. In particular, if the shortest path from the source s to the destination t has length close to L, then even a small detour from the shortest path to visit a cluster of many vertices might result in reaching t after the deadline L. Roughly speaking, [3] shows that an optimum walk can be broken down into segments which are "monotonic," meaning that they visit vertices in increasing order of their (shortest path) distance from s, and segments which are "non-monotonic," in which case the length of the segment is at least 3 times the shortest path between its endpoints. To find a good walk, one can use a dynamic program to "enumerate" all relevant segmentations of the optimal path; for each monotonic segment, one can find the optimum sub-path using dynamic programming. For the non-monotonic segments, one can "skip" the reward from some of them, which saves considerable distance since the detours taken by the path in such segments are large. This saving allows one to take a little extra distance to collect reward in the remaining non-monotonic segments (using an approximation algorithm for k-Stroll) while still keeping the total length of the walk at most L.

Directed Graphs

Orienteering is more challenging in directed graphs, or asymmetric metrics. The first poly-logarithmic approximation algorithms were due independently to [6, 8]; the former gave an $O(\log^2 n / \log\log n)$-approximation using an LP-based approach, while the latter gave an $O(\log^2 \textsc{Opt})$-approximation using combinatorial techniques. The ratio of [6] is better when Opt, the number of vertices visited by an optimal walk, is much less than n and is slightly worse otherwise; on the other hand, the LP of [8] is based on the well-known Held-Karp relaxation for asymmetric TSP, and a conjectured improved upper bound on the integrality gap of this relaxation would immediately give an improved approximation ratio for directed Orienteering. Both these papers also obtain poly-logarithmic approximations for the closely related problem-Directed k-TSP, which is the special case of k-Stroll when $s = t$.

Chekuri and Pál [5] gave a *quasi-polynomial*-time $O(\log n)$-approximation for directed Orienteering and several generalizations, including Orient-TW. This algorithm is based on repeatedly "guessing" the midpoint of sub-paths, and hence it does not appear easy to obtain a polynomial-time equivalent.

Orienteering with Time Windows

Orient-TW in undirected graphs, with arbitrary distinct release times and deadlines for each vertex, was first studied by [2], which gave an $O(\log n)$-approximation for the case with only deadlines (i.e., where $R(v) = 0$ for all v) and $O(\log^2 n)$ for the general problem. Chekuri et al. [6] later gave an $O(\max\{\log \text{OPT}, \log R\})$-approximation for Orient-TW, where R denotes the ratio between the length of the longest and shortest time windows; when all time windows are polynomially bounded, this is an $O(\log n)$-approximation. (For directed graphs, via [6, 8], we lose additional poly-logarithmic factors).

The general approach taken by these papers, following [2], is to use combinatorial techniques to reduce the given instance of the problem to a collection of subproblems in which all vertices have identical or disjoint time windows. For a set of vertices with identical time windows, these windows can be effectively ignored, yielding an instance of Orienteering; walks for different sets with disjoint time windows can be combined using dynamic programming. Thus, Orient-TW can be reduced to Orienteering with the loss of logarithmic factors in the approximation ratio.

Open Problems

There are several natural open problems related to Orienteering.

1. Is there a PTAS for Orienteering in undirected planar graphs? In recent years, PTASes have been obtained for many related problems, including TSP, STEINER TREE, and their prize-collecting versions, but extending these techniques to Orienteering and k-Stroll (or even the easier k-MST problem) seems challenging.
2. Can one obtain an $O(\log n)$ or even $O(1)$-approximation for directed Orienteering? The quasi-polynomial-time approximation of [5] provides some evidence that it may be possible. Can one obtain *any* poly-logarithmic approximation for directed k-Stroll? Currently, only bicriteria approximations are known.
3. Is there an $O(\log n)$-approximation for Orient-TW?

Recommended Reading

1. Arkin E, Mitchell J, Narasimhan G (1998) Resource-constrained geometric network optimization. In: Symposium on computational geometry, Minneapolis, pp 307–316
2. Bansal N, Blum A, Chawla S, Meyerson A (2004) Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In: Proceedings of the 36th annual ACM symposium on theory of computing, Chicago. ACM, New York, pp 166–174
3. Blum A, Chawla S, Karger D, Lane T, Meyerson A, Minkoff M (2007) Approximation algorithms for orienteering and discounted-reward TSP. SIAM J Comput 37(2):653–670
4. Chaudhuri K, Godfrey B, Rao S, Talwar K (2003) Paths, trees, and minimum latency tours. In: 44th annual symposium on foundations of computer science, Cambridge. IEEE Computer Society, pp 36–45
5. Chekuri C, Pál M (2005) A recursive greedy algorithm for walks in directed graphs. In: Proceedings of the 46th annual symposium on foundations of computer science, Pittsburgh. IEEE Computer Society, pp 245–253
6. Chekuri C, Korula N, Pál M (2012) Improved algorithms for orienteering and related problems. ACM Trans Algorithms (TALG) 8(3):23
7. Chen K, Har-Peled S (2008) The orienteering problem in the plane revisited. SIAM J Comput 38(1):385–397, preliminary version in Proceedings of the ACM SoCG, Sedona, 2006, pp 247–254
8. Nagarajan V, Ravi R (2011) The directed orienteering problem. Algorithmica 60(4):1017–1030
9. Toth P, Vigo D (eds) (2001) The vehicle routing problem. SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics, Philadelphia

Orthogonal Range Searching on Discrete Grids

Yakov Nekrich
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Keywords

Orthogonal range searching; Word RAM model

Years and Authors of Summarized Original Work

1988; Chazelle
2000; Alstrup, Brodal, Rauhe
2004; JaJá, Mortensen, Shi
2007; Pătraşcu
2009; Karpinski, Nekrich
2011; Chan, Larsen, Pătraşcu
2013; Chan

Problem Definition

Let S be a set of n d-dimensional points. In the orthogonal range searching problem we keep S in a data structure, so that for an arbitrary query rectangle $Q = [a_1, b_1] \times \cdots \times [a_d, b_d]$ information about points in $Q \cap S$ can be found. Range searching is a fundamental computational geometry problem with numerous applications in data bases, text indexing, string processing, and network analysis. In computational geometry it is frequently assumed that point coordinates are real and the data structure works in the real RAM model. In a vast majority of practical situations we can, however, make a stronger assumption that point coordinates are discrete values. This scenario is captured by the word RAM model of computation: all coordinates are integers that fit into a machine word and standard operations on words can be performed in constant time. In this case, we say that points are on an integer grid. If point coordinates are also bounded by a parameter U, we say that points are on a grid of size U (also called a $U \times U$ grid if points are two-dimensional).

The discrete grid assumption leads to improved results for many range searching problems. In this entry we consider several such problems. We remark that a problem on an integer grid can be always reduced to the same problem on a grid of size n using the technique called reduction to rank space [1, 8]. This reduction takes $O(n)$ additional space and increases the query time by an additive term $\mathrm{pred}(n, U)$, where $\mathrm{pred}(n, U) = O(\min(\log\log U, \sqrt{\log n / \log\log n}))$ is the time needed to answer a predecessor query [2, 21]. Unless specified otherwise, we will assume that all points are on a grid of size n. Throughout this entry, the space usage of described data structures is measured in words; each word consists of $w \geq \log n$ bits and can hold a coordinate of any point from the input set.

Key Results

Orthogonal Range Reporting

Two Dimensions

The problem is to keep a set of points S in a data structure, so that all points in $Q \cap S$ for any query rectangle Q can be reported. Overmars [17] was the first to consider this problem in the discrete grid scenario. His data structure needs $O(n \log n)$ words of space and supports two-dimensional queries on $U \times U$ grid in $O(\log\log U + k)$ time, where k is the number of reported points. Alstrup et al. [1] improved the space usage and described a data structure that uses $O(n \log^{\varepsilon} n)$ space and supports queries in $O(\log\log n + k)$ time. Henceforth, ε denotes an arbitrarily small positive constant. There are also data structures that use less space but need more than constant time per reported point. Chan et al. [6] describe an $O(n)$-space data structure that answers two-dimensional queries in $O((k+1)\log^{\varepsilon} n)$ time and an $O(n \log\log n)$-space data structure that answers queries in $O((k+1)\log\log n)$ time. Any data structure that uses $n \log^{O(1)} n$ space needs $\Omega(\log\log n + k)$ time to answer two-dimensional range reporting queries. This follows from the reduction of two-dimensional range reporting problem to the predecessor problem [12] and the lower bound for the predecessor problem [20].

In the special case of three-sided range reporting queries, the query range is bounded on three sides. Thus, a three-sided query range is a product of a closed interval and a half-open interval. Three-sided range reporting queries can be answered in $O(k+1)$ time using a linear space data structure [1]. The restriction on point coordinates can be relaxed for one dimension in the case of three-sided queries: there is a linear-space data structure for points on $n \times \mathbb{N}$ grid

that answers queries of the form $[a, b] \times (-\infty, d]$ or $[a, b] \times [c, +\infty)$ in $O(k + 1)$ time. Symmetrically, there is also a linear-space data structure for points on $\mathbb{N} \times n$ grid that answers queries $[a, +\infty) \times [c, d]$ or $(-\infty, b] \times [c, d]$ in $O(k + 1)$ time. We list the best currently known results for two-dimensional orthogonal range reporting in rows 1–4 of Table 1.

Three Dimensions

Three-dimensional queries can be answered in optimal $O(\log \log n + k)$ time and $O(n \log^{1+\varepsilon} n)$ space [6]. There is also an $O(n \log n (\log \log n)^3)$-space data structure that answers queries in $O((\log \log n)^2 + k \log \log n)$ time; this result is achieved by combining the approach of [11] with the point location data structure of [4]. Alternatively, there is a data structure that uses $O(n \log n)$ space and answers queries in $O(\log^{1+\varepsilon} n + k \log^{\varepsilon} n)$ time [7]. Better space bounds can be achieved for some special cases. A three-dimensional query that is a product of three closed intervals, $Q = [a, b] \times [c, d] \times [e, f]$, is called a $(2, 2, 2)$-sided query; this is the most general case of three-dimensional queries. We say that a three-dimensional orthogonal range reporting query Q is a $(2, 2, 1)$-sided query if it is a product of two closed intervals and one half-open interval: $Q = [a, b] \times [c, d] \times [e, +\infty)$. There exists a data structure that uses $O(n \log^{\varepsilon} n)$ space and answers $(2, 2, 1)$-sided queries in $O(\log \log n + k)$ time [6]. Since $(2, 2, 1)$-sided queries are not easier to answer than two-dimensional queries, this query time is optimal. There is also a data structure that uses $O(n(\log \log n)^3)$ space and answers $(2, 2, 1)$-sided queries in $O((\log \log n)^2 + k \log \log n)$ time; this result is obtained by combining the approach of Karpinski and Nekrich [11] with the point location data structure of Chan [4]. Finally, we can also answer $(2, 2, 1)$-sided queries in $O(\log^{1+\varepsilon} n + k \log^{\varepsilon} n)$ time using a linear-space data structure [7]. A $(1, 1, 1)$-sided query (also known as dominance query) Q is a product of three half-open intervals: $Q = (-\infty, a] \times (-\infty, b] \times (-\infty, c]$. Chan [4] describes a linear space data structure that answers $(1, 1, 1)$-sided queries in optimal $O(\log \log n + k)$ time. See Table 2.

Orthogonal Range Searching on Discrete Grids, Table 1 Two-dimensional orthogonal range reporting. Four-sided queries denote general two-dimensional queries

Ref.	Space	Query time	Grid	Remarks
[1]	$O(n)$	$O(k + 1)$	$n \times \mathbb{N}$	3-sided
[1]	$O(n \log^{\varepsilon} n)$	$O(\log \log n + k)$	$n \times n$	4-sided
[6]	$O(n \log \log n)$	$O((k + 1) \log \log n)$	$n \times n$	4-sided
[6]	$O(n)$	$O((k + 1) \log^{\varepsilon} n)$	$n \times n$	4-sided
[16]	$O(n \log^{\varepsilon} n)$	$O(\log \log n + k')$	$n \times n$	Sorted 4-sided
[22]	$O(n \log \log n)$	$O((k' + 1) \log \log n)$	$n \times n$	Sorted 4-sided
[16]	$O(n)$	$O((k' + 1) \log^{\varepsilon} n)$	$n \times n$	Sorted 4-sided

Orthogonal Range Searching on Discrete Grids, Table 2 Three-dimensional orthogonal range reporting

Ref.	Space	Query time	Query type
[6]	$O(n \log^{1+\varepsilon} n)$	$O(\log \log n + k)$	$(2, 2, 2)$-sided
[11] + [4]	$O(n \log n (\log \log n)^3)$	$O((\log \log n)^2 + k \log \log n)$	$(2, 2, 2)$-sided
[7]	$O(n \log n)$	$O(\log^{1+\varepsilon} n + k \log^{\varepsilon} n)$	$(2, 2, 2)$-sided
[6]	$O(n \log^{\varepsilon} n)$	$O(\log \log n + k)$	$(2, 2, 1)$-sided
[11] + [4]	$O(n(\log \log n)^3)$	$O((\log \log n)^2 + k \log \log n)$	$(2, 2, 1)$-sided
[7]	$O(n)$	$O(\log^{1+\varepsilon} n + k \log^{\varepsilon} n)$	$(2, 2, 1)$-sided
[4]	$O(n)$	$O(\log \log n + k)$	$(1, 1, 1)$-sided

Multi-dimensional Queries

Using range trees [3], we can extend the three-dimensional data structures so that d-dimensional queries for any integer constant d can be answered. The query time and space usage grow by $O(\log^{d-3} n)$. We can also increase the arity of range trees, so that each internal node has $O(\log^{\varepsilon/(2(d-3))} n)$ children. In this case, the query time grows by $O((\log n/\log\log n)^{d-3})$ and the space usage grows by $O(\log^{d-3+\varepsilon} n)$. Thus, there is a data structure that uses $O(n \log^{2+\varepsilon} n)$ space and answers four-dimensional range reporting queries in $O(\log n + k)$ time [6]. This result almost matches the lower bound of Pătraşcu [19] stating that any $n \log^{O(1)} n$-space data structure answers four-dimensional orthogonal range reporting queries in $\Omega(\log n/\log\log n)$ time. The best currently known d-dimensional data structure needs $O((\log n/\log\log n)^{d-3} \log\log n + k)$ time and uses $O(n \log^{d-2+\varepsilon} n)$ space [6].

Emptiness Queries and One-Reporting Queries

An orthogonal range emptiness query Q asks whether Q contains any points of the input set S. A one-reporting query Q asks for an arbitrary point p in $Q \cap S$ if $Q \cap S \neq \emptyset$; if $Q \cap S = \emptyset$, a special $NULL$ value is returned. We can employ all data structures described above for answering emptiness and one-reporting queries. Any previously described data structure that answers range reporting queries in time $O(q(n) + k \cdot q'(n))$ can be used to answer emptiness and one-reporting queries (for the same dimension and the same query type) in $O(q(n))$ time and $O(q(n) + q'(n))$ time, respectively.

Two-Dimensional Range Successor and Sorted Range Reporting Queries

An orthogonal range successor (also known as range next value) query $Q = [a, b] \times [c, d]$ asks for the leftmost point in $S \cap Q$. In [16] the authors considered the following generalization of range successor queries: for a query range $Q = [a, b] \times [c, d]$ report all points in $Q \cap S$ in left-to-right order. Sorted range reporting queries can also be answered in online modus: a query can be terminated when the k' leftmost points in $Q \cap S$ are reported for any $k' \leq |Q \cap S|$ and k' can be specified at query time. Nekrich and Navarro [16] describe data structures that use $O(n)$ and $O(n \log^{\varepsilon} n)$ space and answer sorted range reporting queries in $O((k' + 1) \log^{\varepsilon} n)$ and $O(\log\log n + k')$ time, respectively. The data structure of [22] needs $O(n \log\log n)$ time and supports queries in $O((k' + 1) \log\log n)$ time. See Table 1. Sorted range reporting queries for $k' = 1$ are equivalent to range successor queries. Thus, data structures for sorted range reporting match the complexity of the best currently known structures for standard two-dimensional point reporting (respectively, the data structures for range successor queries match data structures for one-reporting in two dimensions).

Orthogonal Range Counting

The problem is to keep a set of points S in a data structure so that for any query rectangle Q, the number of points in $Q \cap S$ can be computed. The data structure of [10] uses $O(n(\log n/\log\log n)^{d-2})$ space and answers d-dimensional dominance counting queries (i.e., counts points in a range that is a product of d half-open intervals) in $O((\log n/\log\log n)^{d-1})$ time. We can count points in any d-dimensional rectangle by answering $O(2^d)$ d-dimensional dominance queries. Hence, d-dimensional range counting queries can also be answered in $O((\log n/\log\log n)^{d-1})$ time and $O(n(\log n/\log\log n)^{d-2})$ space for any constant d. Dynamic data structures that use $O(n)$ space, answer two-dimensional range counting queries in $O((\log n/\log\log n)^2)$ time, and support updates in poly-logarithmic time are described in [14] and [9]. Query time of the static data structure is optimal for $d = 2$ dimensions: by the lower bound of [18], any two-dimensional data structure that uses $n \log^{O(1)} n$ space needs $\Omega(\log n/\log\log n)$ time to answer queries.

We can, however, reduce the query cost if k is small, where $k = |Q \cap S|$ is the number of points in a query range. Chan and Wilkinson [5] describe a data structure that uses $O(n \log\log n)$ words of space and answers two-dimensional

range counting queries in optimal $O(\log\log n + \log_w k)$ time, where w is the size of the machine word. Nekrich [15] describes a data structure that uses $O(n)$ words and counts the number of points in a two-dimensional three-sided range in optimal $O(\log_w k)$ time. There are also data structures that provide an approximate answer for two- and three-dimensional range counting queries [5, 13, 15]. These data structures return a value $\overline{k}$ such that $(1-\delta)k \leq \overline{k} \leq (1+\delta)k$ for an arbitrary query Q and some fixed constant $\delta > 0$. We can answer two-dimensional approximate counting queries in $O(\log\log n)$ time using an $O(n\log\log n)$-space data structure [5]; we can answer three-sided approximate counting queries in $O(1)$ time using an $O(n)$-space data structure [15]. An approximate answer to a three-dimensional dominance counting query can be obtained in $O((\log\log n)^3)$ time using an $O(n)$-space data structure; we can estimate the number of points in any three-dimensional range within the same time using an $O(n\log^3 n)$-space data structure [13]. If we plug the point location data structure of [4] into the data structure of [13], then the query time of approximate three-dimensional queries is reduced to $O((\log\log n)^2)$.

Open Problems

In spite of extensive research and significant achievements, many important questions are still open. The best currently known data structure that supports two-dimensional reporting queries in optimal time needs $O(n\log^{\varepsilon} n)$ space [1]. Existence of a data structure that uses $o(n\log^{\varepsilon} n)$ space for any $\varepsilon > 0$ and achieves optimal query time is an interesting open question. Another important problem is improving the space complexity of d-dimensional range reporting for $d > 2$ dimensions and query time of d-dimensional range reporting for $d > 4$ dimensions.

Cross-References

▸ Predecessor Search

Recommended Reading

1. Alstrup S, Brodal G. S., Rauhe T (2000) New data structures for orthogonal range searching. In: Proceedings of the 41st annual symposium on foundations of computer science (FOCS 2000), Redondo Beach, pp 198–207
2. Beame P, Fich F. E. (2002) Optimal bounds for the predecessor problem and related problems. J Comput Syst Sci 65(1):38–72
3. Bentley J. L. (1980) Multi- dimensional divide-and-conquer. Commun ACM 23(4):214–229
4. Chan T. M. (2013) Persistent predecessor search and orthogonal point location on the word RAM. ACM Trans Algorithms 9(3):22
5. Chan T. M., Wilkinson B. T. (2013) Adaptive and approximate orthogonal range counting. In: Proceedings of the 24th annual ACM-SIAM symposium on discrete algorithms (SODA 2013), New Orleans, pp 241–251
6. Chan T. M., Larsen K. G., Pătraşcu M (2011) Orthogonal range searching on the RAM, revisited. In: Proceedings of the 27th SoCG, Paris, pp 1–10
7. Chazelle B (1988) A functional approach to data structures and its use in multi-dimensional searching. SIAM J Comput 17(3):427–462
8. Gabow H. N., Bentley J. L., Tarjan RE (1984) Scaling and related techniques for geometry problems. In: Proceedings of the 16th annual ACM symposium on theory of computing (STOC 84), Washington, DC, pp 135–143
9. He M, Munro J. I. (2014) Space-efficient data structures for dynamic orthogonal range counting. Comput Geom 47(2):268–281
10. JáJá J, Mortensen C. W., Shi Q (2004) Space-efficient and fast algorithms for multi-dimensional dominance reporting and counting. In: Proceedings of the 15th international symposium on algorithms and computation (ISAAC 2004), HongKong, pp 558–568
11. Karpinski M, Nekrich Y (2009) Space-efficient multi-dimensional range reporting. In: Proceedings of the 15th annual international conference on computing and combinatorics (COCOON 2009), Niagara Falls, pp 215–224
12. Miltersen P. B., Nisan N, Safra S, Wigderson A (1998) On data structures and asymmetric communication complexity. J Comput Syst Sci 57(1):37–49
13. Nekrich Y (2009) Data structures for approximate orthogonal range counting. In: Proceedings of the 20th international symposium on algorithms and computation (ISAAC 2009), Honolulu, pp 183–192
14. Nekrich Y (2009) Orthogonal range searching in linear and almost-linear space. Comput Geom 42(4):342–351
15. Nekrich Y (2014) Efficient range searching for categorical and plain data. ACM Trans Database Syst 39(1):9
16. Nekrich Y, Navarro G (2012) Sorted range reporting. In: Proceedings of the 13th Scandinavian symposium

and workshops on algorithm theory (SWAT 2012), Helsinki, pp 271–282
17. Overmars M. H. (1988) Efficient data structures for range searching on a grid. J Algorithms 9(2):254–275
18. Pătraşcu M (2007) Lower bounds for 2-dimensional range counting. In: Proceedings of the 39th annual ACM symposium on theory of computing (STOC 2007), San Diego, pp 40–46
19. Pătraşcu M (2010) Towards polynomial lower bounds for dynamic problems. In: Proceedings of the 42nd ACM symposium on theory of computing (STOC 2010), Cambridge, pp 603–610
20. Pătraşcu M, Thorup M (2006) Time-space trade-offs for predecessor search. In: Proceedings of the 38th annual ACM symposium on theory of computing (STOC 2006), Seattle, pp 232–240
21. van Emde Boas P, Kaas R, Zijlstra E (1977) Design and implementation of an efficient priority queue. Math Syst Theory 10:99–127
22. Zhou G (2013) Sorted range reporting revisited. ArXiv e-prints 12044509 1308.3326

P

P2P

Dahlia Malkhi
Microsoft, Silicon Valley Campus,
Mountain View, CA, USA

Keywords

CDN; Content delivery network; DHT; Distributed hash table; File sharing; Peer to peer; Overlay; Overlay network; Resource sharing

Years and Authors of Summarized Original Work

2001; Stoica, Morris, Karger, Kaashoek, Balakrishnan

Problem Definition

This problem is concerned with efficiently designing a serverless infrastructure for a federation of hosts to store, index and locate information, and for efficient data dissemination among the hosts. The key services of peer-to-peer (P2P) overlay networks are:

1. A keyed lookup protocol locates information at the server(s) that hold it.
2. Data store, update and retrieve operations maintain a distributed persistent data repository.
3. Broadcast and multicast support information dissemination to multiple recipients.

Because of their symmetric, serverless nature, these networks are termed *P2P* networks. Below, we often refer to hosts participating in the network as *peers*.

The most influential mechanism in this area is *consistent hashing*, pioneered in a paper by Karger et al. [21]. The idea is roughly the following. Frequently, a good way of arranging a lookup directory is a hash table, giving a fast $O(1)$-complexity data access. In order to scale and provide highly available lookup services, we partition the hash table and assign different chunks to different servers. So, for example, if the hash table has entries 1 through n, and there are k participating servers, we can have each server select a virtual identifier from 1 to n at random. Server i will then be responsible for key values that are closer to i than to any other server identifier. With a good randomization of the hash keys, we can have a more or less balanced distribution of information between our k servers. In expectation, each server will be responsible for (n/k) keys. Furthermore, the departure/arrival of a server perturbs only one or two other servers with adjacent virtual identifiers.

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

A network of servers that implement consistent hashing is called a *distributed hash table* (DHT). Many current-generation resource sharing networks, and virtually all academic research projects in the area, are built around a DHT idea.

The challenge in maintaining DHTs is two-fold:

Overlay routing Given a hash key i, and starting from any node r in the network, the problem is to find the server s whose key range contains i. The key name i bears no relation to any real network address, such as the IP address of a node, and therefore we cannot use the underlying IP infrastructure to locate s. An overlay routing network links the nodes, and provides them with a routing protocol, such that r can route toward s using the routing target i.

Dynamic maintenance DHTs must work in a highly dynamic environment in which the size of the network is not known a priori, and where there are no permanent servers for maintaining either the hash function or the overlay network (all servers are assumed to be ephemeral). This is especially acute in P2P settings, where the servers are transient users who may come and go as they wish. Hence, there must be a decentralized protocol, executed by joining and leaving peers, that incrementally maintains the structure of the system. Additionally, a joining peer should be able to correctly execute this protocol while initially only having knowledge of a single, arbitrary participating network node.

One of the first overlay network projects was **Chord** [35], after which this encyclopedia entry is named (2001; Stoica, Morris, Karger, Kaashoek, Balakrishnan). More details about Chord are given below.

Key Results

The P2P area is very dynamic and rapidly evolving. The current entry provides a mere snapshot, covering dominant and characteristic strategies, but not offering an exhaustive survey.

Unstructured Overlays

Many of the currently deployed widespread resource-sharing networks have little or no particular overlay structure. More specifically, early systems such as Gnutella version 0.4 had no overlay structure at all, and allowed every node to connect to other nodes arbitrarily. This resulted in severe load and congestion problems.

Two-tier networks were introduced to reduce communication overhead and solve the scalability issues that early networks like Gnutella version 0.4 had. Two-tier networks consist of one tier of relatively stable and powerful nodes, called servers (superpeers, ultrapeers), and a larger tier of clients that search the network though servers. Most current networks, including Edonkey/Emule, KaZaa, and Gnutella, are built using two tiers. Servers provide directory store and search facilities. Searching is either limited to servers to which clients directly connect (eDonkey/eMule) or done by limited-depth flooding among the servers (Gnutella). The two-tier design considerably enhances the scalability and reliability of P2P networks. Nevertheless, the connections among servers and between clients/servers is done in a completely ad hoc manner. Thus, these networks provide no guarantee for the success of searches, nor a bound on their costs.

Structured Overlays Without Locality Awareness

Chord

The Chord system was built at MIT and is currently being developed under FNSF's **IRIS** project (http://project-iris.net/). Several aspects of the Chord [35] design have influenced subsequent systems. We briefly explain the core structure of Chord here. Nodes have binary identifiers, assigned uniformly at random. Nodes are arranged in a linked ring according to their virtual identifiers. In addition, each node has shortcut links to other nodes along the ring, link i to a node 2^i away in the virtual identifier space.

In this way, one can move gradually to the target by decreasing the distance by half at every step. Routing takes on average $\log n$ hops to reach any target, in a network containing n nodes. Each node maintains approximately $\log n$ links, providing the ability to route to geometrically increasing distances.

Constant Per-Node State

Several overlay network algorithms were developed with the goal of pushing the amount of network state kept by each node in the overlay to a minimum. We refer to the state kept by a node as its *degree*, as it mostly reflects the number of connections to other nodes. **Viceroy** [23] was the first to demonstrate a dynamic network in which each node stores only five links to other network nodes, and routes to any other node in a logarithmic number of hops, $\log n$ for a network of n nodes. Viceroy provided a dynamic emulation of a butterfly network (see [11] for a textbook exposition of interconnect networks like butterfly). Later, several emulations of De Bruijn networks emerged, including the generic one of Abraham et al. (AAABMP) [1], the **distance halving** network [26], **D2B** [13], and **Koorde** [20]. Constant-degree overlay networks are too fragile for practical purposes, and may easily degrade in performance or even partition in the face of failures. A study of overlay networks under churn demonstrated these points [18]. Indeed, to the best of our knowledge, none of these constant-degree networks were built. Their main contribution, and the main reason for mentioning these works here, is to know that it is possible in principle to bring the per-node state to a bare, small constant.

Content Addressable Network

The Content Addressable Network (CAN) [31] developed at ICSI builds the network as virtual d-dimensional space, giving every node a d-dimensional identifier. The routing topology resembles a d-dimensional torus. Routing is done by following the Euclidean coordinates in every dimension, yielding a $d n^{1/d}$ hop routing strategy. The parameter d can be tuned by the network administrator. Note that for $d = \log n$, CAN's features are the same as in Chord, namely, logarithmic degree and logarithmic routing hop count.

Overlay Routing Inspired by "Small-World" Networks

The **Symphony** [24] algorithm emulates routing in a small world. Nodes have k links to nodes whose virtual identifiers are chosen at random according to a routable small-world distribution [22]. With k links, Symphony is expected to find a target in $\log^2 / k$ hops.

Overlay Networks Supporting Range Queries

One of the deficiencies of DHTs is that they support only exact key lookup; hence, they do not address well the need to locate a range of keys, or to have a fuzzy search, e.g., search for any key that matches some prefix. **SkipGraphs** [4] and the **SkipNet** [19] scheme from Microsoft (project Herald) independently developed a similar DHT based on a randomized skip list [28] that supports range queries over a distributed network. The approach in both of these networks is to link objects into a double-linked list, sorted by object names, over which "shortcut" pointers are built. Pointers from each object skip to a geometric sequence of distances in the sorted list, i.e., the first pointer jumps two items away, the second four items, and so on, up to pointer $\log n - 1$, which jumps over half of the list. Logarithmic, load-balanced lookup is achieved in this scheme in the same manner as in Chord. Because the identifier space is sorted by object names, rather than hash identifiers, ranges of objects can be scanned efficiently simply by routing to the lowest value in the range; the remaining range nodes reside contiguously along the ring.

By prefixing organization names to object names, SkipNet achieves contiguity of nodes belonging to a single organization along the ring, and the ability to map objects on nodes in their local organizations. In this way SkipNet achieves resource proximity and isolation the only system besides RP [33] to have this feature.

Whereas the SkipGraphs work focuses on randomized load-balancing strategies and proofs, the SkipNet system considers issues of dynamic

P2P, Table 1 Comparison of various measures of lookup schemes with no locality awareness

Overlay lookup scheme	Topology resemblance	Hops	Degree
Chord	Hypercube	$\log n$	$\log n$
Viceroy	Butterfly	$\log n$	5
AAABMP, Distance-halving, Koorde, D2B	De Bruijn	$\log n$	4
Symphony	Small world	$\log^2 n/k$	k
SkipGraphs/SkipNet	Skip list	$\log n$	$\log n$
CAN	Torus	$dn^{1/d}$	d

maintenance, variable base sizes, and adopts the locality-awareness strategy of Pastry [33], which is described below.

Summary of Non-Locality-Aware Networks

Each of the networks mentioned above is distinct in one or more of the following properties: The (intuitive) emulated **topology**; the expected number of **hops** required to reach a target; and the per-node **degree**. Table 1 summarizes these properties.

Locality Awareness

The problem with the approaches listed above is that they ignore the proximity of nodes in the underlying networks, and allow hopping back and forth across large physical distances in search of content. Recent studies of scalable content exchange networks [17] have indicated that up to 80 % of Internet searches could be satisfied by local hosts within one's own organization. Therefore, even one far hop might be too costly. The next systems we encounter consider proximity relations among nodes in order to obtain locality awareness, i.e., that lookup costs are proportional to the actual distance of interacting parties.

Growth-Bounded Networks

Several locality-aware lookup networks were built around a bit-fixing protocol that borrows from the seminal work of Plaxton et al. [27] (PRR). The *growth bounded* network model for which this scheme is aimed views the network as a metric space, and assumes that the densities of nodes in different parts of the network are not terribly different. The PRR [27] lookup scheme uses prefix routing, similar to Chord. It differs from Chord in that a link for flipping the ith identifier bit connects with any node whose length-i prefix matches the next hop. In this way, the scheme favors the closest one in the network. This strategy builds *geometric routing*, whose characteristic is that the routing steps toward a target increase geometrically in distance. This is achieved by having large flexibility in the choice of links for each prefix at the beginning of a route, and narrowing it down as the route progresses. The result is an overlay routing scheme that finds any target with a cost that is proportional to the shortest-distance route.

The systems that adopt the PRR algorithm are **Pastry** [33], **Tapestry** [36], and **Bamboo** [32]. A very close variant is **Kademlia** [25], in which links are symmetric. It is worth mentioning that the LAND scheme [2] improves PRR in providing a nearly optimal guaranteed locality guarantee; however, LAND has not been deployed.

Applications

Caching

The Coral network [14] from NYU, built on top of DSHT [15], has been operational since around 2004. It provides free content delivery services on top of the PlanetLab-distributed test bed [9], similar to the commercial services offered by the Akamai network. People use it to provide multiple, fast access points to content they wish to publish on the Web.

Coral optimizes access locality and download rate using locality-aware lookup provided by DSHT. Within Coral, DSHT is utilized to support locality-aware object location in two applica-

tions. First, Coral contains a collection of HTTP proxies that serve as content providers; DSHT is used by clients for locating a close-by proxy. Second, proxy servers themselves use DSHT to locate a near-by copy of content requested by the client, thus making use of copies of the content that are stored in the network, rather than going to the source of the content.

Multicast

Several works deploy an event notification or publish–subscribe service over an existing routing overlay by building reverse-routing multicast paths from a single "target" to all "sources." For example, multicast systems built in this way include the Bayeux network [38], which is built over Tapestry [36], and SCRIBE [5], which is built over Pastry. In order to publish a file, the source advertises using flooding a tuple which contains the semantic name of a multicast session and a unique ID. This tuple is hashed to obtain a node identifier which becomes the session root node. Each node can join this multicast session by sending a message to the root. Nodes along the way maintain membership information, so that a multicast tree is formed in the reverse direction. The file content (and any updates) is flooded down the tree. Narada [8] is built with the same general architecture, but differs in its choice of links, and the maintenance of data.

Routing Infrastructure

A DHT can serve well to store routing and (potentially dynamic) location information of virtual host names. This idea has been utilized in a number of projects. A naming system for the Internet called CoDoNS [30] was built at Cornell University over the BeeHive overlay [29]. CoDoNS provides a safety net and is a possible replacement for the Domain Name System, the current service for looking up host names. Support for virtual IPv6 network addresses is provided in [37] by mapping names to their up-to-date, reachable IPv4 address. The Internet Indirection Infrastructure [34] built at the University of California, Berkeley provides support for virtual Internet host addresses that allows mobility.

Collaborative Content Delivery

Recent advances provide collaborative content delivery solutions that address both load balance and resilience via striping. The content is split into pieces (quite possibly with some redundancy through error-correcting codes). The source pushes the pieces of the file to an initial group of nodes, each of which becomes a source of a distribution tree for its piece, and pushes it to all other nodes. These works demonstrate clearly the advantages of data striping, i.e., of simultaneously exchanging stripes of data, over a tree-based dissemination of the full content.

SplitStream [6] employs the Pastry routing overlay in order to construct multiple trees, such that each participating node is an inner node in only one tree. It then supports parallel download of stripes within all trees. SplitStream [6] strives to obtain load balancing between multicast nodes. It achieves that by splitting the published content into several parts, called stripes, and publishing each part separately. Each stripe is published using a tree-based multicast. The workload is divided between the participating nodes by sending each stripe using a different multicast tree. Load balance is achieved by carefully choosing the multicast trees so that each node serves as an interior node in at most one tree. This reduces the number of "free riders" who only receive data.

A very popular file-distribution network is the BitTorrent system [10]. Nodes in BitTorrent are divided into *seed* nodes and *clients*. Seed nodes contain the desired content in full (either by being original providers, or by having completed a recent download of the content). Client nodes connect with a seed node or several seed nodes, as well as a *tracker* node, whose goal is to keep track of currently downloading clients. Each client selects a group (currently, of size about 20) of other downloading clients, and exchanges chunks of data obtained from the seed(s) with them. BitTorrent employs several intricate strategies for selecting which chunks to request from what other clients, in order to obtain fair load sharing of the content distribution and, at the same time, achieve fast download.

BitTorrent currently does not contain P2P-searching facilities. It relies on central sites known as "trackers" to locate content, and to coordinate the BitTorrent download process. Recent announcements by Bram Cohen (the creator of BitTorrent) and creators of other BitTorrent clients state that new protocols based on BitTorrent will be available soon, in which the role of trackers is eliminated, and searching and coordination is done in a completely P2P manner.

Experience with BitTorrent and similar systems indicates that the main problem with this approach is that towards the end of a download, many peers may be missing the same rare chunks, and the download slows down. Fairly sophisticated approaches were published in an attempt to overcome this issue.

Recently, a number of works at Microsoft Research have demonstrated the benefits of network coding in efficient multicast, e.g., [7] and Avalanche [16]. We do not cover these techniques in detail here, but only briefly state the principal ideas that underlie them.

The basic approach in network coding is to re-encode all the chunks belonging to the file, so that each one that is shared is actually a linear combination of all the pieces. The blocks are then distributed with a description of the content. Once a node obtains these re-encoded chunks, it can generate new combinations from the ones it has, and can send those out to other peers. The main benefit is that peers can make use of any new piece, instead of having to wait for specific chunks that are missing. This means no one peer can become a bottleneck, since no piece is more important than any other. Once a peer collects sufficiently many such chunks, it may use them to reconstruct the whole file.

It is worth noting that in unstructured settings, it was recently shown that network coding offers no advantage [12].

Cross-References

- ▸ Geometric Spanners
- ▸ Routing
- ▸ Sparse Graph Spanners

Recommended Reading

1. Abraham I, Awerbuch B, Azar Y, Bartal Y, Malkhi D, Pavlov E (2003) A generic scheme for building overlay networks in adversarial scenarios. In: Proceedings of the international parallel and distributed processing symposium (IPDPS 2003)
2. Abraham I, Malkhi D, Dobzinski O (2004) LAND: stretch $(1 + \varepsilon)$ locality aware networks for DHTs. In: Proceedings of the ACM-SIAM symposium on discrete algorithms (SODA'04)
3. Abraham I, Badola A, Bickson D, Malkhi D, Maloo S, Ron S (2005) Practical locality-awareness for large scale information sharing. In: The 4th annual international workshop on peer-to-peer systems (IPTPS'05)
4. Aspnes J, Shah G (2003) Skip graphs. In: Fourteenth annual ACM-SIAM symposium on discrete algorithms, Baltimore, pp 384–393
5. Castro M, Druschel P, Rowstron A (2002) Scribe: a large-scale and decentralised application-level multicast infrastructure. IEEE J Sel Areas Commun (JSAC) 20(8):1489–1499, Spec Issue Netw Support Multicast Commun, ISSN:0733-8716
6. Castro M, Druschel P, Kermarrec A-M, Nandi A, Rowstron A, Singh A (2003) Splitstream: high-bandwidth multicast in a cooperative environment. In: SOSP'03
7. Chou P, Wu Y, Jain K (2004) Network coding for the internet. In: IEEE communication theory workshop
8. Chu Y, Rao SG, Zhang H (2000) A case for end system multicast. In: Proceedings of ACM SIGMETRICS, Santa Clara, pp 1–12
9. Chun B, Culler D, Roscoe T, Bavier A, Peterson L, Wawrzoniak M, Bowman M (2003) Planetlab: an overlay testbed for broadcoverage services. ACM SIGCOMM Comput Commun Rev 33:3–12
10. Cohen B (2003) Incentives build robustness in bittorrent. In: Proceedings of P2P economics workshop
11. Cormen TH, Leiserson CE, Rivest RL (1990) Introduction to algorithms. MIT
12. Fernandess Y, Malkhi D (2006) On collaborative content distribution using multi-message gossip. In: Twentieth IEEE international parallel and distributed processing symposium (IPDPS'06), Greece
13. Fraigniaud P, Gauron P (2003) The content-addressable network D2B. Technical report 1349, LRI, University Paris-Sud
14. Freedman MJ, Freudenthal E, Mazières D (2004) Democratizing content publication with coral. In: Proceedings of the 1st USENIX/ACM symposium on networked systems design and implementation (NSDI'04)
15. Freedman MJ, Mazières D (2003) Sloppy hashing and self-organizing clusters. In: Proceedings of the 2nd international workshop on peer-to-peer systems (IPTPS'03)
16. Gkantsidis C, Rodriguez P (2005) Network coding for large scale content distribution. In: IEEE/INFOCOM

17. Gummadi KP, Dunn RJ, Saroiu S, Gribble SD, Levy HM, Zahorjan J (2003) Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: Proceedings of the nineteenth ACM symposium on Operating systems principles, ACM, pp 314–329
18. Gummadi K, Gummadi R, Gribble S, Ratnasamy S, Shenker S, Stoica I (2003) The impact of DHT routing geometry on resilience and proximity. In: Proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications, ACM, pp 381–394
19. Harvey NJA, Jones MB, Saroiu S, Theimer M, Wolman A (2003) Skipnet: a scalable overlay network with practical locality properties. In: Proceedings of fourth USENIX symposium on internet technologies and systems (USITS'03)
20. Kaashoek F, Karger DR (2003) Koorde: a simple degree-optimal hash table. In: 2nd international workshop on peer-to-peer systems (IPTPS'03)
21. Karger D, Lehman E, Leighton FT, Levine M, Lewin D, Panigrahy R (1997) Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: Proceedings of the 29th annual ACM symposium on theory of computing (STOC'97), pp 654–663
22. Kleinberg J (2000) The small-world phenomenon: an algorithmic perspective. In: Proceedings of the 32nd ACM symposium on theory of computing (STOC'00), pp 163–170
23. Malkhi D, Naor M, Ratajczak D (2002) Viceroy: a scalable and dynamic emulation of the butterfly. In: Proceedings of the 21st ACM symposium on principles of distributed computing (PODC'02), pp 183–192
24. Manku GS, Bawa M, Raghavan P (2003) Symphony: distributed hashing in a small world. In: Proceedings of the 4th USENIX symposium on internet technologies and systems (USITS'03), pp 127–140
25. Maymounkov P, Mazières D (2002) Kademlia: a peer-to-peer information system based on the XOR-metric. In: Proceedings of the 1st international workshop on peer-to-peer systems (IPTPS'02), pp 53–65
26. Naor M, Wieder U (2003) Novel architectures for p2p applications: the continuous-discrete approach. In: The fifteenth annual ACM symposium on parallelism in algorithms and architectures (SPAA'03)
27. Plaxton C, Rajaraman R, Richa A (1997) Accessing nearby copies of replicated objects in a distributed environment. In: Proceedings of the ninth annual ACM symposium on parallel algorithms and architectures (SPAA'97), pp 311–320
28. Pugh W (1989) Skip lists: a probabilistic alternative to balanced trees. In: Workshop on algorithms and data structures, pp 437–449
29. Ramasubramanian V, Sirer EG (2004) Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In: Proceedings of networked system design and implementation (NSDI)
30. Ramasubramanian V, Sirer EG (2004) The design and implementation of a next generation name service for the internet. In: Proceedings of SIGCOMM
31. Ratnasamy S, Francis P, Handley M, Karp R, Shenker S (2001) A scalable content-addressable network. In: Proceedings of the ACM SIGCOMM 2001 Technical Conference
32. Rhea S, Geels D, Roscoe T, Kubiatowicz J (2003) Handling churn in a dht. Technical report, UCB//CSD-03-1299. The University of California, Berkeley
33. Rowstron A, Druschel P (2001) Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM international conference on distributed systems platforms (Middleware), pp 329–350
34. Stoica I, Adkins D, Zhuang S, Shenker S, Surana S (2002) Internet indirection infrastructure. In: Proceedings of ACM SIGCOMM, pp 73–88
35. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H (2001) Chord: a scalable peer-to-peer lookup service for internet applications. In: Proceedings of the SIGCOMM
36. Zhao BY, Huang L, Stribling J, Rhea SC, Joseph AD, Kubiatowicz J (2003) Tapestry: a resilient global-scale overlay for service deployment. IEEE J Sel Areas Commun
37. Zhou L, van Renesse R, Marsh M (2002) Implementing IPv6 as a peer-to-peer overlay network. In: Proceedings of the 21st IEEE symposium on reliable distributed systems (SRDS'02), p 347
38. Zhuang SQ, Zhao BY, Joseph AD, Katz RH, Kubiatowicz J (2001) Bayeux: an architecture for scalable and fault-tolerant widearea data dissemination. In: Proceedings of the eleventh international workshop on network and operating system support for digital audio and video (NOSSDAV'01)

PAC Learning

Joel Ratsaby
Department of Electrical and Electronics Engineering, Ariel University of Samaria, Ariel, Israel

Keywords

Computational learning theory; Probably approximately correct learning

Years and Authors of Summarized Original Work

1984; Valiant

Problem Definition

Valiant's work defines a model for representing the general problem of learning a Boolean concept from examples. The motivation comes from classical fields of artificial intelligence [2], pattern classification [8], and machine learning [13]. Classically, these fields have employed numerous heuristics for representing knowledge and defining criteria by which computer algorithms can learn. The pioneering work of [16, 17] provided the leap from heuristic-based approaches to a rigorous statistical theory of pattern recognition (see also [1, 7, 14]). Their main contribution was the introduction of probabilistic upper bounds on the generalization error which hold uniformly over a whole class of concepts. Valiant's main contribution is in formalizing this probabilistic theory into a general model for computational inference. This model which is known as the *Probably Approximately Correct* (PAC) model of learnability is concerned with computational complexity of learning. In his formulation, learning is depicted as an interaction between a teacher and a learner with two main procedures, one which provides randomly drawn examples x of the concept c that is being learned and the second acts as an oracle which provides the correct classification label $c(x)$. Based on a finite number of such examples drawn identically and independently according to *any* fixed probability distribution, the aim of the learner is to infer an approximation of c which is correct with high confidence. Using the terminology of [12] suppose X denotes the space of instances, i.e., objects which a learner can obtain as training examples. A *concept* over X is a Boolean mapping from X to $\{0,1\}$. Let $\mathbb{P}$ be any fixed probability distribution over X and c a fixed *target* concept to be learned. For any hypothesis concept h over X define by $L(h) = \mathbb{P}(c(x) \neq h(x))$ the *error* of h, i.e., the probability that h disagrees with c on a test instance x which is drawn according to $\mathbb{P}$. Then according to Valiant, an algorithm $\mathbb{A}$ for learning c is one which runs in time t and with a sample of size m where both t and m are polynomials with respect to some parameters (to be specified below) and produces a hypothesis concept h such that with high confidence $L(h)$ is small.

Key Results

The main result of Valiant's work is a formal definition of what constitutes a *learnable* problem. Formally, this is stated as follows: let $\mathcal{H}$ be a class of concepts over X. Then $\mathcal{H}$ is *learnable* if there exists an algorithm $\mathbb{A}$ with the following property: for every possible target concept $c \in \mathcal{H}$, for every probability distribution $\mathbb{P}$ on X (this is sometimes referred to as the "distribution-independence" assumption), for all values of a confidence parameter $0 < \delta < 1/2$ and an approximation accuracy parameter $0 < \epsilon < 1/2$, if $\mathbb{A}$ receives as input the value of δ, ϵ and a sample $S = \{(x_i, c(x_i))\}_{i=1}^{m}$ of cardinality m (which may depend on ε and δ) which consists of examples x_i that are randomly drawn according to $\mathbb{P}$ and labeled by an oracle as $c(x_i)$, then with probability $1 - \delta$, $\mathbb{A}$ outputs a hypothesis concept $h \in \mathcal{H}$ such that the error $L(h) \leq \epsilon$. That ϵ can be arbitrarily close to zero follows from what is known as the "noise-free" assumption, i.e., that the labels comprise the true value of the target concept. If $\mathbb{A}$ runs in time t and if t and m are polynomial in $1/\epsilon$ and $1/\delta$ (and possibly other relevant parameters, such as n if the space of instance X is $\{0,1\}^n$ or $\mathbb{R}^n$), then $\mathcal{H}$ is *efficiently* PAC learnable.

Valiant has shown that the following classes are all efficiently PAC learnable: the class of conjunctive normal form expressions with a bounded number of literals in each clause, the class of monotone disjunctive normal form expressions (here the learner requires in addition to S also an oracle that can answer membership queries, i.e., provide the true label $c(x)$ for an x in question), and the class of arbitrary expressions in which each variable occurs just once (using more powerful oracles). The work following Valiant's

paper (see [11] for references) has shown that the classes of k-DNF, k-CNF, and k-decision lists are efficiently PAC learnable for each fixed k. Under suitable complexity-theoretic hardness assumptions, the class of concepts in the form of a disjunction of two conjunctions is not PAC learnable and neither is the class of existential conjunctive concepts on structural instance spaces with two objects. Linear threshold concepts (perceptrons) are PAC learnable on both Boolean and real-valued instance spaces, but the class of concepts in the form of a conjunction of two linear threshold concepts is not PAC learnable. The same holds for disjunctions and linear thresholds of linear thresholds (i.e., multilayer perceptrons with two hidden units). If the weights are restricted to 1 and 0 (but the threshold is arbitrary), then linear threshold concepts on Boolean instance spaces are not PAC learnable.

It should be noted that the notion of PAC learnability discussed throughout this entry is sometimes referred to as "proper" PAC learnability because of the requirement that, when learning a concept class $\mathcal{H}$, the learning algorithm must output a hypothesis that also belongs to $\mathcal{H}$. Several of the negative results mentioned above can be circumvented in a model of "improper" PAC learning, where the learning algorithm is allowed to output hypotheses from a broader class of functions than $\mathcal{H}$. This is sometimes referred to as agnostic PAC learnability (see [15], Ch. 3, [12] and the proceedings of the COLT conferences for many results of this type).

Applications

Valiant's paper is a milestone in the history of the area known as *Computational Learning Theory* (see proceedings of COLT conferences). The PAC model has been criticized in that the distribution-independence assumption and the notion of target concepts with noise-free training data are unrealistic in practice, e.g., in machine learning and AI. There has thus been much work on learning models that relax several of the assumptions in Valiant's PAC model. These include models which allow noisy labels or remove the assumptions on the independence of training examples, relax the assumption on the probability distribution to be fixed, allow the bounds to be distribution dependent, permit the training sample to be picked by the learner and labeled by the oracle instead of the random sample or chosen by a helpful teacher, allow learning regression, and use generalized loss functions. For references, see Sec. 2.6 of [1] and Ch. 3 of [15]. An important follow-up of Valiant's model was the work of [6] who unified his model with the uniform convergence results of [16]. They showed the important dependence between the notion of learnability and certain combinatorial properties of concept classes, one of which is known as the Vapnik-Chervonenkis (VC) dimension (see Sec. 3.4 of [1] for history on the VC-dimension).

Cross-References

- ▸ Attribute-Efficient Learning
- ▸ Hardness of Proper Learning
- ▸ Learning Constant-Depth Circuits
- ▸ Learning DNF Formulas
- ▸ Learning with Malicious Noise
- ▸ Learning with the Aid of an Oracle

Recommended Reading

For a recommended collection of works on the PAC model and its extensions see [5–7].

1. Anthony M, Bartlett PL (1999) Neural network learning: theoretical foundations. Cambridge University Press, Cambridge, U.K
2. Barr A, Feigenbaum EA, Cohen PR (eds) (1981) The handbook of artificial intelligence, vol 1. William Kaufmann, Los Altos
3. Barr A, Feigenbaum EA, Cohen PR (eds) (1982) The handbook of artificial intelligence, vol 2. William Kaufmann, Los Altos
4. Barr A, Feigenbaum EA, Cohen PR (eds) (1989) The handbook of artificial intelligence, vol 4. Addison-Wesley. Reading
5. Cohen PR, Feigenbaum EA, Cohen PR (eds) (1982) The handbook of artificial intelligence, volu 3. HeurisTech Press/William Kaufmann, Stanford/Los Altos

6. Blumer A, Ehrenfeucht A, Haussler D, Warmuth M (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM 36(4):929–965
7. Devroye L, Gyorfi L, Lugosi G (1996) A probabilistic theory of pattern recognition. Springer, New York
8. Duda RO, Hart PE, Stork DG (2000) Pattern classification. Wiley, New York
9. Haussler D (1990) Applying Valiant's learning framework to AI concept learning problems. In: Michalski R, Kodratoff Y (eds) Machine learning: an artificial intelligence approach, vol III. Morgan Kaufmann, Los Altos, pp 641–669
10. Haussler D (1992) Decision theoretic generalizations of the PAC model for neural net and other learning applications. Inf Comput 100(1):78–150
11. Haussler D (1996) Probably approximately correct learning and decision-theoretic generalizations. In: Smolensky P, Mozer M, Rumelhart D (eds) Mathematical perspectives on neural networks. L. Erlbaum Associates, Mahwah, pp 651–718
12. Kearns MJ, Vazirani UV (1997) An introduction to computational learning theory. MIT, Cambridge
13. Mitchell T (1997) Machine learning. McGraw Hill, New York
14. Pearl J (1979) Capacity and error-estimates for boolean classifiers with limited complexity. IEEE Trans Pattern Recognit Mach Intell PAMI-1(4): 350–356
15. Shalev-Shwartz S, Ben-David S (2014) Understanding machine learning; From theory to algorithms. Cambridge University Press, New York
16. Vapnik VN (1982) Estimations of dependences based on statistical data. Springer, New York
17. Vapnik VN, Chervonenkis AY (1971) On the uniform convergence of relative frequencies of events to their probabilities. Theory Probab Appl 16:264–280

Packet Routing

Lenore J. Cowen
Department of Computer Science, Tufts University, Medford, MA, USA

Keywords

Job shop scheduling; Store-and-forward routing

Years and Authors of Summarized Original Work

1988; Leighton, Maggs, Rao

Problem Definition

A collection of packets need to be routed from a set of specified sources to a set of specified destinations in an arbitrary network. Leighton, Maggs and Rao [5] looked at a model where this task is divided into two separate tasks: the first is the *path selection* task, where for each specified packet i with source s_i and packet destination t_i, a simple (meaning edges don't repeat) path P_i through the network from s_i to t_i is pre-selected. Packets traverse the network in a *store and forward* manner: each time a packet is forwarded it travels along the next link in the pre-selected path. It is assumed that only one packet can cross each individual link at each given global (synchronous) timestep. Thus, when there is contention for a link, packets awaiting traversal are stored in the local link's queue (special source and sink queues of unbounded size are also defined that store packets at their origins and destinations). Thus, the second task, and the focus of the Leighton, Maggs and Rao result (henceforth called the LMR result) is the *scheduling* task: a determination, when a link's queue is not empty, of which packet gets to traverse the link in the next timestep (where it is assumed to immediately join the link queue for its next hop). The goal is to schedule the packets so that the *maximum* time that it takes any packet to reach its destination is minimized.

There are two parameters of the network together with the pre-selected paths that are clearly relevant. One is the *congestion* c, defined as the maximum number of paths that all use the same link. The other is the *dilation* d, which is simply the length of the longest path that any packet traverses in the network. Clearly each of c and d is a lower-bound on the length of any schedule that routes all the packets to their destinations. It is easy to see that a schedule of length at most cd always exists. In fact, any schedule that never lets a link go idle if there is a packet that can use that link at that timestep is guaranteed to terminate in cd steps, because each packet traverses at most d links, and at any link can be delayed by at most $c - 1$ other packets.

Key Results

The surprising and beautiful result of LMR is as follows:

Theorem ([5]) *For any network G with a pre-specified set of paths P with congestion c and dilation d, there exists a schedule of length $O(c+d)$, where the queue sizes at each edge are always bounded by a constant.*

The original proof of the LMR paper is non-constructive. That is, it uses the Local Lemma [3] to prove the existence of such a schedule, but does not give a way to find it. In his book [10], Scheideler showed that in fact, a $O(c + d)$ schedule exists with edge queue sizes bounded by 2 (and gave a simpler proof of the original LMR result). A subsequent paper of Leighton, Maggs and Richa in 1999 [6] provides a constructive version of the original LMR paper as follows:

Theorem ([6]) *For any network G with a pre-specified set of paths P with congestion c and dilation d, there exists a schedule of length $O(c + d)$. Furthermore, such a schedule can be found in $O(p \log^{1+\epsilon} p \log^*(c + d))$ time for any $\epsilon>0$, where p is the sum of the lengths of the paths taken by the packets and ε is incorporated into the constant hidden by the big-O in the schedule length.*

The algorithm in the paper is a randomized one, though the authors claim that it can be derandomized using the method of conditional probabilities. However, even though the algorithm of Leighton, Maggs and Richa is constructive, it is still an offline algorithm: namely, it requires full knowledge of all packets in the network and the precise paths that each will traverse in order to construct the schedule. The original LMR paper also gave a simple randomized *online* algorithm, that, by assigning delays to packets independently and uniformly at random from an appropriate interval, results in a schedule which is much better than greedy schedules, though not as good as the offline constructions.

Theorem ([5]) *There is a simple randomized online algorithm for producing, with high probability, a schedule of length $O(c + d \log(Nd))$ using queues of size $O(\log(Nd))$, where c is the congestion, d is the dilation, and N is the number of packets.*

In the special case where it is assumed that all packets follow shortest paths in the network, Meyer, auf der Heide and Vöcking produced a simple randomized online algorithm that produces, with high probability, a schedule of length $O(c + d + \log Nd)$ steps, but queues can be as large as $O(c)$ [7]. For arbitrary paths, the LMR online result was ultimately improved to $O(c + d + \log^{1+\epsilon} N)$ steps, for any $\epsilon>0$ with high probability, in a series of two papers by Rabani and Tardos [9], and Rabani and Ostrovsky [8]. Online protocols have also been studied in a setting where additional packets are dynamically injected into the network in adversarial settings, see [10] for a survey.

The discussion is briefly returned to the first task, namely to pre-construct the set of paths. Clearly, the goal is to find, for a particular set of packets with pre-specified sources and destinations, a set of paths that minimizes $c + d$. Srinivasan and Teo [12] designed an off-line algorithm that produces a set of paths whose $c + d$ is provably within a constant factor of optimal. Together with the offline LMR result, that gives a constant-factor approximation problem for the offline store-and-forward packet routing problem. Note that the approach of trying to minimize $c + d$ rather than c alone seems crucial; producing schedules within a constant factor of optimal congestion c is hard, and in fact has been shown to be related to the integrality gap for multicommodity flow [1, 2].

Applications

Network Emulations

Typically, a guest network G is emulated by a host network H by embedding G into H. Nodes of G are mapped to nodes of H, while edges of G are mapped to paths in H. If P is the set of

e paths (each corresponding to an edge in the guest network G), the congestion and dilation can be defined analogously as in the main result for the set of paths P, namely c denotes the maximum number of paths that use any one edge of H, and d is the length of the longest path in P. In addition, the *load* l is defined to be the maximum number of nodes in G that are mapped to a single node of H. Once G is embedded in H, H can emulate G as follows: Each node of H emulates the local computations performed by the l (or fewer) nodes mapped to it in $O(l)$ time. Then for each packet sent along an edge of G, H sends a packet along the corresponding path in the embedding; using the offline LMR result this takes $O(c + d)$ steps. Thus, H can emulate each step of G in $O(c + d + l)$ steps.

Job Shop Scheduling

Consider a scheduling problem with jobs $j_1, \ldots j_r$ and machines $m_1, \ldots, m_s$ for which each job must be performed on a specified sequence of machines (in a specified order). Assume each job spends unit time on each machine, and that no machine has to work on any job more than once (In the language of job-shop scheduling, this is the *non-preemptive, acyclic, job-shop scheduling problem*, with unit jobs). There is a mapping of sequences of machines to paths and jobs to packets so that this becomes an encoding of the main packet routing problem, where if c is now to be the maximum number of jobs that have to be run on any one machine, and d to be the maximum number of different machines that work on any single job, there becomes $O(c)$ congestion and $O(d)$ dilation for the corresponding packet-routing instance. Then the offline LMR result shows that there is a schedule that completes all jobs in $O(c + d)$ steps, where in addition, each job waits at most a constant number of steps in between consecutive machines (and the queue of jobs waiting for any particular machine will always be bounded by a constant). Similar techniques to those developed in the LMR paper have subsequently been applied to more general instances of Job-Shop Scheduling; see [4, 11].

Open Problems

The main open problem is whether there is a randomized *online* packet scheduling that matches the offline LMR bound of $O(c + d)$. The bound of [8] is close, but still grows logarithmically with the total number of packets.

For job shop scheduling, it is unknown whether the constant-factor approximation algorithm for the non-preemptive acyclic job-shop scheduling problem with unit length jobs implied by LMR can be improved to a PTAS. It is also unknown whether there is a constant-factor approximation in the case of arbitrary-length jobs.

Recommended Reading

1. Andrews M, Zhang L (2005) Hardness of the undirected congestion minimization problem. In: Proceedings of the 37th annual ACM symposium on theory of computing, pp 284–293
2. Chuzhoy J, Naor J (2004) New hardness results for congestion minimization and machine scheduling. In: Proceedings of the 36th annual ACM symposium on theory of computing. ACM, New York, pp 28–34
3. Erdös P, Lovász L (1975) Problems and results on 3-chromatic hypergraphs and some related questions. Colloq Math Soc János Bolyai 10:609–627
4. Goldberg LA, Patterson M, Srinivasan A, Sweedick E (2001) Better approximation guarantees for job-shop scheduling. SIAM J Discret Math 14(1):67–92
5. Leighton FT, Maggs BM, Rao SB (1994) Packet routing and jobs-hop scheduling in O(congestion+dilation) steps. Combinatorica 14(2):167–180
6. Leighton FT, Maggs BM, Richa AW (1999) Fast algorithms for finding O(congestion + dilation) packet routing schedules. Combinatorica 19(3):375–401
7. Meyer auf der Heide F, Vöcking B (1999) Shortest-path routing in arbitrary networks. J Algorithms 31(1):105–131
8. Ostrovsky R, Rabani Y (1997) Universal O (congestion + dilation + $\log^{1+\varepsilon} N$) local control packet switching algorithm. In: Proceedings of the twenty-ninth ACM symposium on theory of computing, pp 644–653
9. Rabani Y, Tardos E (1996) Distributed packet switching in arbitrary networks. In: The 28th ACM symposium on theory of computing, pp 366–376
10. Scheideler C (1998) Universal routing strategies for interconnection networks, vol 1390, Lecture Notes in Computer Science. Springer

11. Shmoys DB, Stein C, Wein J (1994) Improved approximation algorithms for shop scheduling problems. SIAM J Comput 23(3):617–632
12. Srinivasan A, Teo CP (2000) A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. SIAM J Comput 30(6):2051–2068

Packet Switching in Multi-queue Switches

Markus Schmidt
Institute for Computer Science, University of Freiburg, Freiburg, Germany

Keywords

Online packet buffering; Online packet routing

Years and Authors of Summarized Original Work

2004; Azar, Richter, Albers, Schmidt

Problem Definition

A multi-queue network switch serves m incoming queues by transmitting data packets arriving at m input ports through one single output port. In each time step, an arbitrary number of packets may arrive at the input ports, but only one packet can be passed through the common output port. Each packet is marked with a value indicating its priority in the Quality of Service (QoS) network. Since each queue has bounded capacity B and the rate of arriving packets can be much higher than the transmission rate, packets can be lost due to insufficient queue space. The goal is to maximize the throughput which is defined as the total value of transmitted packets. The problem comprises two dependent questions: buffer management, namely which packets to admit into the queues, and scheduling, i.e., which (FIFO) queue to use for transmission in each time step.

Two scenarios are distinguished: (a) unit packet value (All packets have the same value.), (b) arbitrary packet values.

The problem is considered as an online problem, i.e., at time step t, only the packet arrivals until t are known, but nothing about future packet arrivals. The online switch performance in QoS based networks is studied by using competitive analysis in which the throughput of the online algorithm is compared to the throughput of an optimal offline algorithm knowing the whole arrival sequence in advance.

If not stated otherwise, the admission control is assumed to allows preemption, i.e., packets once enqueued need not necessarily be transmitted, but can be discarded.

Problem 1 (Unit Value Problem) All packets have value 1. Since all packets are thus equally important, the admission control policies simplify: All arriving packets are to be enqueued; in the case of buffer overflow, it does not matter which packets are stored in the queue and which packets are discarded.

Problem 2 (General problem) Each packet has its individual value where usually a range $[1, \alpha]$ is given for all packets. A special case consists in the two value model where the values are restricted to $\{1, \alpha\}$.

Key Results

Unit Value Packets

Deterministic Algorithms

Theorem 1 ([1]) *For any buffer size B, the competitive ratio of each deterministic online algorithm is not smaller than* $(e_B + \frac{2}{B})/(e_B - 1 + \frac{1}{B}) \geq \frac{e}{e-1} \approx 1.58$ *where* $e_B = ((B+1)/B)^B$.

Theorem 2 ([4]) *Every work-conserving online algorithm is 2-competitive.*

Theorem 3 ([1]) *For any buffer size B, the competitive ratio of any greedy algorithm, which*

always serves a longest queue (LQF), is at least $2 - \frac{1}{B}$ *if* $m \gg B$.

Algorithm: *SGR* (Semi-Greedy) In each time step, the algorithm executes the first rule that applies to the current buffer configuration.

1. If there is a queue buffering more than $\lfloor B/2 \rfloor$ packets, serve the queue currently having the maximum load.
2. If there is a queue the hitherto maximum load of which is less than B, serve among these queues the one currently having the maximum load.
3. Serve the queue currently having the maximum load.

Ties are broken by choosing the queue with the smallest index. The hitherto maximum load is reset to 0 for all queues whenever all queues are unpopulated in *SGR*'s configuration.

Theorem 4 ([1]) *If* B *is even, then SGR is* $\frac{17}{9} \approx 1.89$*-competitive. If* B *is odd, then SGR is* $(\frac{17}{9} + \frac{\delta_B}{9})$*-competitive where* $\delta_B = \frac{2}{B+1}$.

Theorem 5 ([3]) *Algorithm* $E^{M^{\hat{EP}'}}$ *(not stated in detail due to space limitation), which is based on a water level algorithm and uses a fractional matching in an online constructed graph, achieves a competitiveness of* $e/(e-1)(1 + (\lfloor H_m + 1 \rfloor)/B)$*, where* H_m *denotes the* m^{th} *harmonic number. Thus,* $E^{M^{\hat{EP}'}}$ *is asymptotically* $\frac{e}{e-1}$*-competitive for* $B \gg \log m$.

Randomized Algorithms

Theorem 6 ([1]) *The competitive ratio of each randomized online algorithm is at least* $\varrho = 1.4659$ *for any buffer size* B *(* $\varrho = 1 + \frac{1}{\alpha+1}$ *where* α *is the unique positive root of* $e^{\alpha} = \alpha + 2$*).*

Theorem 7 (Generalizing technique [9]) *If there is a randomized* c*-competitive algorithm* A *for* $B = 1$*, then there is a randomized* c*-competitive algorithm* $\tilde{A}$ *for all* B.

Algorithm: *RS* (Random Schedule)

1. The algorithm uses m auxiliary queues $Q_1, \ldots, Q_m$ of sizes $B_1, \ldots, B_m$ (different buffer sizes at the distinct ports are allowed), respectively. These queues contain real numbers from the range (0,1), where each number is labeled as either marked or unmarked. Initially, these queues are empty.
2. Packet arrival: If a new packet arrives at queue q_i, then the algorithm chooses uniformly at random a real number from the range (0,1) that is inserted into queue Q_i and labeled as unmarked. If queue Q_i was full when the packet arrived, the number at the head of the queue is deleted prior to the insertion of the new number.
3. Packet transmission: Check whether queues $Q_1, \ldots, Q_m$ contain any unmarked number. If there are unmarked numbers, let Q_i be the queue containing the largest unmarked number. Change the label of the largest number to "marked" and select queue q_i for transmission. Otherwise (no unmarked number), transmit a packet from any non-empty queue if such exists.

Theorem 8 ([4]) *Randomized algorithm RS is* $\frac{e}{e-1} \approx 1.58$*-competitive.*

Algorithm: *RP* (Random Permutation) Let $\mathcal{P}$ be the set of permutations of $\{1, \ldots, m\}$, denoted as m-tuples. Choose $\pi \in \mathcal{P}$ according to the uniform distribution and fix it. In each transmission step, choose among the populated queues that one whose index is most to the front in the m-tuple π.

Theorem 9 ([9]) *Randomized algorithm RP is* $\frac{3}{2}$*-competitive for* $B = 1$*. By Theorem 7, there is a randomized algorithm* $\tilde{RP}$ *that is* $\frac{3}{2}$*-competitive for arbitrary* B.

Arbitrary Value Packets

Definition 1 A switching algorithm *ALG* is called *comparison-based* if it bases its decisions on the relative order between packet values (by performing only comparisons), with no regard to the actual values.

Theorem 10 (Zero-one principle [5]) *Let ALG be a comparison-based switching algorithm (deterministic or randomized). ALG is c-competitive if and only if ALG achieves a c-competitiveness for all packet sequences whose values are restricted to* $\{0, 1\}$ *for every possible way of breaking ties between equal values.*

Algorithm: *GR* (Greedy) Enqueue a new packet if

- the queue is not full
- or a packet with the smallest value in the queue has a lower value then the new packet. In this case, a smallest value packet is discarded and the new packet in enqueued.

Algorithm: *TLH* (Transmit Largest Head)

1. Buffer management: Use algorithm *GR* independently in all m incoming queues.
2. Scheduling: At each time step, transmit the packet with the largest value among all packets at the head of the queues.

Theorem 11 ([5]) *Algorithm TLH is 3-competitive.*

Algorithm: *TL* (Transmit Largest)

1. Buffer management: Use algorithm *GR* independently in all m incoming queues.
2. Scheduling: At each time step, transmit the packet with the largest value among all packets stored in the queues.

Algorithm: GS^A (Generic Switch)

1. Buffer management: Apply buffer management policy A to all m incoming queues.
2. Scheduling: Run a simulation of algorithm *TL* (in the preemptive relaxed model) with the online input sequence σ. Adopt all scheduling decisions of *TL*, i.e., at each time step, transmit the packet at the head of the queue used by *TL* simulation.

Theorem 12 (General reduction [4]) *Let* GS^A *denote the algorithm obtained by running algorithm GS with the event-driven single-queue buffer management policy A (preemptive or non-preemptive) and let* c_A *be the competitive ratio of A. The competitive ratio of* GS^A *satisfies* $c_{GS^A} \leq 2 \cdot c_A$.

Applications

The unit value scenario models most current networks, e.g., IP networks which only support a "best effort" service in which all packet streams are treated equally, whereas the scenario with arbitrary packet values integrates full QoS capabilities.

The general reduction technique allows to restrict oneself to investigate single-queue buffer problems. It can be applied to a 1.75-competitive algorithm named *PG* by Bansal et al. [7], which achieves the best ratio known today, and yields an algorithm GS^{PG} that is 3.5-competitive for multi-queue buffers (3.5 is still higher than 3 which is the competitive ratio of TLH). In the 2-value preemptive model, Lotker and Patt-Shamir [8] presented a *mark&flush* algorithm *mf* that is 1.30-competitive for single queue buffers and that the general reduction technique transforms into a 2.60-competitive algorithm GS^{mf} for multi-queue buffers.

For the general non-preemptive model, Andelman et al. [2] presented a policy for a single queue called *Exponential-Interval Round-Robin* (*EIRR*), which is $(e\lceil \ln \alpha \rceil)$-competitive, and showed also a lower bound of $\Theta(\log \alpha)$. In the multi-queue buffer case, the general reduction technique provides a non-preemptive $(e\lceil \ln \alpha \rceil)$-competitive algorithm.

Open Problems

It is known from Theorem 3 that the competitive ratio of any greedy algorithm in the unit value model is at least 2 if $m \gg B$. Which is the tight upper bound for greedy algorithms in the opposite case $B \gg m$?

The proof of the lower bound $e/(e-1)$ in Theorem 1 uses $m \gg B$ whereas Theorem 5

achieves $e/(e-1)$ as an upper bound for $B \gg \log m$. In [4], a lower bound of 1.366 is shown, independent of B and m. Which is the optimal competitive ratio for arbitrary B and m?

Due to the general reduction technique in Theorem 7, the competitive ratio for multi-queue buffer algorithms can be improved if better competitiveness results for single queue buffer algorithms are achieved. Currently, $\frac{\sqrt{13}+5}{6} \approx 1.43$ [2] and 1.75 [7] are the best known lower and upper bounds, respectively. How to reduce this gap?

Cross-References

▸ Online Paging and Caching
▸ Packet Switching in Single Buffer

Recommended Reading

1. Albers S, Schmidt M (2005) On the performance of greedy algorithms in packet buffering. SIAM J Comput 35:278–304
2. Andelman N, Mansour Y, Zhu A (2003) Competitive queueing policies for QoS switches. In: Proceedings of the 14th ACM-SIAM symposium on discrete algorithms (SODA), pp 761–770
3. Azar Y, Litichevskey M (2004) Maximizing throughput in multiqueue switches. In: Proceedings of the 12th annual European symposium on algorithms (ESA), pp 53–64
4. Azar Y, Richter Y (2003) Management of multi-queue switches in QoS networks. In: Proceedings of the 35th ACM symposium on theory of computing (STOC), pp 82–89
5. Azar Y, Richter Y (2004) The zero-one principle for switching networks. In: Proceedings of the 36th ACM symposium on theory of computing (STOC), pp 64–71
6. Azar Y, Richter Y (2004) An improved algorithm for CIOQ switches. In: Proceedings of the 12th annual European symposium on algorithms (ESA). LNCS, vol 3221, pp 65–76
7. Bansal N, Fleischer L, Kimbrel T, Mahdian M, Schieber B, Sviridenko M (2004) Further improvements in competitive guarantees for QoS buffering. In: Proceedings of the 31st international colloquium on automata, languages, and programming (ICALP), pp 64–71
8. Lotker Z, Patt-Shamir B (2003) Nearly optimal FIFO buffer management for two packet classes. Comput Netw 42(4):481–492
9. Schmidt M (2005) Packet buffering: randomization beats deterministic algorithms. In: Proceedings of the 22nd annual symposium on theoretical aspects of computer science (STACS). LNCS, vol 3404, pp 293–304

Packet Switching in Single Buffer

Rob van Stee
University of Leicester, Leicester, UK

Keywords

Buffering

Years and Authors of Summarized Original Work

2003; Bansal, Fleischer, Kimbrel, Mahdian, Schieber, Sviridenko

Problem Definition

In this entry, consider a quality-of-service (QoS) buffering system that is able to hold B packets. Time is slotted. At the beginning of a time step, a set of packets (possibly empty) arrives, and at the end of the time step, a single packet may leave the buffer to be transmitted. Since the buffer has a bounded size, at some point packets may need to be dropped. The buffer management algorithm has to decide at each step which of the packets to drop and which packets to transmit, subject to the buffer capacity constraint. The value of a packet p is denoted by $v(p)$. The system obtains the value of the packets it sends and gains no value otherwise. The aim of the buffer management algorithm is to maximize the total value of transmitted packets.

In the FIFO model, the packet transmitted at time t is always the first (oldest) packet in the buffer.

In the *nonpreemptive* model, packets accepted to the queue will be transmitted eventually and cannot be dropped. In this model, the best com-

petitive ratio achievable is $\Theta(\log \alpha)$, where α is the ratio of the maximum value of a packet to the minimum [1,2].

In the *preemptive* model, packets can also be dropped at some later time before they are served. The rest of this entry focuses on this model. Mansour, Patt-Shamir, and Lapid [11] were the first to study preemptive queuing policies for a single FIFO buffer, proving that the natural greedy algorithm (see definition in Fig. 1) maintains a competitive ratio of at most 4. This bound was improved to the tight value of 2 by Kesselman, Lotker, Mansour, Patt-Shamir, Schieber, and Sviridenko [8]. An alternative proof of the 2-competitiveness, due to Kimbrel [9], is presented in Epstein and van Stee's survey on buffer management [5].

The greedy algorithm is not optimal since it never preempts a packet until the buffer is full and this might be too late. The first algorithm with a competitive ratio strictly below 2 was presented by Kesselman, Mansour, and van Stee [7]. This algorithm uses a parameter β and introduces an extra rule for processing arrivals that is executed before rules (1) and (2) of the greedy algorithm. This rule is formulated in Fig. 2.

It is shown in [7] that by taking $\beta = 15$, the algorithm preemptive greedy (PG) has a competitive ratio of 1.983. The analysis is rather complicated and is done by assigning the value of packets served by the offline algorithm to packets served by PG.

A lower bound of 5/4 for this problem was shown in [11]. This was improved to $\sqrt{2}$ in [2] and then to 1.419 in [7].

Key Results

A modification of PG was presented by Bansal, Fleischer, Kimbrel, Mahdian, Schieber, and Sviridenko [3]. It changes rule 0 to rule 0′ (Fig. 3).

Thus, the modification compared to PG is that this algorithm finds a "locally optimal" packet to evict. We will denote modified preemptive greedy by MPG.

Packet Switching in Single Buffer, Fig. 1 The natural greedy algorithm

The Greedy Algorithm.
When a packet of value $v(p)$ arrives:
1. Accept p if there is free space in the buffer.
2. Otherwise, reject (drop or preempt) the packet p' that has minimal value among p and the packets in the buffer. If $p' \neq p$, accept p.

0. Preempt (drop) the first packet p' in the FIFO order such that $v(p') \leq v(p)/\beta$, if any (p *preempts* p').

Packet Switching in Single Buffer, Fig. 2 Extra rule for the preemptive greedy algorithm

Packet Switching in Single Buffer, Fig. 3 Modified preemptive greedy

0'. Find the first (i.e., closest to the front of the buffer) packet p' such that p' has value less than $v(p)/\beta$ and not more than the value of the packet after p' in the buffer (if any). If such a packet exists, drop it (p *preempts* p').

Theorem 1 ([3]) *For $\beta = 4$, MPG has a competitive ratio of 1.75.*

The proof begins by showing that in order to analyze the performance of MPG, it is sufficient to consider only input instances in which the value of each packet is either 0 or β^i for some $i \geq 0$, but ties are allowed to be broken by the adversary.

The authors then define an *interval structure* for input instances. An interval I is said to be of type i if at every step $t \in I$, MPG outputs a packet of value at least β^i , and I is a maximal interval with this property.

$\mathcal{I}_i$ is the collection of maximal intervals of type i, and $\mathcal{I}$ is the union of all $\mathcal{I}_i$'s. This is a multiset, since an interval of type i can also be contained in an interval of one or more types $j < i$.

This induces an interval structure which is a sequence of ordered rooted trees in a natural way: The root of each tree is an interval in $\mathcal{I}_0$, and the children of each interval $I \in \mathcal{I}_i$ are all the maximal intervals of type $i + 1$ which are contained in I. These children are ordered from left to right based on time, as are the trees themselves. The intervals of type i (and the vertices that represent them) are distinguished by whether or not an eviction of a packet of value at least β^i occurred during the interval.

To complete the proof, the authors show that for every interval structure $\mathcal{T}$, the competitive ratio of MPG on instances with interval structure $\mathcal{T}$ can be bounded by the solution of a linear program indexed by $\mathcal{T}$. Finally, it is shown that for every $\mathcal{T}$ and every $\beta \geq 4$, the solution of this program is at most $2 - 1/\beta$.

Applications

In recent years, there has been a lot of interest in quality-of-service networks. In regular IP networks, packets are indistinguishable, and in case of overload, any packet may be dropped. In a commercial environment, it is much more preferable to allow better service to higher-paying customers or customers with critical requirements. The idea of quality-of-service guarantees is that packets are marked with values which indicate their importance.

This naturally leads to decision problems at network switches when many packets arrive and overload occurs. The algorithm presented in this entry can be used to maximize network performance in a network which supports quality of service.

Open Problems

Despite substantial advances in improving the upper bound for this problem, a fairly large gap remains. Sgall (quoted in Jawor [6]) showed that the performance of PG is as good as that of MPG. Englert and Westermann [4] showed that PG has a competitive ratio of at most $\sqrt{3} \approx 1.732$ and at least $1 + 1/2\sqrt{2} \approx 1.707$. Thus, to improve further, a different algorithm will be needed.

The authors also note that Lotker and Patt-Shamir [10] studied the special case of two packet values and derived a 1.3-competitive algorithm, which closely matches the corresponding lower bound of 1.28 from Mansour et al. [11]. An open problem is to close the remaining small gap.

Cross-References

▸ Single and Multiple Buffer Processing

Recommended Reading

1. Aiello W, Mansour Y, Rajagopolan S, Rosen A (2000) Competitive queue policies for differentiated services. In: Proceedings of the IEEE INFOCOM, Tel-Aviv. IEEE, pp 431–440
2. Andelman N, Mansour Y, Zhu A (2003) Competitive queueing policies in QoS switches. In: Proceedings of the 14th symposium on discrete algorithms (SODA), Baltimore. ACM/SIAM, San Francisco, pp 761–770
3. Bansal N, Fleischer L, Kimbrel T, Mahdian M, Schieber B, Sviridenko M (2004) Further improvements in competitive guarantees for QoS buffering. In: Proceedings of the 31st international colloquium on automata, languages, and programming (ICALP),

Turku. Lecture notes in computer science, vol 3142. Springer, Berlin, pp 196–207
4. Englert M, Westermann M (2006) Lower and upper bounds on FIFO buffer management in qos switches. In: Azar Y, Erlebach T (eds) Proceedings of the 14th annual European symposium on algorithms – ESA 2006, Zurich. Lecture notes in computer science, vol 4168. Springer, Berlin, pp 352–363
5. Epstein L, van Stee R (2004) Buffer management problems. SIGACT News 35(3):58–66
6. Jawor W (2005) Three dozen papers on online algorithms. SIGACT News 36(1):71–85
7. Kesselman A, Mansour Y, van Stee R (2003) Improved competitive guarantees for QoS buffering. In: Di Battista G, Zwick U (eds) Proceedings of the eleventh annual European symposium on algorithms – ESA 2003, Budapest. Lecture notes in computer science, vol 2380. Springer, Berlin, pp 361–373
8. Kesselman A, Lotker Z, Mansour Y, Patt-Shamir B, Schieber B, Sviridenko M (2004) Buffer overflow management in QoS switches. SIAM J Comput 33(3):563–583
9. Kimbrel T (2004) A simple proof of the 2-competitiveness of the greedy FIFO buffering algorithm. Technical report RC23272, IBM Research
10. Lotker Z, Patt-Shamir B (2002) Nearly optimal FIFO buffer management for DiffServ. In: Proceedings of the 21st ACM symposium on principles of distributed computing (PODC 2002), Monterey. ACM, New York, pp 134–142
11. Mansour Y, Patt-Shamir B, Lapid O (2000) Optimal smoothing schedules for real-time streams. In: Proceedings of the 19th symposium on principles of distributed computing (PODC), Portland. ACM, New York, pp 21–29

PageRank Algorithm

Monika Henzinger
University of Vienna, Vienna, Austria

Keywords

Hyperlink analysis; Web information retrieval

Years and Authors of Summarized Original Work

1998; Brin, Page

Problem Definition

Given a user query, current web search services retrieve all web pages that contain the query terms resulting in a huge number of web pages for the majority of searches. Thus it is crucial to reorder or *rank* the resulting documents with the goal of placing the most relevant documents first. Frequently, ranking uses two types of information: (1) query-specific information and (2) query-independent information. The query-specific part tries to measure how relevant the document is to the query. Since it depends to a large part on the content of the page, it is mostly under the control of the page's author. The query-independent information tries to estimate the quality of the page in general. To achieve an objective measure of page quality, it is important that the query-independent information incorporates a measure that is not controlled by the author. Thus the problem is to find a measure of page quality that: (a) cannot be easily manipulated by the web page's author and (b) works well for *all* web pages. This is challenging as web pages are extremely heterogeneous.

Key Results

The hyperlink structure of the web is a good source for basing such a measure as it is hard for one author or a small set of authors to influence the whole structure, even though they can manipulate a subset of the web pages. Brin and Page showed that a relatively simple analysis of the hyperlink structure of the web can be used to produce a quality measure for web documents that leads to large improvements in search quality. The measure is called the *PageRank* measure.

Linear Algebra-Based Definition

Let n be the total number of web pages. The PageRank vector is an $n$dimensional vector with one dimension for each web page. Let d be a small constant, like 1/8, let $deg(p)$ denote the number of hyperlinks in the body text of page p,

and let $PR(p)$ denote the PageRank value of page p. Assume first that every page contains at least one hyperlink. In such a collection of web pages, the PageRank vector is computed by solving a system of linear equations that contains for each page p the equation

$$PR(p) = d/n + (1-d) * \sum_{q \text{ has hyperlink to } p} PR(q)/deg(q)$$

In matrix notation, the PageRank vector is the Eigenvector with 1Norm one of the matrix A with $d/n + (1-d)/deg(q)$ for entry A_{qp} if q has a hyperlink to p and d/n otherwise.

If web pages without hyperlinks are allowed in this linear system, then they might become "PageRank sinks", i.e., they would "receive" PageRank from the pages pointing to them, but would not "give out" their PageRank, potentially resulting in an "unusually high" PageRank value for themselves. Brin and Page proposed two ways to deal with web pages without out-links, namely either to recursively delete them until no such web pages exist anymore in the collection or to add a hyperlink from each such page to *every* other page.

Random Surfer Model

Let the *web graph* $G = (V, E)$ be a directed graph such that each node corresponds to a web page and every hyperlink corresponds to a directed edge from the referencing node to the referenced node. The PageRank can also be interpreted as the following random walk in the web graph. The random walk starts at a random node in the graph. Assume in step k it visits page q. Then it flips a biased coin, and with probability d or if q has no out-edges, it selects a random node out of V and visits it in step $k+1$. Otherwise it selects a random out-edge of the current node and visits it in step $k+1$. (Note that this corresponds to adding a directed edge from every page without hyperlink to *every* node in the graph.) Under certain conditions (which do not necessarily hold on the web) the stationary distribution of this random walk corresponds to the PageRank vector. See [1, 4] for details.

Brin and Page also suggested computing the PageRank vector approximately using the power method, i.e., by setting all initial values to $1/n$ and then repeatedly using the PageRank vector of the previous iteration to compute the PageRank vector of the current iteration using the above linear equations. After a hundred iterations, barely any values change and the computation is stopped.

Applications

The PageRank measure is used as one of the factors by Google in its ranking of search results. The PageRank computation can be applied to other domains as well. Two examples are reputation management in P2P networks and learning word dependencies in natural language processing. In relational databases PageRank was used to weigh database tuples in order to improve keyword searching when a user does not know the schema. Finally, in rank aggregation PageRank can be used to find a permutation that minimally violates a set of given orderings. See [1] for more details.

Variations of PageRank were studied as well. Personalizing the PageRank computation such that the values reflect the interest of a user has received a lot of attention. See [3] for a survey on this topic. It can also be modified to be used for detecting web search spam, i.e., web pages that try to manipulate web search results [1].

Recommended Reading

1. Berkhin P (2005) A survey on PageRank computing. Internet Math 2(1):73–120

2. Brin S, Page L (1998) The anatomy of a large-scale hypertextual Web search engine. In: Proceedings of the 7th international conference on World Wide Web. Elsevier, Amsterdam, Brisbane, Australia, pp 107–117
3. Haveliwala T, Kamvar S, Jeh G (2003) An analytical comparison of approaches to personalizing PageRank. Technical report, Stanford University, Stanford
4. Langville AN, Meyer CD (2004) Deeper inside PageRank. Internet Math 1(3):335–380
5. Page L, Brin S, Motwani R, Winograd T (1998) The PageRank citation ranking: bringing order to the Web. Technical report, Stanford University, Stanford

Parallel Algorithms for Two Processors Precedence Constraint Scheduling

Maria Serna
Department of Language and System Information, Technical University of Catalonia, Barcelona, Spain

Keywords

Optimal scheduling for two processors

Years and Authors of Summarized Original Work

2003; Jung, Serna, Spirakis

Problem Definition

In the general form of *multiprocessor precedence scheduling problems* a set of n tasks to be executed on m processors is given. Each task requires exactly one unit of execution time and can run on any processor. A directed acyclic graph specifies the precedence constraints where an edge from task x to task y means task x must be completed before task y begins. A solution to the problem is a schedule of shortest length indicating when each task is started. The work of Jung, Serna, and Spirakis provides a parallel algorithm (on a PRAM machine) that solves the above problem for the particular case that $m = 2$, that is where there are two parallel processors.

The *two processor precedence constraint scheduling problem* is defined by a directed acyclic graph (dag) $G = (V, E)$. The vertices of the graph represent unit time tasks, and the edges specify precedence constraints among the tasks. If there is an edge from node x to node y then x is an *immediate predecessor* of y. *Predecessor* is the transitive closure of the relation immediate predecessor, and *successor* is its symmetric counterpart. A *two processor schedule* is an assignment of the tasks to time units $1, \ldots, t$ so that each task is assigned exactly one time unit, at most two tasks are assigned to the same time unit, and if x is a predecessor of y then x is assigned to a lower time unit than y. The length of the schedule is t. A schedule having minimum length is an *optimal* schedule. Thus the problem is the following:

Name	Two processor precedence constraint scheduling
Input	A directed acyclic graph
Output	A minimum length schedule preserving the precedence constraints.

Preliminaries

The algorithm assume that tasks are partitioned into levels as follows:

(i) Every task will be assigned to only one level
(ii) Tasks having no successors will be assigned to level 1 and
(iii) For each level i, all tasks which are immediate predecessors of tasks in level i will be assigned to level $i + 1$.

Clearly topological sort will accomplish the above partition, and this can be done by an NC algorithm that uses $O(n^3)$ processors and $O(\log n)$ time, see [3]. Thus, from now on, it is assumed that a level partition is given as part of the input. For sake of convenience two special tasks, t_0 and t^* are added, in such a way that

the original graph could be taught as the graph formed by all tasks that are successors of t_0 and predecessors of t^*. Thus t_0 is a predecessor of all tasks in the system (actually an immediate predecessor of tasks in level the highest level $L(G)$) and t^* is a successor of all tasks in the system (an immediate successor of level 1 tasks).

Notice that if two tasks are at the same level they can be paired. But when x and y are at different levels, they can be paired only when neither of them is a predecessor of the other. Let $L(G)$ denote the number of levels in a given precedence graph G. A *level schedule* schedules tasks level by level. More precisely, suppose levels $L(G), \ldots, i+1$ have already been scheduled and there are k unscheduled tasks remaining on level i. When k is even, those tasks with are paired with each other. When k is odd, $k-1$ of the tasks are paired with each other, while the remaining task may (but not necessarily) be paired with a task from a lower level.

Given a level schedule level *i jumps to level* i' ($i' < i$) if the last time step containing a task from level i also contains a task from level i'. If the last task from level i is scheduled with an empty slot, it is said that level *i jumps to level* 0. The *jump sequence* of a level schedule is the list of levels jumped to. A *lexicographically first jump schedule* is a level schedule whose jump sequence is lexicographically greater than any other jump sequence resulting from a level schedule.

Given a graph G a *level partition* of G is a partition of the nodes in G into two sets in such a way that levels $0, \ldots, k$ are contained in one set (the upper part) denoted by U, and levels $k+1, \ldots, L$ in the other (the lower part) denoted by L. Given a graph G and a level i, the *i-partition* of G (or the partition at level i) is formed by the graphs U_i and L_i defined as U_i contains all nodes x such that level$(x) < i$ and L_i contains all nodes x with level$(x) > i$. Note that each i-partition determines two different level partitions depending on whether level i nodes are assigned to the upper or the lower part. A task $x \in U_i$ is called *free* with respect to a partition at level i if x has no predecessors in L_i.

Auxiliary Problems

The algorithm for the two processors precedence constraint scheduling problem uses as a building block an algorithm for solving a matching problem in a particular graph class.

A *full convex bipartite graph* G is a triple (V, W, E), where $V = \{v_1, \ldots, v_k\}$ and $W = \{w_1, \ldots, w_{k'}\}$ are disjoint sets of vertices. Furthermore the edge set E satisfies the following property: If $(v_i, w_j) \in E$ then $(v_q, w_j) \in E$ for all $q \geq i$. Thus, from now on it is assumed that the graph is connected.

A set $F \subseteq E$ is a *matching* in the graph $G = (V, W, E)$ iff no two edges in F have a common endpoint. A *maximal matching* is a matching that cannot be extended by the addition of any edge in G. A *lexicographically first maximal matching* is a maximal matching whose sorted list of edges is lexicographically first among all maximal matchings in G.

Key Results

When the number of processors m is arbitrary the problem is known to be NP-complete [8]. For any $m \geq 3$, the complexity is open [6]. Here the case of interest has been $m = 2$. For two processors a number of efficient algorithms has been given. For sequential algorithms see [2, 4, 5] among others. The first deterministic parallel algorithm was given by Helmbold and Mayr [7], thus establishing membership in the class NC. Previously [9] gave a randomized NC algorithm for the problem. Jung, Serna and Spirakis present a new parallel algorithm for the two processors scheduling problem that takes time $O(\log^2 n)$ and uses $O(n^3)$ processors on a CREW PRAM. The algorithm improves the number of processors of the algorithm given in [7] from $O(n^7 L(G)^2)$, where $L(G)$ is the number of levels in the precedence graph, to $O(n^3)$. Both algorithms compute a level schedule that has a lexicographically first jump sequence.

To match jumps with tasks it is used a solution to the problem of computing the lexicographically first matching for a special type of convex

bipartite graphs, here called *full convex bipartite graphs*. A geometric interpretation of this problem leads to the discovery of an efficient parallel algorithm to solve it.

Theorem 1 *The lexicographically first maximal matching of full convex bipartite graphs can be computed in time* $O(\log n)$ *on a CREW PRAM with* $O(n^3/\log n)$ *processors, where* n *is the number of nodes.*

The previous algorithm is used to solve efficiently in parallel two related problems.

Theorem 2 *Given a precedence graph G, there is a PRAM parallel algorithm that computes all levels that jump to level 0 in the graph* L_i *and all tasks in level* $i-1$ *that can be scheduled together with a task in level i, for* $i = 1, \ldots, L(G)$*, using* $O(n^3)$ *processors and* $O(\log^2 n)$ *time.*

Theorem 3 *Given a level partition of a graph G together with the levels in the lower part in which one task remains to be matched with some other task in the upper part of the graph. There is a PRAM parallel algorithm that computes the corresponding tasks in time* $O(\log n)$ *using* $n^3/\log n$ *processors.*

With those building blocks the algorithm for two processor precedence constraint scheduling starts by doing some preprocessing and after that an adequate decomposition that insure that at each recursive call a number of problems of half size are solved in parallel. This recursive schema is the following:

Algorithm Schedule

0. Preprocessing
1. Find a level i such that $|U_i| \leq n/2$ and $|L_i| \leq n/2$.
2. Match levels that jump to free tasks in level i.
3. Match levels that jump to free tasks in U_i.
4. If level i (or $i+1$) remain unmatched try to match it with a non free task.
5. Delete all tasks used to match jumps.
6. Apply (1)–(5) in parallel to L_i and the modified U_i.

Algorithm **Schedule** stops whenever the corresponding graph has only one level.

The correction an complexity bounds for algorithm **Schedule** follows from the previous results, leading to:

Theorem 4 *There is an NC algorithm which finds an optimal two processors schedule for any precedence graph in time* $O(\log^2 n)$ *using* $O(n^3)$ *processors.*

Applications

A fundamental problem in many applications is to devise a proper schedule to satisfy a set of constrains. Assigning people to jobs, meetings to rooms, or courses to final exam periods are all different examples of scheduling problems. A key and critical algorithm in parallel processing is the one mapping tasks to processors. In the performance of such an algorithm relies many properties of the system, like load balancing, total execution time, etc. Scheduling problems differ widely in the nature of the constraints that must be satisfied, the type of processors, and the type of schedule desired.

The focus on precedence-constrained scheduling problems for directed acyclic graphs has a most direct practical application in problems arising in parallel processing. In particular in systems where computations are decomposed, prior to scheduling into approximately equal sized tasks and the corresponding partial ordering among them is computed. These constraints must define a directed acyclic graph, acyclic because a cycle in the precedence constraints represents a Catch situation that can never be resolved.

Open Problems

The parallel deterministic algorithm for the two processors scheduling problem presented here improves the number of processors of the Helmbold and Mayr algorithm for the problem [7]. However, the complexity bounds are far from

optimal: recall that the sequential algorithm given in [5] uses time $O(e + n\alpha(n))$, where e is the number of edges in the precedence graph and $\alpha(n)$ is an inverse Ackermann's function. Such an optimal algorithm might have a quite different approach, in which the levelling algorithm is not used.

Interestingly enough computing the lexicographically first matching for full convex bipartite graphs is in NC, in contraposition with the results given in [1] which show that many problems defined through a lexicographically first procedure in the plane are P-complete. It is an interesting problem to show whether all these problems fall in NC when they are convex.

Cross-References

- Approximation Schemes for Makespan Minimization
- List Scheduling
- Maximum Matching
- Shortest Elapsed Time First Scheduling
- Stochastic Scheduling
- Voltage Scheduling

Recommended Reading

1. Attallah M, Callahan P, Goodrich M (1993) P-complete geometric problems. Int J Comput Geom Appl 3(4):443–462
2. Coffman EG, Graham RL (1972) Optimal scheduling for two processors systems. Acta Informatica 1:200–213
3. Dekel E, Nassimi D, Sahni S (1981) Parallel matrix and graph algorithms. SIAM J Comput 10:657–675
4. Fujii M, Kasami T, Ninomiya K (1969) Optimal sequencing of two equivalent processors. SIAM J Comput 17:784–789
5. Gabow HN (1982) An almost linear time algorithm for two processors scheduling. J ACM 29(3):766–780
6. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP completeness. Freeman, San Francisco
7. Helmbold D, Mayr E (1987) Two processor scheduling is in NC. SIAM J Comput 16(4):747–756
8. Ullman JD (1975) NP-complete scheduling problems. J Comput Syst Sci 10:384–393
9. Vazirani U, Vazirani V (1989) Two-processor scheduling problem is in random NC. SIAM J Comput 18(4):1140–1148

Parallel Connectivity and Minimum Spanning Trees

Tak-Wah Lam
Department of Computer Science, University of Hong Kong, Hong Kong, China

Keywords

EREW PRAM algorithms for finding connected components and minimum spanning trees

Years and Authors of Summarized Original Work

1995; Ka Wong Chong, Lam
2001; Ka Wong Chong, Han, Lam

Problem Definition

Given a weighted undirected graph G with n vertices and m edges, compute a minimum spanning tree (or spanning forest) of G on a parallel random access machine (PRAM) without concurrent write capability.

A minimum spanning tree of a graph is a spanning tree with the smallest possible sum of edge weights. The parallel random access machine (PRAM) is an abstract model for designing parallel algorithms and understanding the power of parallelism. In this model, processors (each being a random access machine) work in a synchronous manner and communicate through a shared memory. PARM can be further classified according to whether it is allowed for more than one processor to read and write into the same shared memory location simultaneously. The strongest model is CRCW (concurrent-read, concurrent-write) PRAM, and the weakest is EREW (exclusive-read, exclusive-write) PRAM. For an introduction to PRAM algorithms, one can refer to Karp and Ramachandran [8] and JáJá [5].

The input graph G is assumed to be given in the form of adjacency lists. Furthermore, isolated (degree-0) vertices are removed, and hence it is assumed that $m \geq n$.

Key Results

The MST problem is related to the connected component (CC) problem, which is to find the connected components of an undirected graph. Sequential algorithms for solving the CC problem and the MST problem in $O(m)$ time and $O(m \log n)$ time, respectively, were known a few decades ago. A number of more efficient MST algorithms have since been published, the most recent of which is Pettie and Ramachandran's algorithm [9], which is provably optimal.

In the parallel context, both problems are often solved in a similar way. With respect to CRCW PRAM, the two problems can be solved using $O(\log n)$ time and $n + m$ processors (see, e.g., Cole and Vishkin [3]). Using randomization, $(n + m)/\log n$ processors are sufficient to solve these problems in $O(\log n)$ expected time [2, 10].

For EREW PRAM, $O(\log^2 n)$ time algorithms for the CC and MST problems were developed in the early 1980s. For a while, it was believed that the exclusive write models (including both concurrent read and exclusive read) could not overcome the $O(\log^2 n)$ time bound [8]. The first breakthrough was due to Johnson and Metaxas [6]; they devised $O(\log^{1.5} n)$ time algorithms for the CC problem and the MST problem. These results were soon improved to $O(\log n \log\log n)$ time by Chong and Lam. If randomization is allowed, the time complexity can be improved to $O(\log n)$ expected time and optimal work [7, 10, 11]. Finally, Chong, Han, and Lam [1] obtained an algorithm for MST (and CC) using $O(\log n)$ time and $n + m$ processors. This algorithm does not need randomization. Notice that $\Theta(\log n)$ is optimal since these graphs' problems are at least as hard as computing the OR of n bits, and Cook et al. [4] have proven that the latter requires $\Omega(\log n)$ time on exclusive-write PRAM no matter how many processors are used.

Below is a sketch of some ideas for computing a minimum spanning tree in parallel without using concurrent write.

Without loss of generality, assume that the edge weights are all distinct. Thus, G has a unique minimum spanning tree, which is denoted by T_G^*. Let B be a subset of edges in G which contains no cycle. B induces a set of trees $F = \{T_1, T_2, \cdots, T_l\}$ in a natural sense – two vertices in G are in the same tree if they are connected by an edge of B. B is said to be a λ-forest if each tree $T \in F$ has at least λ vertices. For example, if B is the empty set, then B is a 1forest; a spanning tree is an n-forest.

Suppose that B is a λ-forest and its edges are all found in T_G^*. Then B can be augmented to give a 2λ-forest using a greedy approach: Let F' be an arbitrary subset of F including all trees $T \in F$ with fewer than 2λ vertices. For every tree in F', pick its minimum external edge (i.e., the smallest-weight edge connecting to a vertex outside the tree). Denote B' as this set of edges. It can be proven that B' consists of edges in T_G^* only and $B \cup B'$ is a 2λ-forest. The above idea allows us to find T_G^* in $\lfloor \log n \rfloor$ stages as follows:

1. $B \leftarrow \phi$
2. **For** $i = 1$ to $\lfloor \log n \rfloor$ **do** /* Stage i */
   1. Let F be the set of trees induced by B on G. Let F' be an arbitrary subset of F such that F' includes all trees $T \in F$ with fewer than 2^i vertices.
   2. $B_i \leftarrow \{e \mid e$ is the minimum external edge of $T \in F'\}$; $B \leftarrow B \cup B_i$
3. **return** B

Different strategies for choosing the set F' in Step 1(a) may lead to different B_i's. Nevertheless, $B[1, i]$ is always a subset of T_G^* and induces a 2^i -forest. In particular, $B[1, \lfloor \log n \rfloor]$ induces exactly one tree, which is exactly T_G^*. Using standard parallel algorithmic techniques, each stage can be implemented in $O(\log n)$ time on EREW PRAM using a linear number of processors (see, e.g., [5],). Therefore, T_G^* can be found in $O(\log^2 n)$ time. In fact, most parallel algorithms for finding MST are based on a similar approach. These parallel algorithms are "sequential" in the sense that the computation of B_i starts only after B_{i-1} is available.

The $O(\log n)$-time EREW algorithm in [1] is based on some structural properties related to MST and can compute the B_i's in a more parallel fashion. In this algorithm, there are $\lfloor \log n \rfloor$ concurrent threads (a thread is simply a group

of processors). For $1 \leq i \leq \lfloor \log n \rfloor$, Thread i aims at computing B_i, and it actually starts long before Thread $i-1$ has computed B_{i-1}, and it receives the output of Threads 1 to $i-1$ (i.e., $B_1, \cdots, B_{i-1}$) incrementally. More specifically, the algorithm runs in $\lfloor \log n \rfloor$ supersteps, where each superstep lasts $O(1)$ time. Thread i delivers B_i at the end of the ith superstep. The computation of Thread i is divided into $\lfloor \log i \rfloor$ phases. Let us first consider a simple case when i is a power of 2. Phase 1 of Thread i starts at the $(i/2+1)$th superstep, i.e., when $B_1, \cdots, B_{i/2}$ are available. Phase 1 takes no more than $i/4$ supersteps, ending at the $(i/2+i/4)$th superstep. Phase 2 starts at the $(i/2+i/4+1)$th superstep (i.e., when $B_{i/2+1}, \cdots, B_{i/2+i/4}$ are available) and uses $i/8$ supersteps. Each subsequent phase uses half as many supersteps as the preceding phase. The last phase (Phase $\log i$) starts and ends within the ith superstep; note that B_{i-1} is available after $(i-1)$th superstep.

Applications

Finding connected components or MST is a key step in several parallel algorithms for other graph problems. For example, the Chong-Han-Lam algorithm implies an $O(\log n)$ time algorithm for finding ear decomposition and biconnectivity without using concurrent write.

Cross-References

- ▸ Graph Connectivity
- ▸ Randomized Parallel Approximations to Max Flow

Recommended Reading

1. Chong KW, Han Y, Lam TW (2001) Concurrent threads and optical parallel minimum spanning trees algorithm. J ACM 48(2):297–323
2. Cole R, Klein PN, Tarjan RE (1996) Finding minimum spanning forests in logarithmic time and linear work using random sampling. In: Proceedings of the 8th annual ACM symposium on parallel architectures and algorithms, Padua, pp 243–250
3. Cole R, Vishkin U (1986) Approximate and exact parallel scheduling with applications to list, tree, and graph problems. In: Proceedings of the 27th annual IEEE symposium on foundations of computer science, Toronto, pp 478–491
4. Cook SA, Dwork C, Reischuk R (1986) Upper and lower time bounds for parallel random access machines without simultaneous writes. SIAM J Comput 15(1):87–97
5. JáJá J (1992) An introduction to parallel algorithms. Addison-Wesley, Boston
6. Johnson DB, Metaxas P (1991) Connected components in $O(\lg^{3/2} |V|)$ parallel time for the CREW PRAM. In: Proceedings of the 32nd annual IEEE symposium on foundations of computer science, San Juan, pp 688–697
7. Karger DR (1995) Random sampling in graph optimization problems. Ph.D. thesis, Department of computer science, Stanford University
8. Karp RM, Ramachandran V (1990) Parallel algorithms for shared-memory machines. In: Van Leeuwen Ed J (ed) Handbook of theoretical computer science, vol A, pp 869–941. MIT Press, Massachusetts
9. Pettie S, Ramachandran V (2002) An optimal minimum spanning tree algorithm. J ACM 49(1):16–34
10. Pettie S, Ramachandran V (2002) A randomized time-work optimal parallel algorithm for finding a minimum spanning forest. SIAM J Comput 31(6):1879–1895
11. Poon CK, Ramachandran V (2003) A randomized linear-work EREW PRAM algorithm to find a minimum spanning forest. Algorithmica 35(3):257–268

Parameterization in Computational Social Choice

Piotr Faliszewski[1] and Rolf Niedermeier[2,3]
[1]AGH University of Science and Technology, Krakow, Poland
[2]Department of Mathematics and Computer Science, University of Jena, Jena, Germany
[3]Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Berlin, Germany

Keywords

Bribery in elections; Control in elections; NP-hard problems; Parameterized and multivariate complexity analysis; Voting problems; Winner determination

Years and Authors of Summarized Original Work

1989; Bartholdi, Tovey, Trick
2009; Betzler, Uhlmann
2011; Liu, Feng, Zhu, Luan
2012; Dorn, Schlotter

Problem Definition

Computational Social Choice is an interdisciplinary research area involving Economics, Political Science, and Social Science on the one side and Mathematics and Computer Science (including artificial intelligence and multi-agent systems) on the other side. Concrete questions addressed in this area include the following three: How efficiently can one determine the winner of an election, given a number of votes with preferences over a number of alternatives? Is it possible to obtain a desirable outcome of an election by executing a number of campaigning actions? (Formally, such problems are often modeled as bribery.) Can the chair of an election influence the result of an election by modifying the set of available alternatives (e.g., by encouraging some alternatives (candidates) to participate in the run)?

The main objective of parameterized complexity is to analyze computationally hard problems with respect to multiple input parameters and to classify these problems according to their inherent difficulty when "viewed through different parameterizations." The complexity of a problem typically depends on the values of a multitude of input parameters, so this approach allows to classify NP-hard problems on a much finer scale than in classical complexity theory (where the complexity of a problem is only measured relative to the input size). In particular, a parameterized problem consisting of an input instance I and a parameter k is called *fixed-parameter tractable* if it can be solved in $f(k) \cdot |I|^{O(1)}$ time for a computable function f which typically grows at least exponentially in k. This still means, however, that the problem is efficiently solvable for small parameter values.

Parameterized complexity analysis, which still adheres to a worst-case complexity scenario, has been successfully applied in many areas. Computational Social Choice, although being a more recent application area [5], is among the most natural ones. To make this more precise, we next discuss in more detail three prominent voting problems that were already briefly mentioned in the introductory part: winner determination, campaign management/bribery, and control.

An election $E := (C, V)$ consists of a set C of m alternatives $c_1, c_2, \ldots, c_m$ and a list V of n voters $v_1, v_2, \ldots, v_n$. Each voter v has a linear order $\succ_v$ over the set C which we call a *preference order*. For example, if $C = \{c_1, c_2, c_3\}$, then the preference order $c_1 \succ_v c_2 \succ_v c_3$ of voter v indicates that v likes c_1 most (the 1st position), then c_2, and c_3 least (the 3rd position). For any two distinct alternatives c and c', we write $c \succ_v c'$ if voter v prefers c over c'. For an election $E = (C, V)$, an alternative $c \in C$ is a *Condorcet winner* if any other alternative $c' \in C \setminus \{c\}$ satisfies

$$|\{v \in V \mid c \succ_v c'\}| > |\{v \in V \mid c' \succ_v c\}|.$$

It is important to note that voting problems typically come along with many natural parameterizations, the two most obvious ones being the number of candidates (which is typically small in political elections, say) or the number of voters (which is often small in applications concerning multi-agent systems or decision making by committees). A further, standard type of parameter refers to the size (or the value) of the solution that we seek.

We now give an example of a winner determination problem, focusing on the voting rule due to Dodgson (also known as the writer Lewis Carroll). In Dodgson's system, the score of a candidate is the smallest number of swaps of adjacent candidates needed to ensure that this candidate is a Condorcet winner. In the DODGSON SCORE problem we ask about the score of a given candidate in an election:

Input: An election $E := (C, V)$, a distinguished alternative $c \in C$, and a nonnegative integer k.
Question: Can one make c the Condorcet winner by swapping a total number of at most k pairs of neighboring alternatives (i.e., k "bubble sort operations") in the voters' preference orders?

Our campaign management/bribery example is based on the t-Approval voting rule. In t-Approval, every voter can assign one point to each of t most preferred alternatives, and the alternatives with maximum total number of points win. Notably, 1-Approval simply is the frequently used Plurality voting rule. Now, the NP-hard problem SWAP BRIBERY FOR t-APPROVAL reads as follows.

Input: An election $E := (C, V)$, a distinguished alternative $c \in C$, a cost function assigning a nonnegative integer cost to every swap-of-consecutive-candidates operation, and a nonnegative integer β called the budget.
Question: Can one make c a winner by swap operations of total cost at most β?

Intuitively, each swap operation means a campaigning effort that convinces the voter that one candidate is better than the other and comes at a given cost (measured in time or money or some other way).

Finally, our control example focuses on control by adding alternatives in an election based on the plurality voting role. Note that other control actions include, for example, deleting alternatives or adding/deleting voters (assuming a powerful and corrupted chair of an election). The goal of the chair can either be to ensure someone's victory (constructive control) or preclude someone from winning (destructive control); we focus on the former. The NP-hard problem PLURALITY CONSTRUCTIVE CONTROL BY ADDING ALTERNATIVES reads as follows:

Input: An election $E := (C, V)$, a distinguished alternative $c \in C$, a set of "spoiler alternatives" for possible addition, and a nonnegative integer k.
Question: Can one make c a winner by adding at most k spoiler alternatives?

Note that we assume that every voter has a clear linear order over all alternatives (including the spoiler alternatives) and that this is known by the manipulating election chair.

Key Results

We again start with our winner determination example. Bartholdi, Tovey, and Trick [1] were the first to provide an "ILP-based" fixed-parameter tractability result in the context of Computational Social Choice (actually the result was stated implicitly). They developed an integer linear program (ILP) to solve the NP-hard DODGSON SCORE problem and gave a running time bound based on a famous result of Lenstra, concerning the exact solvability of integer linear programs with "few" variables. Without having explicitly stated this in their publication, this implies fixed-parameter tractability for DODGSON SCORE with respect to the parameter number m of alternatives.

Bartholdi et al. [1]'s integer linear program for DODGSON SCORE reads as follows. It computes the Dodgson score of an alternative c.

$$\min \sum_{i,j} j \cdot x_{i,j} \text{ subject to}$$

$$\forall i \in \tilde{V} : \sum_{j} x_{i,j} = N_i,$$

$$\forall y \in C : \sum_{i,j} e_{i,j,y} \cdot x_{i,j} \geq d_y,$$

$$x_{i,j} \geq 0.$$

Here, $\tilde{V}$ denotes the set of preference order types (i.e., the set of *different* preference orders in the given election), N_i denotes the number of voters of type i, $x_{i,j}$ denotes the number of voters with preference order of type i for which alternative c will be moved upward by j positions,

$e_{i,j,y}$ is 1 if the result of moving alternative c by j positions upward in a preference order of type i is that c gains an additional voter support against alternative y, and 0 otherwise. Furthermore, d_y is the deficit of c with respect to alternative y, that is, the minimum number of voter supports that c must gain against y to defeat y in a pairwise comparison. If c already defeats y, then $d_y = 0$. Altogether, the integer linear program contains at most $m \cdot m!$ variables $x_{i,j}$, where m denotes the number of alternatives. Thus, the number of variables in the described integer linear program is upper-bounded by a function in parameter m, yielding fixed-parameter tractability due to Lenstra's result.

We remark that beyond the above parameterization by the number m of alternatives, it is also known that DODGSON SCORE is fixed-parameter tractable with respect to the parameter total number of swaps [4] (this is an example of a parameter that measures the solution value).

Now we briefly discuss some parameterized complexity results for SWAP BRIBERY FOR t-APPROVAL due to Dorn and Schlotter [8]. The SWAP BRIBERY problem was introduced by Elkind et al. [9], who have shown that the problem is NP-complete for a variety of voting rules, including t-Approval (for $t \geq 2$). Dorn and Schlotter provided a detailed discussion of the complexity of the problem for t-Approval, and, in particular, they have shown that the problem is fixed-parameter tractable when parameterized by the number of voters. On the contrary, if we take t to be the parameter (i.e., the problem no longer considers a single, fixed voting rule but a whole family of them), then the problem is W[1]-hard (and, unless unlikely complexity class collapses occur, the problem is not fixed-parameter tractable).

The complexity of control problems was first studied by Bartholdi, Tovey, and Trick [2], who – in particular – have shown that PLURALITY CONSTRUCTIVE CONTROL BY ADDING ALTERNATIVES (PCCAC) is NP-complete. Control problems have received relatively little attention from the parameterized complexity perspective. For example, among other issues, Betzler and Uhlmann [3] considered parameterization by the number of voters (for Copeland voting rule, which we do not discuss here). Liu et al. [10] considered the parameterization by the solution size (i.e., the number of candidates to be added) and, among other results, obtained W[2]-hardness for PCCAC (in essence, this already follows from the proof of Bartholdi, Tovey, and Trick).

Open Problems

1. We sketched the ILP-based fixed-parameter tractability result for DODGSON SCORE. A key question for this and many other problems shown to be fixed-parameter tractable using Lenstra's result is whether the (impractical) ILP formulation can be replaced by a direct combinatorial algorithm (still providing fixed-parameter tractability); we point to a recent survey [6] for a broader exposition on that. Concerning DODGSON SCORE, it is also interesting to settle its parameterized complexity with respect to the number of votes [4].
2. One of the most intriguing questions regarding the complexity of SWAP BRIBERY is whether the problem (for some given voting rule) is fixed-parameter tractable when parameterized by the number of candidates (or, if not, then if at least there is a fixed-parameter tractable approximation scheme; interestingly, for SHIFT BRIBERY, a significantly simpler variant of the problem, such an approximation scheme indeed exists [7]). Dorn and Schlotter [8] showed that this problem is fixed-parameter tractable, but their proof only applies (in essence) to the case where each swap has the same cost.
3. While there is quite a number of NP-completeness results regarding control problems and various voting rules, there are relatively few parameterized results. Can one

turn some of these NP-completeness results into fixed-parameter tractability results for some natural parameters?

A recent survey article [6] contains several more research challenges concerning the parameterized complexity of problems from Computational Social Choice.

Recommended Reading

1. Bartholdi JJ III, Tovey CA, Trick MA (1989) Voting schemes for which it can be difficult to tell who won the election. Soc Choice Welf 6(2): 157–165
2. Bartholdi JJ III, Tovey CA, Trick MA (1992) How hard is it to control an election? Math Comput Model 16(8/9):27–40
3. Betzler N, Uhlmann J (2009) Parameterized complexity of candidate control in elections and related digraph problems. Theor Comput Sci 410(52): 43–53
4. Betzler N, Guo J, Niedermeier R (2010) Parameterized computational complexity of Dodgson and Young elections. Inf Comput 208(2): 165–177
5. Betzler N, Bredereck R, Chen J, Niedermeier R (2012) Studies in computational aspects of voting – a parameterized complexity perspective. In: Bodlaender HL, Downey R, Fomin FV, Marx D (eds) The multivariate algorithmic revolution and beyond. LNCS, vol 7370. Springer, Berlin/New York, pp 318–363
6. Bredereck R, Chen J, Faliszewski P, Guo J, Niedermeier R, Woeginger GJ (2014) Parameterized algorithmics for computational social choice: nine research challenges. Tsinghua Sci Technol 19(4):358–373
7. Bredereck R, Chen J, Nichterlein A, Faliszewski P, Niedermeier R (2014) Prices matter for the parameterized complexity of shift bribery. In: Proceedings of the 28th Conference on Artificial Intelligence (AAAI'14), Quebéc City, pp 1398–1404
8. Dorn B, Schlotter I (2012) Multivariate complexity analysis of swap bribery. Algorithmica 64(1):126–151
9. Elkind E, Faliszewski P, Slinko A (2009) Swap bribery. In: Proceedings of the 2nd International Symposium on Algorithmic Game Theory (SAGT '09). LNCS, vol 5814. Springer, Berlin/Heidelberg, pp 299–310
10. Liu H, Feng H, Zhu D, Luan J (2009) Parameterized computational complexity of control problems in voting systems. Theor Comput Sci 410(27–29):2746–2753

Parameterized Algorithms for Drawing Graphs

Henning Fernau
Fachbereich 4, Abteilung Informatik wissenschaften, University of Trier, Trier, Germany
Institute for Computer Science, University of Trier, Trier, Germany

Years and Authors of Summarized Original Work

2004; Dujmovic, Whitesides

Problem Definition

ONE-SIDED CROSSING MINIMIZATION (OSCM) can be viewed as a specific form of drawing a bipartite graph $G = (V_1, V_2, E)$, where all vertices from partition V_i are assigned to the same line (also called layer) L_i in the plane, with L_1 and L_2 being parallel. The vertex assignment to L_1 is fixed, while that to L_2 is free and should be chosen in a way to minimize the number of crossings between etdes drawn as straight-line segments.

Notations

A graph G is described by its vertex set V and its edge set E, i.e., G=(V, E), with $E \subseteq V \times V$. The (open) *neighborhood* of a vertex v, denoted $N(v)$, collects all vertices that are adjacent to v. $N[v] = N(v) \cup \{v\}$ denotes the *closed neighborhood* of v. $\deg(v) = |N(v)|$ is the *degree* of v. For a vertex set S, $N(S) = \bigcup_{v \in S} N(v)$, and $N[S] = N(S) \cup S$. $G[S]$ denotes the graph induced by vertex set S, i.e., $G[S] = (S, E \cap (S \times S))$. A graph $G = (V, E)$ with vertex set V and edge set $E \subseteq V \times V$ is *bipartite* if there is a partition of V into two sets V_1 and V_2 such that $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, and $E \subseteq V_1 \times V_2$. For clarity, $G = (V_1, V_2, E)$ is written in this case.

A *two-layer drawing* of a bipartite graph $G = (V_1, V_2, E)$ can be described by two linear orders $<_1$ on V_1 and $<_2$ on V_2. This drawing can be realized as follows: the vertices of V_1 are placed on a line L_1 (also called *layer*) in the order induced by $<_1$ and the vertices of V_2 are placed on a second layer L_2 (parallel to the first one) in the order induced by $<_2$; then, draw a straight-line segment for each edge $e = (u_1, u_2)$ in E connecting the points that represent u_1 and u_2, respectively. A *crossing* is a pair of edges $e = (u_1, u_2)$ and $f = (v_1, v_2)$ that cross in the realization of a two-layer drawing $(G, <_1, <_2)$. It is well-known that two edges cross if and only if $u_1 <_1 v_1$ and $v_2 <_2 u_2$; in other word, this notion is a purely combinatorical object, independent of the concrete realization of the two-layer drawing. $\mathrm{cr}(G, <_1, <_2)$ denotes the number of crossings in the described two-layer drawing. In the graph drawing context, it is of course desirable to draw graphs with few crossings. In its simplest (yet probably most important) form, the vertex order in one layer is fixed, and the aim is to minimize crossings by choosing an order of the second layer. Formally, this means:

Problem 1 (k–OSCM)
INPUT: A simple n-vertex bipartite graph $G = (V_1, V_2, E)$ and a linear order $<_1$ on V_1, a nonnegative integer k (the parameter).
OUTPUT: If possible, a linear order $<_2$ on V_2 such that $\mathrm{cr}(G, <_1, <_2) \leq k$. If no such order exists, the algorithm should tell so.

Given an instance $G = (V_1, V_2, E)$ and $<_1$ of OSCM and two vertices $u, v \in V_2$,

$$c_{uv} = \mathrm{cr}(G[N[\{u, v\}]], <_1 \cap (N(\{u, v\}) \times N(\{u, v\})), \{(u, v)\}) .$$

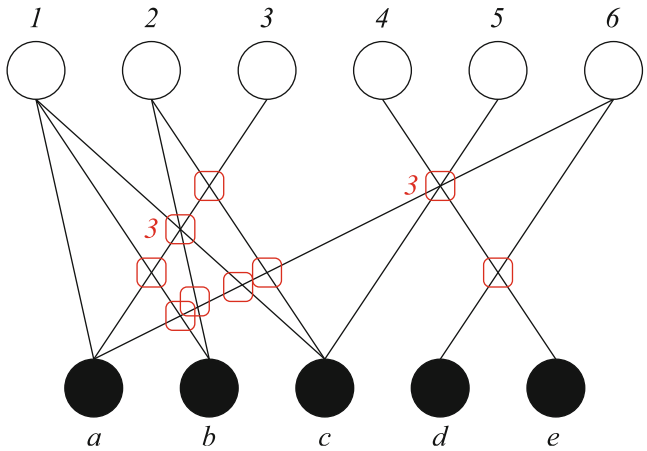
Hence, the closed neighborhoods of u and v are considered when assuming the ordering $u <_2 v$.

Consider the following as a running example:

Example 1 In Fig. 1, a concrete drawing of a bipartite graph is shown. Is this drawing optimal with respect to the number of crossings, assuming the ordering of the upper layer being fixed? At some points, more than two edges cross; in that case, a number is shown to count the crossings. All crossings are emphasized by a surrounding box.

Let us now compute the *crossing number matrix* (c_{uv}) for this graph.

c_{uv}	a	b	c	d	e
a	–	4	5	0	1
b	1	–	1	0	0
c	3	3	–	0	1
d	3	2	3	–	1
e	2	3	2	0	–



Parameterized Algorithms for Drawing Graphs, Fig. 1 The running example for OSCM

P

The number of crossings in the given drawing can be hence computed as

$$c_{ab} + c_{ac} + c_{ad} + c_{ae} + c_{bc} + c_{bd} + c_{be} + c_{cd} + c_{ce} + c_{de} = 13.$$

Key Results

Exact exponential-time algorithms are mostly interesting when dealing with problems for which no polynomial-time algorithm is expected to exist.

Theorem 1 ([6]) *The decision problem corresponding to k-OSCM is $\mathcal{NP}$-complete.*

In the following, to state the results, let $G = (V_1, V_2, E)$ be an instance of OSCM, where the ordering $<_1$ of V_1 is fixed.

It can be checked in polynomial time if an order of V_2 exists that avoids any crossings. This observation can be based on either of the following graph-theoretic characterizations:

Theorem 2 ([3]) $\mathrm{cr}(G, <_1, <_2) = 0$ *if and only if G is acyclic and, for every path (x, a, y) of G with $x, y \in V_1$, it holds: for all $u \in V_1$ with $x <_1 u <_1 y$, the only edge incident to u (if any) is (u, a).*

The previously introduced notion is crucial due to the following facts:

Lemma 3 $\sum_{u,v \in V_2, u <_2 v} c_{uv} = \mathrm{cr}(G, <_1, <_2)$.

Theorem 4 ([9]) *If k is the minimum number of edge crossings in an OSCM instance $(G = (V_1, V_2, E), <_1)$, then*

$$\sum_{u,v \in V_2, u \neq v} \min\{c_{uv}, c_{vu}\} \leq k < 1.4664 \sum_{u,v \in V_2, u \neq v} \min\{c_{uv}, c_{vu}\}.$$

In fact, Nagamochi also presented an approximation algorithm with a factor smaller than 1.4664.

Furthermore, for any $u \in V_2$ with $\deg(u) > 0$, let l_u be the leftmost neighbor of u on L_1, and r_u be the rightmost neighbor of u. Two vertices $u, v \in V_2$ are called *unsuited* if there exists some $x \in N(u)$ with $l_v <_1 x <_1 r_v$, or there exists some $x \in N(v)$ with $l_u <_1 x <_1 r_u$. Otherwise, they are called *suited*. Observe that, for $\{u, v\}$ suited, $c_{uv} \cdot c_{vu} = 0$. Dujmović and Whitesides have shown:

Lemma 5 ([5]) *In any optimal ordering $<_2$ of the vertices of V_2, $u <_2 v$ is found if $r_u \leq_1 l_v$.*

This means that all suited pairs appear in their *natural ordering*.

This already allows us to formulate a first parameterized algorithm for OSCM, which is a simple search tree algorithm. In the course of this algorithm, a suitable ordering $<_2$ on V_2 is gradually constructed; when settling the ordering between u and v on V_2, $u <_2 v$ or $v <_2 u$ is *committed*. A *generalized instance* of OSCM therefore contains, besides the bipartite graph $G = (V_1, V_2, E)$, a partial ordering $<_2$ on V_2. A vertex $v \in V_2$ is *fully committed* if, for all $u \in V_2 \setminus \{u, v\}$, $\{u, v\}$ is committed.

Lemma 5 allows us to state the following rule:

RR1: For every pair of vertices $\{u, v\}$ from V_2 with $c_{uv} = 0$, commit $u <_2 v$. In the example, d would be fully committed by applying RR1, since the d-column in the crossing number matrix is all zeros; hence, ignore d in what follows.

Algorithm 1 is a simple search tree algorithm for OSCM that repeatedly uses Rule RR1.

Lemma 6 *OSCM can be solved in time $\mathcal{O}^*(2^k)$.*

Proof Before any branching can take place, the graph instance is reduced, so that every pair of vertices $\{u, v\}$ from V_2 which is not committed satisfies $\min\{c_{uv}, c_{vu}\} \geq 1$. Therefore, each recursive branch reduces the parameter by at least one. □

It is possible to improve on this very simple search tree algorithm. A first observation is that it is not necessary to branch at $\{x, y\} \subset V_2$ with $c_{xy} = c_{yx}$. This means two modifications to Algorithm 1:

Algorithm 1 A search tree algorithm solving OSCM, called OSCM-ST-simple

Require: a bipartite graph $G = (V_1, V_2, E)$, an integer k, a linear ordering $<_1$ on V_1, a partial ordering $<_2$ on V_2
Ensure: YES iff the given OSCM instance has a solution

```
repeat
    Exhaustively apply the reduction rules, adjusting <_2 and k accordingly.
    Determine the vertices whose order is settled by transitivity and adjust <_2 and k accordingly.
until there are no more changes to <_2 and to k
if k < 0 or <_2 contains both (x, y) and (y, x) then
    return NO.
else if ∃{x, y} ⊆ V_2 : neither x <_2 y nor y <_2 x is settled then
    if OSCM-ST-simple(G, k − 1, <_1, <_2 ∪ {(x, y)}) then
        return YES
    else
        return OSCM-ST-simple(G, k − 1, <_1, <_2 ∪ {(y, x)})
    end if
else
    return YES
end if
```

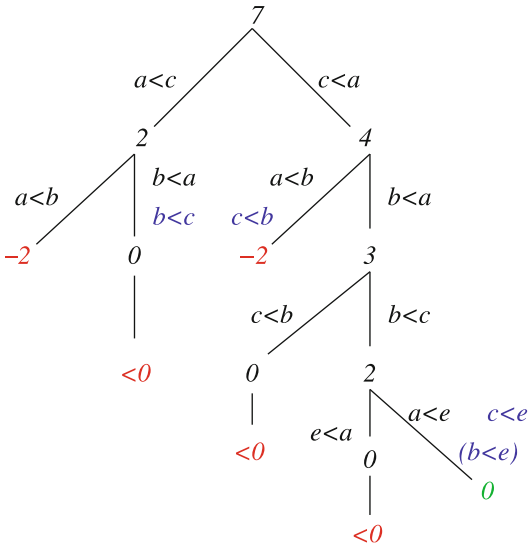
- Line 5 should exclude $c_{xy} = c_{yx}$.
- Line 12 should arbitrary commit some $\{x, y\} \subset V_2$ that are not yet committed and recurse; only if all $\{x, y\} \subset V_2$ are committed, YES is to be returned.

These modifications immediately yield an $\mathcal{O}^*(1.6182^k)$ algorithm for OSCM. This is also the core of the algorithm proposed by Dujmović and Whitesides [5]. There, more details are discussed, as, for example:

- How to efficiently calculate all the crossing numbers c_{xy} in a preprocessing phase.
- How to integrate branch and cut elements in the algorithm that are surely helpful from a practical perspective.
- How to generalize the algorithm for instances that allow integer weights on the edges (multiple edges).

Further improvements are possible if one gives a deeper analysis of local patterns $\{x, y\} \in V_2$ such that $c_{xy}c_{yx} \leq 2$. This way, it has been shown:

Theorem 7 ([4]) *OSCM can be solved in time* $\mathcal{O}^*(1.4656^k)$.



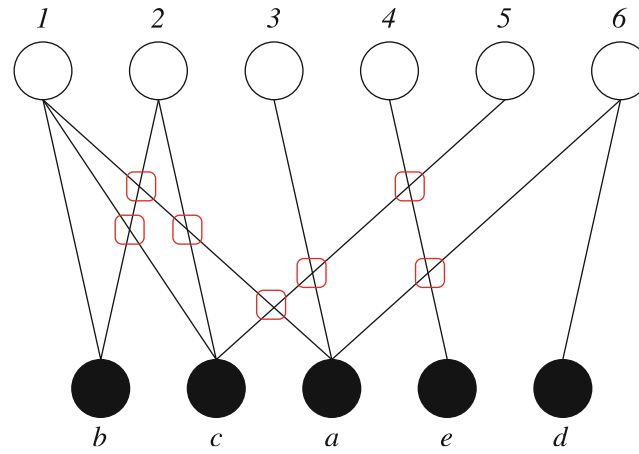
Parameterized Algorithms for Drawing Graphs, Fig. 2 A search tree example for OSCM

A possible run of the improved search tree algorithm is displayed in Fig. 2, with the (optimal) outcome shown in Fig. 3.

Variants and Related Problems have been discussed in the literature.

1. Change the goal of the optimization: minimize the number of edges involved in cross-

P



Parameterized Algorithms for Drawing Graphs, Fig. 3 An optimal solution to the example instance

ings (ONE-LAYER PLANARIZATION (OLP)). As observed in [7, 10], Theorem 2 almost immediately leads to an $\mathcal{O}^*(3^k)$ algorithm for OLP that was subsequently improved down to $\mathcal{O}^*(2^k)$ in [10].

2. One could allow more degrees of freedom by considering two (or more) layer assignments at the same time. For both the crossing minimization and the planarization variants, parameterized algorithms are reported in [3, 7, 10].
3. One can consider other additional constraints on the drawings or the admissible orderings; in [8], parameterized algorithms for two-layer assignment problems are discussed where the admissible orderings are restricted by binary trees.

Applications

Besides seeing the question of drawing bipartite graphs as an interesting problem in itself, e.g., for nice drawings of relational diagrams, this question quite naturally shows up in the so-called Sugiyama approach to hierarchical graph drawing, see [12]. This very popular approach tackles the problem of laying out a hierarchical graph in three phases: (1) cycle removal (2) assignment of verticesto layers, (3) assignment of vertices to layers. The last phase is usually performed in a sweep-line fashion, intermediately solving many instances of OSCM. The third variant in the discussion above has important applications in computational biology.

Open Problems

As with all exponential-time algorithms, it is always a challenge to further improve on the running times of the algorithms or to prove lower bounds on those running times under reasonable complexity theoretic assumptions. Let us notice that the tacit assumptions underlying the approach by parameterized algorithmics are well met in this application scenario: e.g., one would not accept drawings with many crossings anyways (if such a situation is encountered in practice, one would switch to another way of representing the information); so, one can safely assume that the parameter is indeed small.

This is also true for other $\mathcal{NP}$-hard subproblems that relate to the Sugiyama approach. However, no easy solutions should be expected. For example, the DIRECTED FEEDBACK ARC SET PROBLEM [1] that is equivalent to the first phase is not known to admit a nice parameterized algorithm, see [2].

Experimental Results

Suderman [10] reports on experiments with nearly all problem variants discussed above, also see [11] for a better accessible presentation of some of the experimental results.

URL to Code

Suderman presents several JAVA applets related to the problems discussed in this article, see http://cgm.cs.mcgill.ca/~msuder/.

Cross-References

Other parameterized search tree algorithms are explained in the contribution ▶ Vertex Cover Search Trees by Chen, Kanj, and Jia.

Recommended Reading

1. Chen J, Liu Y, Lu S, O'Sullivan B, Razgon I (2008) A fixed-parameter algorithm for the directed feedback vertex set problem. In: 40th ACM symposium on theory of computing STOC 2008, Victoria, 17–20 May 2008
2. Downey RG, Fellows MR (1999) Parameterized complexity. Springer, Berlin
3. Dujmovic V, Fellows MR, Hallett M, Kitching M, Liotta G, McCartin C, Nishimura N, Ragde P, Rosamond FA, Suderman M, Whitesides S, Wood DR (2006) A fixed-parameter approach to 2-layer planarization. Algorithmica 45: 159–182
4. Dujmovic V, Fernau H, Kaufmann M (2004) Fixed parameter algorithms for one-sided crossing minimization revisited. In: Liotta G (ed) Graph drawing, 11th international symposium GD 2003. LNCS, vol 2912. Springer, Berlin, pp 332–344, A journal version has been accepted to J Discret Algorithms see doi:10.1016/j.jda.2006.12.008
5. Dujmovic V, Whitesides S (2004) An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. Algorithmica 40:15–32
6. Eades P, Wormald NC (1994) Edge crossings in drawings of bipartite graphs. Algorithmica 11: 379–403
7. Fernau H (2005) Two-layer planarization: improving on parameterized algorithmics. J Graph Algorithms Appl 9:205–238
8. Fernau H, Kaufmann M, Poths M (2005) Comparing trees via crossing minimization. In: Ramanujam R, Sen S (eds) Foundations of software technology and theoretical computer science FSTTCS 2005. LNCS, vol 3821. Springer, Berlin, pp 457–469
9. Nagamochi H (2005) An improved bound on the one-sided minimum crossing number in two-layered drawings. Discret Comput Geom 33:569–591
10. Suderman M (2005) Layered graph drawing. PhD thesis, McGill University, Montréal
11. Suderman M, Whitesides S (2005) Experiments with the fixed-parameter approach for two-layer planarization. J Graph Algorithms Appl 9(1): 149–163
12. Sugiyama K, Tagawa S, Toda M (1981) Methods for visual understanding of hierarchical system structures. IEEE Trans Syst Man Cybern 11(2): 109–125

Parameterized Pattern Matching

Moshe Lewenstein
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Years and Authors of Summarized Work

1994; Amir, Farach, Muthukrishnan
1995; Kosaraju
1996; Baker

Problem Definition

Parameterized strings, or *p-strings*, are strings that contain both ordinary symbols from an alphabet Σ and parameter symbols from an alphabet Π. Two equal-length p-strings s and s' are a parameterized match, or *p-match*, if one p-string can be transformed into the other by applying a one-to-one function that renames the parameter

symbols. The following example of a p-match is one with both ordinary and parameter symbols. The ordinary symbols are in lowercase and the parameter symbols are in uppercase:

$$s = A\ b\ A\ b\ C\ A\ d\ b\ A\ C\ d\ d$$
$$s' = D\ b\ D\ b\ E\ D\ d\ b\ D\ E\ d\ d$$

In some of the problems to be considered, it will be sufficient to solve for p-strings in which all symbols are parameter symbols, as this is the more difficult part of the problem. In other words, the case in which $\Sigma = \emptyset$. In this case, the definition can be reformulated so that s and s' are a p-match if there exists a bijection $\pi : \Pi_s \rightarrow \Pi'_s$, such that $\pi(s) = s'$, where $\pi(s)$ is the renaming of each character of s via π.

The following problems will be considered. *Parameterized matching* – given a parameterized pattern p of length m and parameterized text t, find all locations i of a parameterized text t for which p p-matches $t_i \ldots t_{i+m-1}$, where $m = |p|$. The same problem is also considered in two dimensions. *Approximate parameterized matching* – find all substrings of a parameterized text t that are approximate parameterized matches of a parameterized pattern p (to be fully defined later).

Key Results

Baker [4] introduced parameterized matching in the framework of her seminal work on discovering duplicate code within large programs for the sake of code minimization. An example of two code fragments that p-match taken from the X Windows system can be found in [4].

Parameterized Suffix Trees

In [4] and in the follow-up journal versions [6,7], a novel method was presented for parameterized matching by constructing *parameterized suffix trees*. The advantage of the parameterized suffix tree is that it supports indexing, i.e., one can preprocess a text and subsequently answer parameterized queries p in $O(|p|)$ time. In order to achieve parameterized suffix trees, it is necessary to introduce the concept of a *predecessor string*. A *predecessor string* of a string s has at each location i the distance between i and the location containing the previous appearance of the symbol. The first appearance of each symbol is replaced with a 0. For example, the predecessor string of *aabbaba* is 0,1,0,1,3,2,2. A simple and well-known fact is that:

Observation 1 ([7]) s and s' p-match if and only if they have the same predecessor string.

Notice that this implies transitivity of parameterized matching, since if s and s' p-match and s' and s'' p-match, then, by the observation, s and s' have the same predecessor string and, likewise, s' and s'' have the same predecessor string. This implies that s and s'' have the same predecessor string and hence, by the observation, p-match.

Moreover, one may also observe that if r is a prefix of s, then the predecessor string of r, by definition, is exactly the $|r|$-length prefix of the predecessor string of s. Hence, similar to regular pattern matching, a parameterized pattern p p-matches at location i of t if and only if the $|p|$-length predecessor string of p is equal to the $|p|$-length prefix of the predecessor string of the suffix $t_i \ldots t_n$. Combining these observations, it is natural to do as follows: create a (parameterized suffix) tree with a leaf for each suffix where the path from the root to the leaf corresponding to a given suffix will have its predecessor string labeling the path. Branching in the parameterized suffix tree, as with suffix trees, occurs according to the labels of the predecessor strings. See [4,6,7] for an example.

Baker's method essentially mimics the McCreight suffix tree construction [18]. However, while the suffix tree and the parameterized suffix tree are very similar, there is a slight hitch. A strong component of the suffix tree construction is the suffix link. This is used for the construction and, sometimes, for later pattern searches. The suffix link is based on the *distinct right context* property, which does not hold for the parameterized suffix tree. In fact, the node that is pointed

to by the suffix link may not even exist. The main parts of [6, 7] are dedicated to circumventing this problem.

In [7] Baker added the notion of "bad" suffix links, which point to the vertex just above, i.e., closer to the root than the desired place, and of updating them with a lazy evaluation when they are used. The algorithm runs in time $O(n|\Pi|\log|\Sigma|)$. In [6] (which is chronologically later than [7] despite being the first to appear) Baker changed the definition of "bad" suffix links to point to just below the desired place. This turns out to have nice properties, and one can use more sophisticated data structures to improve the construction time to $O(n(|\Pi| + \log|\Sigma|))$.

Kosaraju [16] made a careful analysis of Baker's properties utilized in the algorithm of [6] which suffer from the $|\Pi|$ factor. He pointed out two sources for this large factor. He handled these two issues by using a concatenable queue and maintaining it in a lazy manner. This is sufficient to reduce the $|\Pi|$ factor to a $\log|\Pi|$ factor, yielding an algorithm of time $O(n(\log|\Pi| + \log|\Sigma|))$.

Obviously if the alphabet or symbol set is large, the construction time may be $O(n \log n)$. Cole and Hariharan [9] showed how to construct the parameterized suffix trees in randomized $O(n)$ time for alphabets and parameters taken from a polynomially sized range, e.g., $[1, \ldots, n^c]$. They did this by adding additional nodes to the tree in a back-propagation manner which is reminiscent of fractional cascading. They showed that this adds only $O(n)$ nodes and allows the updating of the missing suffix links. However, this causes other problems and one may find the details of how this is handled in their paper.

More Methods for Parameterized Matching

Obviously the parameterized suffix tree efficiently solves the parameterized matching problem. Nevertheless, a couple of other results on parameterized matching are worth mentioning.

First, in [6] it was shown how to construct the parameterized suffix tree for the pattern and then to run the parameterized text through it, giving an algorithm with $O(m)$ space instead of $O(n)$.

Amir et al. [2] presented a simple method to solve the parameterized matching problem by mimicking the algorithm of Knuth, Morris, and Pratt. Their algorithm works in $O(n * \min(\log|\Pi|, m))$ time independent of the alphabet size ($|\Sigma|$). Moreover, they proved that the log factor cannot be avoided for large symbol sets.

In [5] parameterized matching was solved with a Boyer-Moore type algorithm. In [10] the problem was solved with a Shift-Or type algorithm. Both handle the average case efficiently. In [10] emphasis was also put on the case of multiple parameterized matching, which was previously solved in [14] with an Aho-Corasick automaton-style algorithm.

Two-Dimensional Parameterized Matching

Two-dimensional parameterized matching arises in applications of image searching; see [13] for more details. Two-dimensional parameterized matching is the natural extension of parameterized matching where one seeks pmatches of a two-dimensional parameterized pattern p within a two-dimensional parameterized text t. It must be pointed out that classical methods for two-dimensional pattern matching, such as the L-suffix tree method, fail for parameterized matching. This is because known methods tend to cut the text and pattern into pieces to avoid going out of boundaries of the pattern. This is fine because each pattern piece can be individually evaluated (checked for equality) to a text piece. However, in parameterized matching, there is a strong dependency between the pieces.

In [1] an innovative solution for the problem was given based on a collection of linearizations of the pattern and text with the property to be currently described. Consider a linearization. Two elements with the same character, say "a," in the pattern are defined to be neighbors if there is no other "a" between them in this linearization. Now take all the "a"s of the pattern and create a graph G_a with "a"s as the nodes and edges between the two if they are

neighbors in some linearization. We say that two "a"s are chained if there is a path from one to the other in G_a. Applying one-dimensional parameterized matching on these linearizations ensures that any two elements that are chained will be evaluated to map to the same text value (the parameterized property). A collection of linearizations has the *fully chained* property if every two locations in p with the same character are chained. It was shown in [1] that one can obtain a collection of $\log\ m$ linearizations that is fully chained and that does not exceed pattern boundary limits. Each such linearization is solved with a convolution-based pattern-matching algorithm. This takes $O(n^2 \log\ m)$ time for each linearization, where the text size is n^2. Hence, overall the time is $O(n^2 \log^2 m)$.

A different solution was proposed in [13], where it was shown that it is possible to solve the problem in $O(n^2 + m^{2.5}\text{polylog}\ m)$, where the text size is $O(n^2)$ and the pattern size is $O(m^2)$. Clearly, this is more efficient for large texts.

Approximate Parameterized Matching

Our last topic relates to parameterized matching in the presence of errors. Errors occur in various applications and it is natural to consider parameterized matching with the Hamming distance metric or the edit distance metric.

In [8] the parameterized matching problem was considered in conjunction with the edit distance. Here the definition of edit distance was slightly modified so that the edit operations are defined to be insertion, deletion, and parameterized replacements, i.e., the replacement of a substring with a string that p-matches it. An algorithm for finding the "parameterized edit distance" of two strings was devised whose efficiency is close to the efficiency of the algorithms for computing the classical edit distance.

However, it turns out that the operation of parameterized replacement relaxes the problem to an easier problem. The reason that the problem becomes easier is that two substrings that participate in two parameterized replacements are independent of each other (in the parameterized sense).

A more rigid, but more realistic, definition for the Hamming distance variant was given in [3]. For a pair of equal-length strings s and s' and a bijection π defined on the alphabet of s, the π-mismatch is the Hamming distance between the image under π of s and s'. The minimal π-mismatch over all bijections π is the approximate parameterized match. The problem considered in [3] is to find for each location i of a text t the approximate parameterized match of a pattern p with the substring beginning at location i. In [3] the problem was defined and linear-time algorithms were given for the case where the pattern is binary or the text is binary. However, this solution does not carry over to larger alphabets.

Unfortunately, under this definition, the methods for classical string matching with errors for the Hamming distance, also known as pattern matching with mismatches, seem to fail. Following is an outline of a classical method [17] for pattern matching with mismatches that uses suffix trees.

The pattern is compared separately with each suffix of the text, beginning at locations $1 \leq i \leq n - m + 1$. Using a suffix tree of the text and precomputed longest common ancestor information (which can be computed once in linear time [11]), one can find the longest common prefix of the pattern and the corresponding suffix (in constant time). There must be a mismatch immediately afterwards. The algorithm jumps over the mismatch and repeats the process, taking into consideration the offsets of the pattern and suffix.

When attempting to apply this technique to a parameterized suffix tree, it fails. To illustrate this, consider the first matching substring (up until the first error) and the next matching substring (after the error). Both of these substrings p-match the substring of the text that they are aligned with. However, it is possible that combined they do not form a p-match. See the example below. In the example *abab* p-matches *cdcd* followed by a mismatch and subsequently followed by *abaa* p-matching *efee*. However, different π's are required for the local p-matches. This example also emphasizes why the definition of [8] is a simplification. Specifically, each local p-matching

substring is one replacement, i.e., *abab* with *cdcd* is one replacement and *abaa* with *efee* is one more replacement. However, the definition of [3] captures the globality of the parameterized matching, not allowing, in this case, *abab* to p-match to two different substrings.

$$p = a\ b\ a\ b\ a\ a\ b\ a\ a \ldots$$
$$t = \ldots c\ d\ c\ d\ d\ e\ f\ e\ e \ldots \cdot$$

In [12] the problem of *parameterized matching with k mismatches* was considered. The parameterized matching problem with k mismatches seeks all locations i in text t where the minimal π-mismatch between p to $t_i \ldots t_{i+m-1}$ has at most k mismatches. An $O(nk^{1.5} + mk \log m)$ time algorithm was presented in [12]. At the base of the algorithm, i.e., for the case where $|p| = |t| = m$, an $O(m + k^{1.5})$ algorithm is used based on maximum matching algorithms. Then the algorithm uses a doubling scheme to handle the growing distance between potential parameterized matches (with at most k mismatches). Also shown in [12] is a strong relationship between maximum matching algorithms in sparse graphs and parameterized matching with k errors.

The rigid, but more realistic, definition for the Hamming distance version given in [3] can be naturally extended to the edit distance. Lately, it was shown that this problem is nondeterministic polynomial-time complete [15].

Applications

Parameterized matching has applications in code duplication detection in programming languages, in homework plagiarism detection, and in image processing, among others [1, 4].

Cross-References

- Approximate String Matching
- Multidimensional String Matching
- String Matching
- Suffix Tree Construction

Recommended Reading

1. Amir A, Aumann Y, Cole R, Lewenstein M, Porat E (2003) Function matching: algorithms, applications and a lower bound. In: Proceedings of the 30th international colloquium on automata, languages and programming (ICALP), Eindhoven, pp 929–942
2. Amir A, Farach M, Muthukrishnan S (1994) Alphabet dependence in parameterized matching. Inf Process Lett 49:111–115
3. Apostolico A, Erdös P, Lewenstein M (2007) Parameterized matching with mismatches. J Discret Algorithms 5(1):135–140
4. Baker BS (1993) A theory of parameterized pattern matching: algorithms and applications. In: Proceedings of the 25th annual ACM symposium on the theory of computation (STOC), San Diego, pp 71–80
5. Baker BS (1995) Parameterized pattern matching by Boyer-Moore-type algorithms. In: Proceedings of the 6th annual ACM-SIAM symposium on discrete algorithms (SODA), San Francisco, pp 541–550
6. Baker BS (1996) Parameterized pattern matching: algorithms and applications. J Comput Syst Sci 52(1):28–42
7. Baker BS (1997) Parameterized duplication in strings: algorithms and an application to software maintenance. SIAM J Comput 26(5):1343–1362
8. Baker BS (1999) Parameterized diff. In: Proceedings of the 10th annual ACM-SIAM symposium on discrete algorithms (SODA), Baltimore, pp 854–855
9. Cole R, Hariharan R (2000) Faster suffix tree construction with missing suffix links. In: Proceedings of the 32nd ACM symposium on theory of computing (STOC), Portland, pp 407–415
10. Fredriksson K, Mozgovoy M (2006) Efficient parameterized string matching. Inf Process Lett 100(3):91–96
11. Harel D, Tarjan RE (1984) Fast algorithms for finding nearest common ancestor. J Comput Syst Sci 13:338–355
12. Hazay C, Lewenstein M, Sokol D (2007) Approximate parameterized matching. ACM Trans Algorithms 3(3):29
13. Hazay C, Lewenstein M, Tsur D (2005) Two dimensional parameterized matching. In: Proceedings of the 16th symposium on combinatorial pattern matching (CPM), Jeju Island, pp 266–279
14. Idury RM, Schäffer AA (1996) Multiple matching of parameterized patterns. Theor Comput Sci 154(2):203–224
15. Keller O, Kopelowitz T, Lewenstein M. Parameterized LCS and edit distance are NP-complete. Manuscript
16. Kosaraju SR (1995) Faster algorithms for the construction of parameterized suffix trees. In: Proceedings of the 36th annual symposium on foundations of computer science (FOCS), Milwaukee, pp 631–637

17. Landau GM, Vishkin U (1988) Fast string matching with k differences. J Comput Syst Sci 37(1): 63–78
18. McCreight EM (1976) A space-economical suffix tree construction algorithm. J ACM 23: 262–272

Parameterized SAT

Stefan Szeider
Department of Computer Science, Durham University, Durham, UK

Keywords

Structural parameters for SAT

Years and Authors of Summarized Original Work

2003; Szeider

Problem Definition

Much research has been devoted to finding classes of propositional formulas in conjunctive normal form (CNF) for which the recognition problem as well as the propositional satisfiability problem (SAT) can be decided in polynomial time. Some of these classes form infinite chains $C_1 \subset C_2 \subset \cdots$ such that every CNF formula is contained in some C_k for k sufficiently large. Such classes are typically of the form $C_k = \{F \in \text{CNF}: \pi(F) \leq k\}$, where π is a computable mapping that assigns to CNF formulas F a non-negative integer $\pi(F)$; we call such a mapping a *satisfiability parameter*. Since SAT is an NP-complete problem (actually, the first problem shown to be NP-complete [1]), we must expect that, the larger k, the longer the worst-case running times of the polynomial-time algorithms that recognize and decide satisfiability of formulas in C_k. Whence there is a certain tradeoff between the generality of classes and the performance guarantee for the corresponding algorithms. Szeider [12] initiates a broad investigation of this tradeoff in the conceptional framework of *parameterized complexity* [2, 3, 6]. This investigation draws attention to satisfiability parameters π for which the following holds: recognition and satisfiability decision of formulas F with $\pi(F) \leq k$ can be carried out in *uniform polynomial time*, that is, by algorithms with running time bounded by a polynomial whose order is independent of k (albeit, possibly involving a constant factor that is exponential in k). If a satisfiability parameter π allows satisfiability decision in uniform polynomial time, we say that *SAT is fixed-parameter tractable with respect to* π.

Satisfiability Parameters Based on Graph Invariants

One can define satisfiability parameters by means of certain graphs associated with CNF formulas. The *primal graph* of a CNF formula is the graph whose vertices are the variables of the formula; two variables are joined by an edge if the variables occur together in a clause. The *incidence graph* of a CNF formula is the bipartite graph whose vertices are the variables and clauses of the formula; a variable and a clause are joined by an edge if the variable occurs in the clause.

Satisfiability Parameters Based on Backdoor Sets

The concept of backdoor sets [13] gives rise to several interesting satisfiability parameters. Let C be a class of CNF formulas. A set B of variables of a CNF formula F is a *strong C-backdoor set* if for every partial truth assignment $\tau: B \to \{\text{true}, \text{false}\}$, the restriction of F to τ belongs to C. Here, the restriction of F to τ is the CNF formula obtained from F by removing all clauses that contain a literal that is true under τ and by removing from the remaining clauses all literals that are false under τ.

Key Results

Theorem 1 (Gottlob, Scarcello, and Sideri [4]) *SAT is fixed-parameter tractable with respect to the treewidth of primal graphs.*

Several satisfiability parameters that generalize the treewidth of primal graphs, such as the *treewidth and clique-width of incidence graphs*, have been studied [5, 10, 12].

The *maximum deficiency* of a CNF formula F is the number of clauses remaining exposed by a maximum matching of the incidence graph of F.

Theorem 2 (Szeider [11]) *SAT is fixed-parameter tractable with respect to maximum deficiency.*

A CNF formula is *minimal unsatisfiable* if it is unsatisfiable but removing any of its clauses makes it satisfiable. Recognition of minimal unsatisfiable formulas is DP-complete [9].

Corollary 1 (Szeider [11]) *Recognition of minimal unsatisfiable CNF formulas is fixed-parameter tractable with respect to the difference between the number of clauses and the number of variables.*

Theorem 3 (Nishimura, Ragde, and Szeider [7]) *SAT is fixed-parameter tractable with respect to the size of strong Horn-backdoor sets and with respect to the size of strong 2CNF-backdoor sets.*

Applications

Satisfiability provides a powerful and general formalism for solving various important problems including hardware and software verification and planning. Instances stemming from applications usually contain a "hidden structure" (see, e.g., [13]). The satisfiability parameters considered above are designed to make this hidden structure explicit in the form of small values for the parameter. Thus, satisfiability parameters are a way to make the hidden structure accessible to an algorithm.

Open Problems

A new line of research is concerned with the identification of further satisfiability parameters that allow a fixed-parameter tractable SAT decision are more general than the known parameters and apply well to real-world problem instances.

Cross-References

- ▸ Maximum Matching
- ▸ Treewidth of Graphs

Recommended Reading

1. Cook SA (1971) The complexity of theorem-proving procedures. In: Proceedings of the 3rd annual symposium on theory of computing, Shaker Heights, pp 151–158
2. Downey RG, Fellows MR (1999) Parameterized complexity, Monographs in computer science. Springer, Berlin
3. Flum J, Grohe M (2006) Parameterized complexity theory. Texts in theoretical computer science. An EATCS series, vol XIV. Springer, Berlin
4. Gottlob G, Scarcello F, Sideri M (2002) Fixed-parameter complexity in AI and nonmonotonic reasoning. Artif Intellect 138:55–86
5. Gottlob G, Szeider S (2007) Fixed-parameter algorithms for artificial intelligence, constraint satisfaction, and database problems. Comput J Spec Issue Parameterized Complex Adv Access
6. Niedermeier R (2006) Invitation to fixed-parameter algorithms, Oxford lecture series in mathematics and its applications. Oxford University Press, Oxford
7. Nishimura N, Ragde P, Szeider S (2004) Detecting backdoor sets with respect to Horn and binary clauses. In: Informal proceedings of SAT 2004, 7th international conference on theory and applications of satisfiability testing, Vancouver, 10–13 May 2004, pp 96–103
8. Nishimura N, Ragde P, Szeider S (2007) Solving SAT using vertex covers. Acta Inf 44(7–8):509–523
9. Papadimitriou CH, Wolfe D (1988) The complexity of facets resolved. J Comput Syst Sci 37:2–13

10. Samer M, Szeider S (2007) Algorithms for propositional model counting. In: Proceedings of LPAR 2007, 14th international conference on logic for programming, artificial intelligence and reasoning, Yerevan, 15–19 Oct 2007. Lecture notes in computer science, vol 4790. Springer, Berlin, pp 484–498
11. Szeider S (2004) Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. J Comput Syst Sci 69:656–674
12. Szeider S (2004) On fixed-parameter tractable parameterizations of SAT. In: Giunchiglia E, Tacchella A (eds) Theory and applications of satisfiability, 6th international conference, SAT 2003, selected and revised papers. Lecture notes in computer science, vol 2919. Springer, Berlin, pp 188–202
13. Williams R, Gomes C, Selman B (2003) On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In: Informal proceedings of SAT 2003 sixth international conference on theory and applications of satisfiability testing, 5–8 May 2003, S. Margherita Ligure – Portofino, pp 222–230

Parity Games

Tim A.C. Willemse and Maciej Gazda
Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Keywords

Automata; Computer-aided verification; Determinacy; Infinite duration games; Perfect information; Two-player games

Years and Authors of Summarized Original Work

1991; Emerson, Jutla
1991; Mostowski

Problem Definition

A parity game is an infinite duration game, played by players *odd* and *even*, denoted by $\Box$ and $\Diamond$, respectively on a directed, finite graph. Throughout this note, we let $\bigcirc$ denote an arbitrary player and we write $\bar{\bigcirc}$ for $\bigcirc$'s opponent; i.e. $\bar{\Diamond} = \Box$ and $\bar{\Box} = \Diamond$.

Definition 1 (Parity game) A parity game is a tuple $(V, E, \Omega, (V_\Diamond, V_\Box))$, where

- V is a set of vertices, partitioned in a set $V_\Diamond$ of vertices owned by player $\Diamond$, and a set of vertices $V_\Box$ owned by player $\Box$,
- $E \subseteq V \times V$ is a total edge relation,
- $\Omega{:}V \to \mathbb{N}$ is a priority function that assigns priorities to vertices.

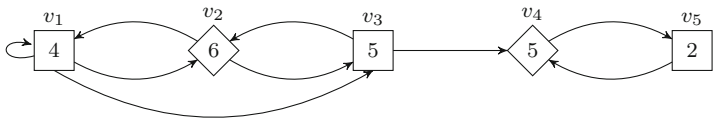
The graph (V, E) underlying a parity game is often referred to as the *arena*. Parity games are depicted as graphs with diamond-shaped vertices representing vertices owned by player $\Diamond$ and box-shaped vertices representing those owned by player $\Box$. The priorities associated with vertices are written inside vertices; see the game depicted in Fig. 1.

Imagine the following game, played on an arena. One starts by placing a token on a vertex. Then, players perpetually move this token according to a single simple rule: if the token is on some vertex $v \in V_\bigcirc$, player $\bigcirc$ gets to move the token to an adjacent vertex. The infinite sequence of vertices visited this way is referred to as a *play* and the *parity* of the *lowest* priority (associated with the vertices) that occurs infinitely often on the play defines its *winner*: player $\Diamond$ wins if and only if this priority is even.

It does not immediately follow from the above notion of winning that there is always a player that can win all plays that start in a given vertex. The most elementary of all problems concerning parity games is thus as follows.

Problem 1 Are parity games *determined*? That is, for a given parity game and some vertex in that game, is there always a way in which one of the players can play such that, regardless of how her opponent moves, the resulting plays are won by her?

The determinacy problem can be formalized by describing the choices players make when



Parity Games, Fig. 1 Parity games depicted as graphs with priorities associated with vertices, written inside vertices

they engage in a play; that is, we can formally capture their *strategies*. More specifically, a strategy for player ○ determines for a vertex v belonging to that player which of its adjacent vertices the token will be moved to next once the game play moves the token to v. Of course, such a decision can be based on the history of the play so far. We write vE to denote the set $\{w \in V \mid (v,w) \in E\}$. A strategy for player ○ is thus described adequately by a partial function $\sigma{:}V^*V_\bigcirc \to V$ satisfying that for all sequences $v_1 \dots v_n \in V^*$ for which σ is defined, we have $\sigma(v_1 \dots v_n) \in v_nE$. For a given strategy σ, a play p *conforms* to σ if for all finite prefixes q of p, whenever σ is defined for q, also $q\sigma(q)$ is a prefix of p. We say that a strategy σ for player ○ is *winning* from a vertex v if and only if ○ is the winner of every play that starts in v and that conforms to σ. The problem of determinacy can then be rephrased as follows: *for given vertex v in a game, is there always a player with a winning strategy from v?*

A vertex is won by some player if that player has a winning strategy from that vertex. The vertices won by player ◇ and player □ are denoted $W_\diamond$ and $W_\square$, respectively; determining these sets is typically referred to as *solving* a game. This immediately leads to the second fundamental question:

Problem 2 Can we compute $W_\diamond$ and $W_\square$?

Solving the above computational problem may not necessarily involve computing the winning strategies for both players. Nonetheless, the winning strategies themselves have their own merit, if only that they serve as certificates in proving that the winning sets for both players are indeed just that. This leads to the third problem of interest:

Problem 3 Can we, for all vertices that are won by one of the players, compute winning strategies for that player?

Key Results

The first claim, stated below, positively answers the determinacy problem for parity games.

Theorem 1 *Parity games are determined: for every vertex either player ◇ or player □ has a winning strategy from that vertex.*

Determinacy of parity games already follows from a general result due to Martin [9] who showed that Borel games (which subsume parity games) are determined. The proof of the latter result employs strategies that require infinite memory. A deep result by Emerson and Jutla [1], and found independently by Mostowski [11], states that parity games are in fact *memoryless determined*: if a player has a winning strategy from a vertex v, she also has a memoryless strategy that is winning for her; that is, player ○ always has a strategy σ for which $\sigma(pv) = \sigma(p'v)$ for all sequences pv and $p'v$ for which it is defined and which can thus be represented by a partial function $\sigma'{:}V_\bigcirc \to V$.

A simpler and constructive proof of memoryless determinacy for parity games with a finite arena was subsequently proposed by McNaughton [10] and extended to games with an infinite arena by Zielonka [15]. From the memoryless determinacy result, it follows that the problem of deciding whether player ◇ has a winning strategy from a given vertex is in NP: essentially one can guess a memoryless strategy for player ◇ and check in polynomial time whether it is winning; the latter can be done efficiently by showing the absence of odd cycles using e.g. [8]. In a

similar manner, one can prove that the problem is in coNP, making the decision problem one of those interesting problems that are in NP ∩ coNP. In fact, in 1998, Jurdziński [5] showed that the problem is in UP ∩ coUP. The complexity class UP is a subclass of NP and is defined to contain all problems that can be recognized by a nondeterministic polynomial time Turing machine that for every input has at most one accepting computation. From the memoryless determinacy, one also immediately obtains a positive answer to the second question.

Theorem 2 *The problem of deciding whether $v \in W_\diamond$ is in NP ∩ coNP, and the sets $W_\diamond$ and $W_\square$ can be computed.*

McNaughton and Zielonka's constructive proofs can be converted into a recursive algorithm for computing $W_\diamond$ and $W_\square$ for games with a finite arena. This algorithm, which we will introduce shortly, can be modified in a straightforward manner to also produce winning (memoryless) strategies for both players, thus also answering the third question.

Theorem 3 *Winning strategies for players $\diamond$ and $\square$ can be computed.*

Algorithms for Parity Games

While the problem of computing the winning sets of a parity game is decidable, its exact complexity is still open, but over the years, the upper bound has been improved. We therefore summarize the various algorithms that have been invented for solving parity games so far, starting with a brief exposition of the recursive algorithm which is, as we described above, also interesting for its theoretical consequences. For the remainder of this section, fix a parity game $G = (V, E, \Omega, (V_\diamond, V_\square))$ with n vertices, m edges, and d different priorities.

The recursive algorithm effectively decomposes a game into subgames and solves these. An essential ingredient in this decomposition is the notion of an $\bigcirc$-attractor into a set of vertices U, denoting $Attr^0_\bigcirc(U)$, which is the least set A, satisfying $A \supseteq U$ and

Algorithm 1: The recursive algorithm

```
function SOLVE(G)
    Input: parity game G = (V, E, Ω, (V_◇, V_□))
    Output: winning partition (W_◇, W_□)

    if V = ∅ then (W_◇, W_□) ← (∅, ∅)
    else
        m ← min{Ω(v) | v ∈ V}
        if m mod 2 = 0 then p ← ◇ else p ← □
    end if
        U ← {v ∈ V | Ω(v) = m}
        A ← Attr^0_p(U)
        (W'_◇, W'_□) ← SOLVE(G \ A)
        if W'_p̄ = ∅ then (W_p, W_p̄) ← (A ∪ W'_p, ∅)
        else
            B ← Attr^0_p̄(W'_p̄)
            (W'_◇, W'_□) ← SOLVE(G \ B)
            (W_p, W_p̄) ← (W'_p, W'_p̄ ∪ B)
        end if
    end if
    return (W_◇, W_□)
end function
```

- for all $v \in V_\bigcirc$, if $vE \cap A \neq \emptyset$, then also $v \in A$;
- for all $v \in V_{\bar{\bigcirc}}$, if $vE \subseteq A$, then also $v \in A$.

Intuitively, the $\bigcirc$-attractor into U contains U and exactly those vertices for which $\bigcirc$ can force play into U.

Writing $G \cap A$ for the parity game $(V \cap A, E \cap (A \times A), \Omega|_A, (V_\diamond \cap A, V_\square \cap A))$, where $\Omega|_A$ is the priority function Ω restricted to the vertices in A, and writing $G \setminus A$ for $G \cap (V \setminus A)$, the recursive algorithm is defined as in Algorithm 1.

The correctness of Algorithm 1 leans on the observation that lower priorities in the game dominate higher priorities and that revisiting these lower priorities is beneficial to the player with the "same parity" as this priority. The algorithm runs in polynomial space and its runtime complexity is $\mathcal{O}(m \cdot n^d)$.

The first improvement on this runtime that maintains a polynomial space complexity was achieved by an algorithm due to Jurdziński [6] in 2000. This algorithm, colloquially known as the *small progress measures* (SPM) algorithm, runs in time $\mathcal{O}(d \cdot m \cdot (\frac{n}{\lfloor \frac{d}{2} \rfloor})^{\lfloor \frac{d}{2} \rfloor})$. Jurdziński's algorithm builds on decorations of the game arena,

called *parity progress measures* and *game parity progress measures*; the former exist for a parity game with only odd-owned vertices if and only if the game only has even cycles (and thus is won by player $\diamond$); the latter extend parity progress measures to arbitrary games and are essentially witnesses for winning strategies for player $\diamond$. The SPM algorithm computes game parity progress measures using a fixpoint iteration, ensuring the measures are the least measures that decrease along a play with each bad odd priority that is encountered and only increase when reaching beneficial even priorities.

The next improvement came in 2006 and was based on a modification of the recursive algorithm, resulting in a subexponential algorithm with running time $n^{\mathcal{O}(\sqrt{n})}$; see [7]. It relies on a notion called a dominion for a player: a set of vertices D that is won by that player by staying *within* D and without allowing her opponent to leave D. The main idea behind the algorithm is that it identifies small dominions (of size at most $\sqrt{2n}$) using a dedicated algorithm and removes them from the game prior to the recursive calls. This algorithm, in turn, inspired Schewe [13] to improve on the runtime complexity for games with a small number of priorities. Rather than using a brute-force method for searching and eliminating dominions, Schewe's algorithm utilizes a modified SPM algorithm while executing the standard recursive algorithm. As a result, this reduces the complexity of solving parity games to $\mathcal{O}(m \cdot (\frac{\kappa \cdot n}{d})^{\gamma(d)})$, where κ is a small constant and $\gamma(d) \approx \frac{d}{3}$.

In parallel to the abovementioned solutions, a different family of algorithms has been developed. They are based on the notion of *strategy improvement*, which has been known in the game theory since the 1960s. The first algorithm of this kind designed specifically for parity games is due to Vöge and Jurdziński [14]. In this approach, one requires devising an order on strategies $\prec_{\bigcirc}$ that satisfies two conditions. Firstly, the maximal strategy w.r.t $\prec_{\bigcirc}$ is winning for $\bigcirc$ on $W_{\bigcirc}$. Secondly, there has to be an (efficiently computable) improvement procedure which, for every strategy σ that is not $\prec_{\bigcirc}$-maximal, computes a better strategy $\sigma' \succ_{\bigcirc} \sigma$. Strategy improvement algorithms start with a certain initial strategy and perform a sequence of improvement steps, until the maximal strategy is reached. While a single iteration (improvement step) is typically efficient, so far no policy guaranteeing a polynomial number of iterations has been found. In fact, Friedmann [3, 4] has proved that the key strategy improvement algorithms have worst-case exponential running time.

Applications

Parity games underlie a number of problems in theoretical computer science. For instance, they served as a vehicle for elegantly proving the complementation lemma for automata on infinite trees, a crucial lemma in Rabin's proof of the decidability of a particular second-order mathematical theory. Parity games are also used in word and emptiness problems for a variety of (alternating) automata [1]. Moreover, they are closely related to other two-player, infinite duration games with perfect information such as mean payoff games, which have, among others, applications in scheduling.

The practical significance of parity games stems from the fact that they have proved to be of great value in computer-aided software and hardware verification and synthesis. Of particular importance is the result that parity games are polynomial-time equivalent to model checking for the modal μ-calculus (see, for instance, [2]), a modal logic that expressively subsumes most of the popular temporal logics used in computer-aided verification. We present this transformation to illustrate the tight connection between game theory and logic.

Parity Games for Model Checking

Say we are given a structure (S, A, R), where A is a set of atomic actions, (S, R) is a directed graph in which S is a set of states, and $R \subseteq S \times A \times S$ is a (for simplicity) total edge-labeled transition relation. The structure (S, A, R) is often referred to as a *Labeled Transition System* (LTS), and it serves the purpose of modeling the behavior

of software or hardware. The modal μ-calculus allows for reasoning about such behaviors; the logic is defined through the following grammar:

$$f, g ::= true \mid false \mid X \mid f \wedge g \mid f \vee g \mid [a]f \mid \langle a \rangle f \mid \nu X.f \mid \mu X.f$$

where $a \in A$ and X is a propositional variable, taken from a sufficiently large set of variables $\mathcal{X}$. We write σ to denote either μ or ν. For simplicity, we assume that a propositional variable X in f is bound at most once (by some σ) and occurrences of X are all within the scope of its binder. Expressions in the logic are interpreted in the context of an LTS (S, A, R) and a mapping $e{:}\mathcal{X} \rightarrow 2^S$, typically referred to as an environment, assigning sets of states to propositional variables. The modal operators $\langle \_ \rangle \_$ and $[\_]\_$ allow for reasoning with the transition relation of an LTS; e.g.,, $\langle a \rangle f$ will hold in states that have *some* a-successor satisfying f, whereas $[a]f$ will hold in states for which *all* a-successors (if any) satisfy f. More formally, the meaning of a formula f is established by stating which states in the LTS satisfy it; this satisfaction relation, denoting $s, e \models f$, is defined inductively as follows:

$$\begin{array}{lll}
s, e \models true & & \\
s, e \not\models false & & \\
s, e \models X & \text{iff} & s \in e(X) \\
s, e \models f \wedge g & \text{iff} & s, e \models f \text{ and } s, e \models g \\
s, e \models f \vee g & \text{iff} & s, e \models f \text{ or } s, e \models g \\
s, e \models [a]f & \text{iff} & \text{for all } (s, a, t) \in R \;\; t, e \models f \text{ holds} \\
s, e \models \langle a \rangle f & \text{iff} & \text{exists } (s, a, t) \in R \text{ such that } t, e \models f \text{ holds} \\
s, e \models \nu X.f & \text{iff} & s \in \bigcup \{S' \subseteq S \mid S' \subseteq F(S')\} \\
s, e \models \mu X.f & \text{iff} & s \in \bigcap \{S' \subseteq S \mid F(S') \subseteq S'\}
\end{array}$$

where $F(T) = \{t \in S \mid t, e[X \mapsto T] \models f\}$ is a monotone operator in the complete lattice $(2^S, \subseteq)$, and where $e[X \mapsto T]$ is the environment e in which X is assigned set T. Perhaps due to its extreme expressive power, expressions in the modal μ-calculus are famously known for being hard to understand. Expressions using only one fixpoint are reasonably straightforward to interpret. For instance, the formula $\nu X.\langle a \rangle X$ holds for a state s with an infinite a-path: essentially, we are looking for the largest solution (its *fixpoint*) to the equation "$X = \langle a \rangle X$," or, more semantically, the largest set $T \subseteq S$ that can be assigned to X that satisfies the equation $T = \{s \in S \mid \exists t \in T : (s, a, t) \in R\}$. An expression such as $\mu Y.\langle a \rangle Y \vee \langle a \rangle true$ holds for a state s whenever there is a finite sequence of b-transitions leading to a state having an a-successor. Mixing fixpoints allows for expressing more complicated properties, unfortunately at the expense of readability, making formulas such as $\nu X.\mu Y.\langle a \rangle X \vee \langle b \rangle Y$ (expressing that, when it holds, there is an infinite sequence of a, b steps in which stretch of b-steps (if any) is of finite length), hard to understand.

The model checking problem is to decide, given some formula f, a state s in an LTS and some environment e, whether $s, e \models f$. This problem can be reduced to solving a parity game as follows: define a parity game $(V, E, \Omega, (V_\Diamond, V_\Box))$ in which $V = S \times \Phi(f)$, where $\Phi(f)$ is the set of all subformulas of f, and in which E, $V_\Diamond$, and $V_\Box$ are defined structurally as follows:

vertex	successor(s)	owner
$(s, true)$	$(s, true)$	$\Diamond$
$(s, false)$	$(s, false)$	$\Diamond$
(s, X) and $\sigma X.g \in \Phi(f)$	$(s, \sigma X.g)$	$\Diamond$
(s, X) and $\sigma X.g \notin \Phi(f)$	(s, X)	$\Diamond$
$(s, f \wedge g)$	(s, f) and (s, g)	$\Box$
$(s, f \vee g)$	(s, f) and (s, g)	$\Diamond$
$(s, [a]f)$	all (t, f) for $(s, a, t) \in R$	$\Box$
$(s, \langle a \rangle f)$	all (t, f) for $(s, a, t) \in R$	$\Diamond$
$(s, \nu X.f)$	(s, f)	$\Diamond$
$(s, \mu X.f)$	(s, f)	$\Diamond$

The priority function is assigned in such a way that it meets the following conditions:

- $\Omega((s, true)) = 0$ and $\Omega((s, false)) = 1$;
- if $\sigma X.g \notin \Phi(f)$, then $\Omega((s, X)) = 0$ if $s \in e(X)$ and $\Omega((s, X)) = 1$ otherwise;
- if $\sigma X.g \in \Phi(f)$, then
  - $\Omega((s, X))$ is *even* if $\sigma = \nu$ and *odd* otherwise, and
  - $\Omega((s, X)) \leq \Omega((t, Y))$ if Y depends on X (if X is *free* in g in $\sigma Y.g$);
- $\Omega((s, g))$ is maximal for other formulas $g \in \Phi(f)$.

There is an "optimal" assignment of priorities that does not assign values larger than the alternation depth of the μ-calculus formula f [12]. Intuitively, the alternation depth is a measure of the degree of semantic alternations between μ- and ν-operators. Using an optimal priority assignment yields better complexity bounds for the model checking problem. The theorem below establishes the connection between the model checking problem and the parity game solving problem.

Theorem 4 *$s, e \models f$ iff player $\Diamond$ wins (s, f) in the constructed parity game.*

On the one hand, through the above reduction, practical algorithmic progress in solving parity games directly impacts the performance and scalability of tooling for conducting the verification and synthesis. Abstracting from syntactic details, the more elementary parity games setting, on the other hand, permits studying the true complexity of model checking and, at the same time, provides a better understanding of the dynamics of the modal μ-calculus.

Open Problems

Parity games are among the few problems that are in NP $\cap$ co-NP for which no polynomial time algorithm has been found. The key open problem is thus whether there is a polynomial time algorithm for solving parity games.

Problem 4 Can $W_\Diamond$ and $W_\Box$ be computed in polynomial time?

Cross-References

▸ Symbolic Model Checking

Recommended Reading

1. Emerson E, Jutla C (1991) Tree automata, mu-calculus and determinacy. In: FOCS'91. IEEE Computer Society, Washington, DC, pp 368–377. 10.1109/SFCS.1991.185392
2. Emerson E, Jutla C, Sistla A (2001) On model checking for the μ-calculus and its fragments. Theor Comput Sci 258(1–2):491–522. 10.1016/S0304-3975(00)00034-7
3. Friedmann O (2011) An exponential lower bound for the latest deterministic strategy iteration algorithms. Log Methods Comput Sci 7(3)
4. Friedmann O (2013) A superpolynomial lower bound for strategy iteration based on snare memorization. Discret Appl Math 161(10–11):1317–1337
5. Jurdziński M (1998) Deciding the winner in parity games is in UP $\cap$ co-UP. IPL 68(3):119–124. 10.1016/S0020-0190(98)00150-1
6. Jurdziński M (2000) Small progress measures for solving parity games. In: STACS'00. LNCS, vol 1770. Springer, pp 290–301. 10.1007/3-540-46541-3_24
7. Jurdziński M, Paterson M, Zwick U (2006) A deterministic subexponential algorithm for solving parity games. In: SODA'06. ACM/SIAM, pp 117–123. 10.1145/1109557.1109571
8. King V, Kupferman O, Vardi MY (2001) On the complexity of parity word automata. In: FOSSACS. LNCS, vol 2030. Springer, pp 276–286. 10.1007/3-540-45315-6_18
9. Martin D (1975) Borel determinacy. Ann Math 102:363–371. 10.2307/1971035
10. McNaughton R (1993) Infinite games played on finite graphs. APAL 65(2):149–184. 10.1016/0168-0072(93)90036-D
11. Mostowski A (1991) Hierarchies of weak automata and weak monadic formulas. Theor Comput Sci 83(2):323–335. 10.1016/0304-3975(91)90283-8
12. Niwinski D (1997) Fixed point characterization of infinite behavior of finite-state systems. Theor Comput Sci 189(1–2):1–69. 10.1016/S0304-3975(97)00039-X
13. Schewe S (2007) Solving parity games in big steps. In: FSTTCS'07. LNCS, vol 4855. Springer, pp 449–460. 10.1007/978-3-540-77050-3
14. Vöge J, Jurdzinski M (2000) A discrete strategy improvement algorithm for solving parity games. In: Emerson EA, Sistla AP (eds) CAV. LNCS, vol 1855. Springer, Heidelberg, pp 202–215
15. Zielonka W (1998) Infinite games on finitely coloured graphs with applications to automata on infinite trees. TCS 200(1–2):135–183. 10.1016/S0304-3975(98)00009-7

Pattern Matching on Compressed Text

Masayuki Takeda[1] and Ayumi Shinohara[2]
[1]Department of Informatics, Kyushu University, Fukuoka, Japan
[2]Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Keywords

Collage system; Compressed pattern matching; Straight-line program

Years and Authors of Summarized Original Work

2003; Kida, et al.
2011; Gawrychowski
2013; Gawrychowski

Problem Definition

Let $\mathbf{c}$ be a given compression function that maps strings A to their compressed representations $\mathbf{c}(A)$. The problem of *compressed pattern matching (CPM)* is defined as follows:

Problem 1 (Compressed Pattern Matching) Given a pattern string P and a compressed text string $\mathbf{c}(T)$, determine whether there is an occurrence of P in T, without decompressing T.

A CPM algorithm is said to be *optimal* if it runs in $O(|P| + |\mathbf{c}(T)|)$ time. The time/space complexity of the CPM problem can be a new criterion to evaluate compression schemes in addition to the traditional ones: the compression ratio and the time/space complexity of compression/decompression.

The CPM problem was first defined in the work of Amir and Benson [1], and many studies have been made over different compression formats. Kida et al. [9] introduced a useful CPM-oriented abstraction of compression formats, named *collage systems*. Outputs of various compression algorithms – not only dictionary-based compression algorithms but also grammar-based compression algorithms – can be regarded as collage systems, and hence algorithmic research working on collage systems is of great significance. They presented in the same paper a general Knuth-Morris-Pratt (KMP) algorithm on collage systems. A general Boyer-Moore (BM) algorithm on collage systems was also designed by almost the same authors [17].

Collage Systems

Let Σ be a fixed finite alphabet. A *collage system* on Σ is a pair $\langle \mathcal{D}, \mathcal{S} \rangle$ defined as follows:

- $\mathcal{D}$ is a sequence of assignments $X_1 = expr_1;\ X_2 = expr_2;\ \cdots;\ X_n = expr_n$, where, for each $k = 1, \ldots, n$, X_k is a variable and $expr_k$ is any of the form:
- $\mathcal{S}$ is a sequence $X_{i_1} \cdots X_{i_\ell}$ of variables defined in $\mathcal{D}$.

By *the j length prefix (resp. suffix) truncation*, we mean an operation on strings which takes a string w and returns the string obtained from w by removing its prefix (resp. suffix) of length j. The variables X_k represent the strings $val(X_k)$ obtained by evaluating their expressions. A collage system $\langle \mathcal{D}, \mathcal{S} \rangle$ represents the string obtained by concatenating the strings $val(X_{i_1}), \ldots, val(X_{i_\ell})$ represented by variables $X_{i_1}, \ldots, X_{i_\ell}$ of $\mathcal{S}$.

The *size* of $\mathcal{D}$ is the number n of assignments and denoted by $|\mathcal{D}|$. The *height* of $\mathcal{D}$, denoted by $height(\mathcal{D})$, is defined to be the longest path length of the dependency graph of $\mathcal{D}$, namely, a directed acyclic graph such that (1) the vertices are the variables in $\mathcal{D}$ and (2) a directed edge from X_k to X_i exists if and only if $\mathcal{D}$ contains a non-primitive assignment $X_k = expr_k$ such that X_i appears in $expr_k$. The *length* of $\mathcal{S}$ is the number ℓ of variables in $\mathcal{S}$ and denoted by $|\mathcal{S}|$. The *size* of collage system $\langle \mathcal{D}, \mathcal{S} \rangle$ is defined to be $|\mathcal{D}| + |\mathcal{S}|$.

It should be noted that any collage system can be converted into an equivalent one with $|\mathcal{S}| = 1$, by adding a series of assignments with concatenation operations into $\mathcal{D}$. This may imply $\mathcal{S}$ is unnecessary. However, a variety of compression schemes can be captured naturally by

separating $\mathcal{D}$ (defining *phrases*) from $\mathcal{S}$ (giving a factorization of text T into phrases). How to express outputs of existing compression algorithms is found in [9].

A collage system $\langle \mathcal{D}, \mathcal{S} \rangle$ is said to be *truncation-free* if $\mathcal{D}$ contains no truncation operation, *repetition-free* if $\mathcal{D}$ contains no repetition operation, and *regular* if it is truncation- and repetition-free. A regular collage system $\langle \mathcal{D}, \mathcal{S} \rangle$ is *simple* if $|val(Y)| = 1$ or $|val(Z)| = 1$ for every assignment $X = YZ$ of $\mathcal{D}$.

Outputs of grammar-based compression algorithms such as Re-Pair, Sequitur, and Byte Pair Encoding (BPE) fall into the class of regular collage systems, and outputs of LZ78/LZW fall into the class of simple collage systems. LZ77 factorization is an abstraction of LZ77 and its variants, which has two variations depending upon whether self-referencing is allowed. The LZ77 factorization Z of T with (resp. without) self-referencing can be transformed into a collage system (resp. a repetition-free collage system) of size $O(|Z| \cdot \log |Z|)$ generating T (see [5]).

It should be mentioned that the so-called straight-line programs (SLPs) are the regular collage systems with $|\mathcal{S}| = 1$.

Key Results

Theoretical Aspect

Amir et al. [2] presented two solutions to CPM for LZW with time complexities $O(|\mathbf{c}(T)| \log |P| + |P|)$ and $O(|\mathbf{c}(T)| + |P|^2)$, respectively. The latter was generalized by Kida et al. [9] via the unified framework of collage systems.

Theorem 1 (Kida et al. [9]) *CPM for collage systems can be solved in* $O\big((|\mathcal{D}| + |\mathcal{S}|) \cdot height(\mathcal{D}) + |P|^2\big)$ *time using* $O(|\mathcal{D}| + |P|^2)$ *space. The factor* $height(\mathcal{D})$ *is dropped for truncation-free collage systems.*

We briefly sketch the algorithm of [9]. It is originally intended to solve the *all-occurrence* version of the CPM problem and reports all locations of T at which P occurs with additional time linearly proportional to the number of pattern occurrences. The algorithm has two stages: First, it preprocesses $\mathcal{D}$ and P, and second it processes the variables of $\mathcal{S}$. In the second stage, it simulates the move of KMP automaton running on uncompressed text, by using two functions *Jump* and *Output*, both take as input a state q and a variable X. The former is used to substitute just one state transition for the consecutive state transitions of the KMP automaton for the string $val(X)$ for each variable X of $\mathcal{S}$, and the latter is used to report all pattern occurrences found during the state transitions. Let δ be the state-transition function of the KMP automaton. Then $Jump(q, X) = \delta(q, val(X))$, and $Output(q, X)$ is the set of lengths $|w|$ of nonempty prefixes w of $val(X)$ such that $\delta(q, w)$ is the final state. A naive two-dimensional array implementation of the two functions requires $\Omega(|\mathcal{D}| \cdot |P|)$ space, and the size of $Output(q, X)$ can be exponential in $|\mathcal{D}|$. The data structures of [9] use only $O(|\mathcal{D}| + |P|^2)$ space, are built in $O(|\mathcal{D}| \cdot height(\mathcal{D}) + |P|^2)$ time, and enable us to compute $Jump(q, X)$ in $O(1)$ time and enumerate the set $Output(q, X)$ in $O(height(\mathcal{D}) + \ell)$ time where $\ell = |Output(q, X)|$. The factor $height(\mathcal{D})$ is dropped for truncation-free collage systems.

By replacing $|\mathcal{D}| + |\mathcal{S}|$ with $|\mathbf{c}(T)|$, the above theorem means that the existence version of CPM can be solved in $O(|\mathbf{c}(T)| + |P|^2)$ time for *any* compression formats that fall into the class of truncation-free collage systems. $|P|^2$ is acceptable since it is often smaller than $|\mathbf{c}(T)|$ in

a	for $a \in \Sigma \cup \{\varepsilon\}$,	(*primitive assignment*)
$X_i X_j$	for $i, j < k$,	(*concatenation*)
${}^{[j]}X_i$	for $i < k$ and a positive integer j,	(j *length prefix truncation*)
$X_i^{[j]}$	for $i < k$ and a positive integer j,	(j *length suffix truncation*)
$(X_i)^j$	for $i < k$ and a positive integer j.	(j *times repetition*)

practice. But removing the quadratic dependency on $|P|$ is of great interest in theory.

Consider a maximal variable sequence $\mathcal{S}[i..j]$ of $\mathcal{S}[1..\ell]$ such that the string $val(\mathcal{S}[k])$ is a factor of pattern P for any $k \in [i..j]$. Take the longest suffix h of $val(\mathcal{S}[i-1])$ and the longest prefix t of $val(\mathcal{S}[j+1])$ to obtain the sequence $h, val(\mathcal{S}[i]), \ldots, val(\mathcal{S}[j]), t$ of pattern factors. Any pattern occurrence must appear in such a pattern factor sequence, except for the case that some variable $X = \mathcal{S}[k]$ contains a pattern occurrence in its string $val(X)$. The CPM task can thus be reduced into a number of instances of pattern matching in a sequence of pattern factors. It should be noted that the task of pattern matching in a pattern factor sequence depends only on P, not depending on T nor its compression format. Gawrychowski [7] described an elaborate technique to perform this task in linear time. On the other hand, in the reduction task, we are faced with the need to solve the so-called factor concatenation problem [9]:

> Preprocess P to build a data structure that returns in constant time the vertex representing the factor xy, for any two (explicit or implicit) vertices of suffix tree of P representing factors x, y of P.

An $O(|P|^2)$ time preprocessing was presented in [9]. For LZW or, more generally, simple collage systems, it is rather straightforward to see that if the alphabet is of constant size, the preprocessing requires only $O(|P|)$ time since either x or y is of length 1, and the reduction task thus takes $O(|\mathcal{D}|+|\mathcal{S}|+|P|)$ time. CPM for simple collage systems can therefore be solved in optimal linear time for a constant alphabet. Gawrychowski [7] further described how to keep it linear even in the case of integer alphabet for LZW.

Theorem 2 (Gawrychowski [7]) *CPM for LZW can be solved in optimal linear time even for a polynomial size integer alphabet, assuming the word RAM model.*

For LZ77, one possible solution is to convert the input LZ77 factorization into a truncation-free collage system and then apply the CPM algorithm of [9]. We can convert an LZ77 factorization of T into a truncation-free collage system with an increase in size by a factor of $O(\log \frac{|T|}{|\mathbf{c}(T)|})$ in time linear to the output size (see [4]). The resulting algorithm thus has time complexity of $O(|\mathbf{c}(T)| \log \frac{|T|}{|\mathbf{c}(T)|} + |P|^2)$. Gawrychowski in [6] successfully removed the quadratic dependency on $|P|$ again.

Theorem 3 (Gawrychowski [6]) *CPM for LZ77 can be solved in* $O(|\mathbf{c}(T)| \log \frac{|T|}{|\mathbf{c}(T)|} + |P|)$ *time.*

Table 1 summarizes the best known solutions to CPM for several compression formats.

Practical Aspect

From a practical viewpoint, we have two goals. One is to perform the CPM task in less time compared with a decompression followed by an ordinary search (Goal 1), and the other is to perform it in less time compared with an ordinary search over uncompressed text (Goal 2). An optimal CPM algorithm theoretically achieves the two goals if $|\mathbf{c}(T)| = o(|T|)$. However, we often observe $|\mathbf{c}(T)| = \Theta(|T|)$ in practice, and hence reducing the constant factors hidden behind the O-notation of time complexity of CPM algorithms plays a crucial role in achieving the two goals, especially for Goal 2. For example, code words are limited to multiples of 8 bits to avoid bit manipulation.

Kida et al. [8] reported the first experimental results in this area, achieving Goal 1 for LZW. Navarro and Tarhio [14] presented BM-type algorithms for LZ78/LZW compression schemes and showed they are twice as fast as a decompression followed by a search using the best algorithms.

Data compression can be regarded as a preprocessing that allows a fast search in the context of Goal 2. An appropriate compression format would be chosen for this purpose. We note that in general, some occurrences of the encoded pattern can be false matches, and/or the pattern possibly occurs in several different forms within the encoded text. There are two lines of research work addressing Goal 2. One is to put a restriction on the compression scheme so that every pattern occurrence can be identified simply as a substring of the encoded text that is identical to the encoded

Pattern Matching on Compressed Text, Table 1 Best known solutions to CPM for several compression formats

Compression formats	Time complexity	Work
Run-length	$O(\|\mathfrak{c}(T)\| + \|P\|)$	Trivial
LZW	$O(\|\mathfrak{c}(T)\| + \|P\|)$	[7]
LZ77	$O(\|\mathfrak{c}(T)\| \log \frac{\|T\|}{\|\mathfrak{c}(T)\|} + \|P\|)$	[6]
Simple collage systems	$O((\|\mathcal{D}\| + \|\mathcal{S}\|) + \|P\|)$	[7]
Truncation-free collage systems	$O((\|\mathcal{D}\| + \|\mathcal{S}\|) + \|P\|^2)$	[9]
Collage systems	$O((\|\mathcal{D}\| + \|\mathcal{S}\|) \cdot height(\mathcal{D}) + \|P\|^2)$	[9]

pattern. The advantage is that *any* favored pattern matching algorithm can be used to search the encoded text for the encoded pattern. The works of Manber [10] and Rautio et al. [15] are along this line. The latter is based on a combination of the so-called stopper encoding and the Boyer-Moore-Horspool (BMH) algorithm and is regarded as the fastest combination that achieves Goal 2. The drawback of this line is, however, that the restriction considerably sacrifices the compression ratio (e.g., 60–70 % for typical English texts). In the case of natural language texts written in western languages such as English (having explicit word boundaries), there are some compression formats that enable us to achieve Goal 2 by using a modification of byte-oriented Huffman coding on words (see, e.g., [3]).

The other line is to suppress a false detection or detection omission by an algorithmic device, without putting such a restriction on the compression scheme. The work of Miyazaki et al. [13] for Huffman encoding and the works of Shibata et al. [16, 17] for BPE are along this line. While all of the works [10, 13, 15–17] mentioned here achieve Goal 2, the compression ratios are poor: BPE is the best among them. A BPE compressed text is a regular collage system with limitation $|\mathcal{D}| \leq 256$. Matsumoto [12] extended BPE to get a higher compression ratio by easing the limitation and using the byte-oriented Huffman coding for representing the variables occuring in $\mathcal{S}$. Their CPM algorithm runs fast to achieve Goal 2, but memory requirement increases as $|\mathcal{D}|$ grows. Maruyama [11] introduced a new compression scheme, called the *context-sensitive grammar transform*, of which compression ratio is a match for gzip and Re-Pair. The search speed of their CPM algorithm is almost twice faster than the KMP-type CPM algorithm of [16] on BPE and faster than [15] for short patterns.

Cross-References

- Grammar Compression
- Huffman Coding
- Lempel-Ziv Compression
- String Matching
- Suffix Trees and Arrays

Recommended Reading

1. Amir A, Benson G (1992) Efficient two-dimensional compressed matching. In: Proceedings of the data compression conference (DCC'92), Snowbird, p 279
2. Amir A, Benson G, Farach M (1996) Let sleeping files lie: pattern matching in Z-compressed files. J Comput Syst Sci 52(2):299–307
3. Brisaboa N, Fariña A, Navarro G, Paramá J (2007) Lightweight natural language text compression. Inf Retr 10:1–33
4. Charikar M, Lehman E, Liu D, Panigrahy R, Prabhakaran M, Rasala A, Sahai A, Shelat A (2002) Approximating the smallest grammar: Kolmogorov complexity in natural models. In: STOC, Montréal, pp 792–801
5. Gąsieniec L, Karpinski M, Plandowski W, Rytter W (1996) Efficient algorithms for Lempel-Ziv encoding. In: Proceedings of the 5th Scandinavian workshop on algorithm theory (SWAT'96), Reykjavík. Lecture notes in computer science, vol 1097, pp 392–403
6. Gawrychowski P (2011) Pattern matching in Lempel-Ziv compressed strings: fast, simple, and deterministic. In: ESA, Saarbrücken, pp 421–432
7. Gawrychowski P (2013) Optimal pattern matching in LZW compressed strings. ACM Trans Algorithms 9(3):25
8. Kida T, Takeda M, Shinohara A, Miyazaki M, Arikawa S (2000) Multiple pattern matching in LZW compressed text. J Discret Algorithms 1(1):133–158

9. Kida T, Matsumoto T, Shibata Y, Takeda M, Shinohara A, Arikawa S (2003) Collage systems: a unifying framework for compressed pattern matching. Theor Comput Sci 298(1):253–272
10. Manber U (1997) A text compression scheme that allows fast searching directly in the compressed file. ACM Trans Inf Syst 15(2):124–136
11. Maruyama S, Tanaka Y, Sakamoto H, Takeda M (2010) Context-sensitive grammar transform: compression and pattern matching. IEICE Trans Inf Syst 93-D(2):219–226
12. Matsumoto T, Hagio K, Takeda M (2009) A runtime efficient implementation of compressed pattern matching automata. Int J Found Comput Sci 20(4):717–733
13. Miyazaki M, Fukamachi S, Takeda M, Shinohara T (1998) Speeding up the pattern matching machine for compressed texts. Trans Inf Process Soc Jpn 39(9):2638–2648
14. Navarro G, Tarhio J (2005) LZgrep: A Boyer-Moore string matching tool for Ziv-Lempel compressed text. Softw Pract Exp 35(12):1107–1130
15. Rautio J, Tanninen J, Tarhio J (2002) String matching with stopper encoding and code splitting. In: Proceedings of the 13th annual symposium on combinatorial pattern matching (CPM'02), Fukuoka, pp 42–52
16. Shibata Y, Kida T, Fukamachi S, Takeda M, Shinohara A, Shinohara T, Arikawa S (2000) Speeding up pattern matching by text compression. In: Proceedings of the 4th Italian conference on algorithms and complexity (CIAC'00), Rome. Lecture notes in computer science, vol 1767. Springer, pp 306–315
17. Shibata Y, Matsumoto T, Takeda M, Shinohara A, Arikawa S (2000) A Boyer-Moore type algorithm for compressed pattern matching. In: Proceedings of the 11th annual symposium on combinatorial pattern matching (CPM'00), Montréal. Lecture notes in computer science, vol 1848. Springer, pp 181–194

Patterned Self-Assembly Tile Set Synthesis

Shinnosuke Seki
Department of Computer Science, Helsinki Institute for Information Technology (HIIT), Aalto University, Aalto, Finland

Keywords

Computer-assisted proof; Heuristic algorithms; Minimization; NP-hardness; Pattern self-assembly; Rectilinear tile assembly system (RTAS)

Years and Authors of Summarized Original Work

2008; Ma, Lombardi
2013; Seki
2014; Göös, Lampiäinen, Czeizler, Orponen
2014; Kari, Kopecki, Meunier, Patitz, Seki

Problem Definition

A *tile type* is a colored unit square each of whose four sides is provided with a glue. An *assembly* is a partial function from $\mathbb{Z}^2$ (2D-grid) to a tile type set T. A (rectangular) pattern P (of width w and height h) is a function from the rectangular domain $[w] \times [h]$ to a set of colors, where $[m] = \{1, \ldots, m\}$ for $m \in \mathbb{N}$. If at most k colors appear on P, we say P is *k-colored*. Tiles being colored, an assembly of domain $[w] \times [h]$ induces a unique pattern of width w and height h.

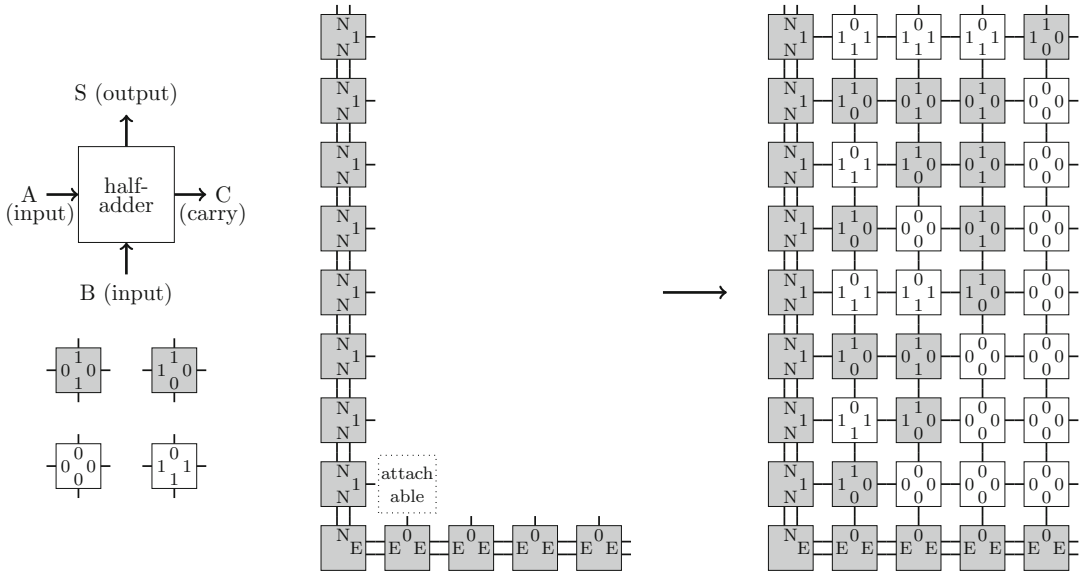
The rectilinear tile assembly system (RTAS) is a variant of Winfree's aTAM system [10]. Figure 1 illustrates how an RTAS self-assembles the binary counter pattern. An RTAS is a pair of a finite set of tile types and an L-shape seed, which is an assembly of domain $\{(0,0)\} \cup [w] \times \{0\} \cup \{0\} \times [h]$. Starting from the L-shape seed, it tiles the plain according to the following rule:

RTAS's tiling rule: A tile can attach at a position (x, y) if its west glue matches the east glue of the tile at $(x-1, y)$ and its south glue matches the north glue label of the tile at $(x, y-1)$.

A tile hence finds the sole attachable position (1, 1) on the L-shape seed. The attachment of tile there makes the positions (2, 1) and (1, 2) attachable. Tiles attach in this manner one after another and an assembly grows rectilinearly.

An RTAS is *directed* (a.k.a. *deterministic*) if no two tile types share the west and south glues, as the one in Fig. 1. A directed RTAS admits a unique assembly $\mathcal{A}_\square$ to which no tile can attach any more. Then we say that it *uniquely self-assembles the pattern of* $\mathcal{A}_\square$.

Patterned self-assembly tile set synthesis (PATS), proposed by Ma and Lombardi [7], aims



Patterned Self-Assembly Tile Set Synthesis, Fig. 1 (*Left*) Four tile types implement together the half-adder with two inputs A, B from the west and south, the output S to the north, and the carryout C to the east (*Right*) Copies of the "half-adder" tile types turn the L-shape seed into the binary counter pattern

at minimizing the RTAS that self-assembles a given pattern, where an RTAS is measured by the cardinality of its tile type set. Since the minimum RTAS is known to be directed [2], this problem is formulated as follows:

Patterned self-assembly tile set synthesis (PATS) [7]

GIVEN: A (rectangular) pattern P

FIND: A minimum directed RTAS that uniquely self-assembles P

For $k \geq 1$, the k-colored PATS (k-PATS) is a practical variant of PATS which takes only the k-colored patterns as input.

Key Results

PATS and k-PATS have been studied mainly in two research directions so far: its computational complexity and algorithms.

The **NP**-hardness of 2-PATS was claimed in [8], but what was proved **NP**-hard there was something different. Czeizler and Popa proved that PATS is **NP**-hard [1] (its concise proof is in [5]) by a polynomial-time reduction of 3SAT to the decision variant of PATS: given a pattern P and $n \in \mathbb{N}$, decide whether P can be uniquely self-assembled by a directed RTAS with n tile types. Variables and clauses of a given 3SAT instance ϕ are color-coded so that the number of colors in the reduced pattern is in proportion to the size of ϕ.

Potential of geometry, or more precisely, configuration of colors, as a medium of encoding a 3SAT encoding a 3SAT instance ϕ was suggested by Seki [9]. In the reduction, a set T_{eval} of 84 tile types is designed as a 3SAT-verifier, i.e., using tiles in the set, a directed RTAS evaluates ϕ to be true and assembles a pattern $P(\phi)$, starting from a seed encoding ϕ and some satisfying assignment. A subpattern GADGET of $P(\phi)$ endows $P(\phi)$ with the property that in order for a directed RTAS with a set T of at most 84 tile types to assemble $P(\phi)$, T must be isomorphic to T_{eval}. The pattern $P(\phi)$ is 60-colored. Hence, ϕ is satisfiable if and only if $(P(\phi), 84)$ is a yes-instance of 60-PATS. Johnsen, Kao, and Seki have refined the original design so that the number of colors decreased to 29 [3] and further to 11 [4].

In these proofs, quite a few colors are devoted just to make the property of GADGET manually

checkable. Giving up the manual checkability of the GADGET property has yielded a computer-assisted proof of **NP**-hardness of 2-PATS by Kari et al. [6]. The proof was verified in two different environments. Note that in this proof, everything but the GADGET property is manually checkable.

The **NP**-hardness of 2-PATS makes it essentially indispensable for exact PATS-solvers to search for the exponential number of solution candidates. Göös et al. designed PATS-solvers: an exhaustive partition-search branch-and-bound (PS-BB) algorithm, its heuristic modification (PS-H), and an ASP-solver-based algorithm [2]. PS-BB is an exact algorithm for PATS, running in practical time just for small patterns, say 7×7, while PS-H works even for larger patterns in exchange for the loss of guarantee on the minimality of its output.

Let P be an input pattern of width w and height h. The colors on P induce a partition $\pi(c)$ of the P's domain $[w] \times [h]$. In principle, among all possible partitions, PS-BB searches for the one $\pi_{\min}$ of least cardinality satisfying:

- $\pi(c)$ is coarser than $\pi_{\min}$ in the sense that $\forall q \in \pi_{\min}, \exists p \in \pi(c), q \subseteq p$.
- One can associate each class p of $\pi_{\min}$ with a quadruple of glues such that a directed RTAS with the associated tile type set uniquely self-assembles P.

Note that the finest partition $\pi_{\max} = \{\{(x, y)\} \mid (x, y) \in [w] \times [h]\}$ satisfies these conditions (associated tile types "hardcode their position" in glues). The coarser-finer relation yields a tree of partitions whose root is $\pi_{\max}$. This is the search tree of PS-BB (and PS-H).

PS-BB employs branch-pruning by bounding function to save computational resources. PS-H more greedily optimizes the order in which the coarsenings of a partition are explored by preferring some search paths to the others. Random choice of the one among the preferred lets PS-H perform differently at each run. PS-H$_n$ is a variant of PS-H which runs multiple independent searches in parallel for efficiency.

Open Problems

The lack of guarantee on the minimality of tile type sets output by the heuristic algorithms or on their running time motivates the design of a polynomial-time approximation algorithm for PATS. The ratio $14/13 \approx 1.077$ is known to be unachievable in polynomial-time, unless **P** = **NP** [6].

A manually checkable proof for the **NP**-hardness of 2-PATS is of not practical but theoretical interest.

URLs to Code and Data Sets

The computer program for the computer-assisted proof is available online:
http://self-assembly.net/wiki/index.php?title=2PATS-tileset-search.

Cross-References

- ▸ Circuit Placement
- ▸ Combinatorial Optimization and Verification in Self-Assembly
- ▸ Experimental Implementation of Tile Assembly

Recommended Reading

1. Czeizler E, Popa A (2012) Synthesizing minimal tile sets for complex patterns in the framework of patterned DNA self-assembly. In: Proceedings of the DNA 18, Aarhus. LNCS, vol 7433. Springer, pp 58–72
2. Göös M, Lampiäinen T, Czeizler E, Orponen P (2014) Search methods for tile sets in patterned DNA self-assembly. J Comput Syst Sci 80:297–319
3. Johnsen A, Kao MY, Seki S (2013) Computing minimum tile sets to self-assemble color patterns. In: Proceedings of the ISAAC 2013, Hong Kong. LNCS, vol 8283. Springer, pp 699–710
4. Johnsen A, Kao MY, Seki S (2014, Submitted) A manually-checkable proof for the NP-hardness of 11-color pattern self-assembly tile set synthesis
5. Kari L, Kopecki S, Seki S (2013) 3-color bounded patterned self-assembly. In: Proceedings of the DNA 19, Tempe. LNCS, vol 8141. Springer, pp 105–117

6. Kari L, Kopecki S, Étienne Meunier P, Patitz MJ, Seki S (2014) Binary pattern tile set synthesis is NP-hard. arXiv: 1404.0967
7. Ma X, Lombardi F (2008) Synthesis of tile sets for DNA self-assembly. IEEE Trans Comput-Aided Des Integr Circuits Syst 27(5):963–967
8. Ma X, Lombardi F (2009) On the computational complexity of tile set synthesis for DNA self-assembly. IEEE Trans Circuits Syst II 56(1):31–35
9. Seki S (2013) Combinatorial optimization in pattern assembly (extended abstract). In: Proceedings of the UCNC 2013, Milan. LNCS, vol 7956. Springer, pp 220–231
10. Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology

Peptide De Novo Sequencing with MS/MS

Bin Ma
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada
Department of Computer Science, University of Western Ontario, London, ON, Canada

Keywords

De novo sequencing; Peptide sequencing

Years and Authors of Summarized Original Work

2005; Ma, Zhang, Liang

Problem Definition

De novo sequencing arises from the identification of peptides by using tandem mass spectrometry (MS/MS). A peptide is a sequence of amino acids in biochemistry and can be regarded as a string over a finite alphabet from a computer scientist's point of view. Each letter in the alphabet represents a different kind of amino acid and is associated with a mass value. In the biochemical experiment, a tandem mass spectrometer is utilized to fragment many copies of the peptide into pieces and to measure the mass values (in fact, the mass to charge ratios) of the fragments simultaneously. This gives a tandem mass spectrum. Since different peptides normally produce different spectra, it is possible, and now a common practice, to deduce the amino acid sequence of the peptide from its spectrum. Often this deduction involves the searching in a database for a peptide that can possibly produce the spectrum. But in many cases such a database does not exist or is not complete, and the calculation has to be done without looking for a database. The latter approach is called de novo sequencing.

A general form of de novo sequencing problems is described in [2]. First, a score function $f(P, S)$ is defined to evaluate the pairing of a peptide P and a spectrum S. Then the de novo sequencing problem seeks for a peptide P such that $f(P, S)$ is maximized for a given spectrum S.

When the peptide is fragmented in the tandem mass spectrometer, many types of fragments can be generated. The most common fragments are the so called b-ions and y-ions. b-ions correspond to the prefixes of the peptide sequence, and y-ions the suffixes. Readers are referred to [8] for the biochemical details of the MS/MS experiments and the possible types of fragment ions. For clarity, in what follows only b-ions and y-ions are considered, and the de novo sequencing problem will be formulated as a pure computational problem.

A spectrum $S = \{(x_i, h_i)\}$ is a set of peaks, each has a mass value x_i and an intensity value h_i. A peptide $P = a_1 a_2 \ldots a_n$ is a string over a finite alphabet Σ. Each $a \in \Sigma$ is associated with a positive mass value $m(a)$. For any string $t = t_1 t_2 \ldots t_k$, denote $m(t) \sum_{i=1}^{k} m(t_i)$. The mass of a length-$i$ prefix (b-ion) of P is defined as $b_i = c_b + m(a_1 a_2 \ldots a_i)$. The mass of a length-$i$ suffix (y-ion) of P is defined as $y_i = c_y + m(a_{k-i+1} \ldots a_{k-1} a_k)$. Here c_b and c_y are two constants related to the nature of the MS/MS experiments. If the mass unit used for measuring each amino acid is dalton, then $c_b = 1$ and $c_y = 19$.

Let δ be a mass error tolerance that is associated with the mass spectrometer. For mass value m, the peaks *matched* by m is defined as $D(m) = \{(x_i, h_i) \in S | \ |x_i - m| \leq \delta\}$. The general idea of de novo sequencing is to maximize the number and intensities of the peaks matched by all b and y ions. Normally, δ is far less than the minimum mass of an amino acid. Therefore, for different i and j, $D(b_i) \cap D(b_j) = \emptyset$ and $D(y_i) \cap D(y_j) = \emptyset$. However, $D(b_i)$ and $D(y_j)$ may share common peaks. So, if not defined carefully, a peak may be counted twice in the score function. There are two different definitions of de novo sequencing problem, corresponding to two different ways of handling this situation.

Definition 1 (Anti-symmetric de novo sequencing)

Instance: A spectrum S, a mass value M, and an error tolerance δ.
Solution: A peptide P such that $m(P) = M$, and $D(b_i) \cap D(y_j) = \emptyset$ for any i, j.
Objective: Maximize $\sum_{k=1}^{n} \sum_{(x_i,h_i) \in D(b_k) \cup D(y_k)} h_i$.

This definition discards the peptides that give a pair of b_i and y_j with similar mass values, because this happens rather infrequently in practice. Another definition allows the peptides to have pairs of b_i and y_j with similar mass values. However, when a peak is matched by multiple ions, it is counted only once. More precisely, define the matched peaks by P as

$$D(P) = \bigcup_{i=1}^{n} (D(b_i) \cup D(y_i)).$$

Definition 2 (De novo sequencing)

Instance: A spectrum S, a mass value M, and an error tolerance δ.
Solution: A peptide P such that $m(P) = M$.

$$f(P, S) = \sum_{(x_i,h_i) \in D(P)} h_i.$$

Objective: Maximize

Key Results

Anti-symmetric de novo sequencing was studied in [1, 2]. These studies convert the spectrum into a spectrum graph. Each peak in the spectrum generates a few of nodes in the spectrum graph, corresponding to the different types of ions that may produce the peak. Each edge in the graph indicates that the mass difference of the two adjacent nodes is approximately the mass of an amino acid, and the edge is labeled with the amino acid. When at least one of each pair of b_i and y_{n-i} matches a peak in the spectrum, the de novo sequencing problem is reduced to the finding of the "anti-symmetric" longest path in the graph. A dynamic programming algorithm for such purpose was published in [1].

Theorem 1 ([1]) *The longest anti-symmetric path in a spectrum graph $G = \langle V, E \rangle$ can be found in $O(|V| \, |E|)$ time.*

Under Definition 2, de novo sequencing was studied in [6] and a polynomial time algorithm was provided. The algorithm is again a dynamic programming algorithm. For two mass values (m, m'), the dynamic programming calculates an optimal pair of prefix Aa and suffix $a'A'$, such that

1. *$m(Aa) = m$ and $m(a'A') = m'$.*
2. *Either $c_b + m(A) < c_y + m(a'A') \leq c_b + m(Aa)$ or $c_y + m(A') \leq c_b + m(A) < c_y + m(a'A')$.*

The calculation for (m, m') is based on the optimal solutions of smaller mass values. Because of the second above requirement, it is proved in [6] that not all pairs of (m, m') are needed. This is used to speed up the algorithm. A carefully designed strategy can eventually output a prefix and a suffix so that their concatenation form the optimal solution of the de novo sequencing problem. More specifically, the following theorem holds.

Theorem 2 ([5]) *The de novo sequencing problem has an algorithm that gives the optimal peptide in $O(|\Sigma| \times \delta \times \max_{a \in \Sigma} m(a) \times M)$.*

Because $|\Sigma|$, δ, $\max_{a\in\Sigma} m(a)$ are all constants, the algorithm in fact runs in linear time with a large coefficient.

Although the above algorithms are designed to maximize the total intensities of the matched peaks, they can be adapted to work on more sophisticated score functions. Some studies of other score functions can be found in [2–5]. Some of these score functions require new algorithms.

Applications

The algorithms have been implemented into software programs to assist the analyses of tandem mass spectrometry data. Software using the spectrum graph approach includes Sherenga [2]. The de novo sequencing algorithm under the second definition was implemented in PEAKS [5]. More complete lists of the de novo sequencing software and their comparisons can be found in [7, 9].

URL to Code

PEAKS free trial version is available at http://www.bioinfor.com/.

Recommended Reading

1. Chen T, Kao M-Y, Tepel M, Rush J, Church G (2001) A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry. J Comput Biol 8(3):325–337
2. Dančík V, Addona T, Clauser K, Vath J, Pevzner P (1999) De novo protein sequencing via tandem mass-spectrometry. J Comput Biol 6:327–341
3. Fischer B, Roth V, Roos F, Grossmann J, Baginsky S, Widmayer P, Gruissem W, Buhmann J (2005) NovoHMM: a hidden Markov model for de novo peptide sequencing. Anal Chem 77:7265–7273
4. Frank A, Pevzner P (2005) Pepnovo: De novo peptide sequencing via probabilistic network modeling. Anal Chem 77:964–973
5. Ma B, Zhang K, Lajoie G, Doherty-Kirby A, Hendrie C, Liang C, Li M (2003) PEAKS: powerful software for peptide de novo sequencing by tandem mass spectrometry. Rapid Commun Mass Spectrom 17(20):2337–2342
6. Ma B, Zhang K, Liang C (2005) An effective algorithm for the peptide de novo sequencing from MS/MS spectrum. J Comput Syst Sci 70:418–430
7. Pevtsov S, Fedulova I, Mirzaei H, Buck C, Zhang X (2006) Performance evaluation of existing de novo sequencing algorithms. J Proteome Res 5(11):3018–3028. ASAP Article 10.1021/pr060222h
8. Steen H, Mann M (2004) The ABC's (and XYZ's) of peptide sequencing. Nat Rev Mol Cell Biol 5(9):699–711
9. Xu C, Ma B (2006) Software for computational peptide identification from MS/MS. Drug Discov Today 11(13/14):595–600

Perceptron Algorithm

Shai Shalev-Shwartz
School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel

Keywords

Online learning; Single-layer neural network

Years and Authors of Summarized Original Work

1959; Rosenblatt

Problem Definition

The Perceptron algorithm [1, 13] is an iterative algorithm for learning classification functions. The Perceptron was mainly studied in the online learning model. As an online learning algorithm, the Perceptron observes instances in a sequence of trials. The observation at trial t is denoted by $\mathbf{x}_t$. After each observation, the Perceptron predicts a yes/no $(+/-)$ outcome, denoted $\hat{y}_t$, which is calculated as follows:

$$\hat{y}_t = \text{sign}(\langle \mathbf{w}_t, \mathbf{x}_t \rangle),$$

where $\mathbf{w}_t$ is a weight vector which is learned by the Perceptron and $\langle \cdot, \cdot \rangle$ is the inner product operation. Once the Perceptron has made a prediction, it receives the correct outcome, denoted y_t, where $y_t \in \{+1, -1\}$. If the prediction

of the Perceptron was incorrect, it updates its weight vector, presumably improving the chance of making an accurate prediction on subsequent trials. The update rule of the Perceptron is

$$\mathbf{w}_{t+1} = \begin{cases} \mathbf{w}_t + y_t \mathbf{x}_t & \text{if } \hat{y}_t \neq y_t \\ \mathbf{w}_t & \text{otherwise} \end{cases}. \quad (1)$$

The quality of an online learning algorithm is measured by the number of prediction mistakes it makes along its run. Novikoff [12] and Block [2] have shown that whenever the Perceptron is presented with a sequence of linearly separable examples, it makes a bounded number of prediction mistakes which does not depend on the length of the sequence of examples. Formally, let $(\mathbf{x}_i, y_1), \ldots, (\mathbf{x}_T, y_T)$ be a sequence of instance-label pairs. Assume that there exists a unit vector $\mathbf{u}(\|\mathbf{u}\|_2 = 1)$ and a positive scalar $\gamma > 0$ such that for all t, $y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq \gamma$. In words, $\mathbf{u}$ separates the instance space into two half-spaces such that positively labeled instances reside in one half-space, while the negatively labeled instances belong to the second half-space. Moreover, the distance of each instance to the separating hyperplane, $\{\mathbf{x} : \mathbf{u} \cdot \mathbf{x} = 0\}$, is at least γ. The scalar γ is often referred to as the margin attained by $\mathbf{u}$ on the sequence of examples. Novikoff and Block proved that the number of prediction mistakes the Perceptron makes on a sequence of linearly separable examples is at most $(R/\gamma)^2$, where $R = \max_t \|\mathbf{x}_t\|_2$ is the minimal radius of an origin-centered ball enclosing all the instances. In 1969, Minsky and Papert [11] underscored serious limitations of the Perceptron by showing that it is impossible to learn many classes of patterns using the Perceptron (e.g., XOR functions). This fact caused a significant decrease of interest in the Perceptron. The Perceptron has gained back its popularity after Freund and Schapire [9] proposed to use it in conjunction with kernels. The kernel-based Perceptron not only can handle non-separable datasets but can also be utilized for efficiently classifying nonvectorial instances such as trees and strings (see, e.g., [5]).

To implement the Perceptron in conjunction with kernels, one can utilize the fact that at each trial of the algorithm, the weight vector $\mathbf{w}_t$ can be written as a linear combination of the instances

$$\mathbf{w}_t = \sum_{x \in I_t} y_i \mathbf{x}_i,$$

where $I_t = \{i < t : \hat{y}_i \neq y_i\}$ is the set of indices of trials in which the Perceptron made a prediction mistake. Therefore, the prediction of the algorithm can be rewritten as

$$\hat{y}_t = \text{sign}\left(\sum_{i \in I_t} y_i \ \langle \mathbf{x}_i, \mathbf{x}_t \rangle\right),$$

and the update rule of the weight vector can be replaced with an update rule for the set of erroneous trials

$$I_{t+1} = \begin{cases} I_t \cup \{t\} & \text{if } \hat{y}_t \neq y_t \\ I_t & \text{otherwise} \end{cases}. \quad (2)$$

In the kernel-based Perceptron, the inner product $\langle \mathbf{x}_i, \mathbf{x}_t \rangle$ is replaced with a Mercer kernel

Perceptron Algorithm, Table 1 Correspondence between the standard Perceptron algorithm and the kernel-based Perceptron

Online Perceptron	Kernel-based online Perceptron
INITIALIZATION: $\mathbf{w}_1 = \mathbf{0}$	INITIALIZATION: $\mathbf{I}_1 = \{\cdot\}$
For $t = 1, 2, \ldots$	For $t = 1, 2, \ldots$
Receive an instance $\mathbf{x}_t$	Receive an instance $\mathbf{x}_t$
Predict an outcome $\hat{y}_t = \text{sign}(\langle \mathbf{x}_t, \mathbf{x}_t \rangle)$	Predict an outcome $\hat{y}_t = \text{sign}\left(\sum_{i \in I_t} K(\mathbf{x}_i, \mathbf{x}_t)\right)$
Receive correct outcome $y_t \in \{+1, -1\}$	Receive correct outcome $y_t \in \{+1, -1\}$
Update: $\mathbf{w}_{t+1} = \begin{cases} \mathbf{w}_t + y_t \mathbf{x}_t & \text{if } \hat{y}_t \neq y_t \\ \mathbf{w}_t & \text{otherwise} \end{cases}$	Update: $I_{t+1} = \begin{cases} I_t \cup \{t\} & \text{if } \hat{y}_t \neq y_t \\ I_t & \text{otherwise} \end{cases}$

function, $K(\mathbf{x}_i, \mathbf{x}_t)$, without any further changes to the algorithm (for a discussion on Mercer kernels, see, e.g., [15]). Intuitively, the kernel function $K(\mathbf{x}_i, \mathbf{x}_t)$ implements an inner product $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_t) \rangle$ where ϕ is a nonlinear mapping from the original instance space into another (possibly high-dimensional) Hilbert space. Even if the original instances are not linearly separable, the images of the instances due to the nonlinear mapping ϕ can be linearly separable and thus the kernel-based Perceptron can handle non-separable datasets. Since the analysis of the Perceptron does not depend on the dimensionality of the instances, all of the formal results still hold when the algorithm is used in conjunction with kernel functions (Table 1).

Key Results

In the following a mistake bound for the Perceptron in the non-separable case (see, e.g., [10, 14]) is provided.

Theorem Assume that the Perceptron is presented with the sequence of examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ and denote $R = \max_t \|\mathbf{x}_t\|_2$. Let $\mathbf{u}$ be a unit length weight vector ($\|\mathbf{u}\|_2 = 1$), let $\gamma > 0$ be a scalar, and denote

$$L = \sum_{t=1}^{T} \max\{0, 1 - y_t \langle \mathbf{u}/\gamma, \mathbf{x}_t \rangle\}.$$

Then, the number of prediction mistakes the Perceptron makes on the sequence of example is at most

$$L + \left(\frac{R}{\gamma}\right)^2 + \frac{R\sqrt{L}}{\gamma}.$$

Note that if there exists $\mathbf{u}$ and γ such that $y_t \langle \mathbf{u}, \mathbf{x}_t \rangle \geq \gamma$ for all t, then $L = 0$ and the above bound reduces to Novikoff's bound,

$$\left(\frac{R}{\gamma}\right)^2.$$

Note also that the bound does not depend on the dimensionality of the instances. Therefore, it holds for the kernel-based Perceptron as well with $R = \max_t K(\mathbf{x}_t, \mathbf{x}_t)$.

Applications

So far the Perceptron has been viewed in the prism of online learning. Freund and Schapire [9] proposed a simple conversion of the Perceptron algorithm to the batch learning setting. A batch learning algorithm receives as input a training set of examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$ sampled independently from an underlying joint distribution over the instance and label space. The algorithm is required to output a single classification function which performs well on unseen examples as long as the unseen examples are sampled from the same distribution as the training set. The conversion of the Perceptron to the batch setting proposed by Freund and Schapire is called the voted Perceptron algorithm. The idea is to simply run the online Perceptron on the training set of examples, thus producing a sequence of weight vectors $\mathbf{w}_1, \ldots, \mathbf{w}_T$. Then, the single classification function to be used for unseen examples is a majority vote over the predictions of the weight vectors. That is,

$$f(\mathbf{x}) = \begin{cases} +1 \text{ if} |\{t : \langle \mathbf{w}_t, \mathbf{x} \rangle > 0\}| > \\ \qquad |\{t : \langle \mathbf{w}_t, \mathbf{x} \rangle < 0\}| \\ -1 \text{ otherwise} \end{cases}$$

It was shown (see again [9]) that if the number of prediction mistakes the Perceptron makes on the training set is small, then $f(\mathbf{x})$ is likely to perform well on unseen examples as well.

Finally, it should be noted that the Perceptron algorithm was used for other purposes such as solving linear programming [3] and training support vector machines [14]. In addition, variants of the Perceptron were used for numerous additional problems such as online learning on a budget [4,8], multiclass categorization and ranking problems [6,7], and discriminative training for hidden Markov models [5].

Cross-References

▸ Support Vector Machines

Recommended Reading

1. Agmon S (1954) The relaxation method for linear inequalities. Can J Math 6(3):382–392
2. Block HD (1962) The perceptron: a model for brain functioning. Rev Mod Phys 34:123–135
3. Blum A, Dunagan JD (2002) Smoothed analysis of the perceptron algorithm for linear programming. In: Proceedings of the thirteenth annual symposium on discrete algorithms, San Francisco
4. Cesa-Bianchi N, Gentile C (2006) Tracking the best hyperplane with a simple budget perceptron. In: Proceedings of the nineteenth annual conference on computational learning theory, Pittsburgh
5. Collins M (2002) Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In: Conference on empirical methods in natural language processing, Philadelphia
6. Crammer K, Dekel O, Keshet J, Shalev-Shwartz S, Singer Y (2006) Online passive aggressive algorithms. J Mach Learn Res 7:551–585
7. Crammer K, Singer Y (2002) A new family of online algorithms for category ranking. In: Proceedings of the 25th annual international ACM SIGIR conference on research and development in information retrieval, Tampere
8. Dekel O, Shalev-Shwartz S, Singer Y (2005) The Forgetron: a kernel-based perceptron on a fixed budget. Adv neural Inf Process Syst 18: 259–266
9. Freund Y, Schapire RE (1998) Large margin classification using the perceptron algorithm. In: Proceedings of the eleventh annual conference on computational learning theory, Madison
10. Gentile C (2002) The robustness of the p-norm algorithms. Mach Learn 53(3), 265–299
11. Minsky M, Papert S (1969) Perceptrons: an introduction to computational geometry. MIT, Cambridge
12. Novikoff ABJ (1962) On convergence proofs on perceptrons. In: Proceedings of the symposium on the mathematical theory of automata, New York, vol XII, pp 615–622
13. Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65:386–407
14. Shalev-Shwartz S, Singer Y (2005) A new perspective on an old perceptron algorithm. In: Proceedings of the eighteenth annual conference on computational learning theory, Bertinoro, 264–278
15. Vapnik VN (1998) Statistical learning theory. Wiley, New York

Perfect Phylogeny (Bounded Number of States)

Jesper Jansson
Laboratory of Mathematical Bioinformatics, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto, Japan

Keywords

Bounded number of states; Character state matrix; Phylogenetic reconstruction; Phylogenetic tree; Perfect phylogeny

Years and Authors of Summarized Original Work

1994; Agarwala, Fernández-Baca
1997; Kannan, Warnow

Problem Definition

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of elements called *objects* and let $C = \{c_1, c_2, \ldots, c_m\}$ be a set of functions called *characters* such that each $c_j \in C$ is a function from S to the set $\{0, 1, \ldots, r_j - 1\}$ for some integer r_j. For every $c_j \in C$, the set $\{0, 1, \ldots, r_j - 1\}$ is called the set of *allowed states* of character c_j, and for any $s_i \in S$ and $c_j \in C$, it is said that the *state of* s_i *on* c_j is α, or that the *state of* c_j *for* s_i is α, where $\alpha = c_j(s_i)$. The *character state matrix* for S and C is the $(n \times m)$-matrix in which entry (i, j) for any $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, m\}$ equals the state of s_i on c_j.

In this encyclopedia entry, a *phylogeny for* S is an unrooted tree whose leaves are bijectively labeled by S. For every $c_j \in C$ and $\alpha \in \{0, 1, \ldots, r_j - 1\}$, define the set $S_{c_j,\alpha}$ by $S_{c_j,\alpha} = \{s_i \in S : \text{the state of } s_i \text{ on } c_j \text{ is } \alpha\}$. A *perfect phylogeny for* (S, C) (if one exists) is a phylogeny T for S such that the following holds: for each $c_j \in C$ and pair of allowed states α, β of c_j with $\alpha \neq \beta$, the minimal subtree of T

that connects $S_{c_j,\alpha}$ and the minimal subtree of T that connects $S_{c_j,\beta}$ are vertex disjoint. See Fig. 1 for an example. *The Perfect Phylogeny Problem* (also called *the Character Compatibility Problem* in the literature [2,9]) is the following:

Problem 1 (The Perfect Phylogeny Problem)

INPUT: An $(n \times m)$-character state matrix M for some S and C.

OUTPUT: A perfect phylogeny for (S, C), if one exists; otherwise, *null*.

Below, we define $r = \max_{j \in \{1,2,\ldots,m\}} r_j$ for the input character state matrix M.

Key Results

The following negative result was proved by Bodlaender, Fellows, and Warnow [2] and, independently, by Steel [14]:

Theorem 1 ([2, 14]) *The Perfect Phylogeny Problem is NP-hard.*

On the other hand, certain restrictions of the Perfect Phylogeny Problem can be solved efficiently. One such special case occurs if the number of allowed states of each character is limited. For this case, Agarwala and Fernández-Baca [1] designed a dynamic programming-based algorithm that builds perfect phylogenies on certain subsets of S called *c-clusters* (also referred to as *proper clusters* in [5, 10] and as *character subfamilies* in [6]) in a bottom-up fashion. Each c-cluster G has the property that: (1) G and $S \setminus G$ share at most one state of each character; and (2) for at least one character, G and $S \setminus G$ share no states. The number of c-clusters is at most $2^r m$, and the algorithm's total running time is $O(2^{3r}(nm^3 + m^4))$, i.e., exponential in r. Hence, the Perfect Phylogeny Problem is polynomial-time solvable when the number of allowed states of every character is upper-bounded by $O(\log(m+n))$. Subsequently, Kannan and Warnow [10] presented a modified algorithm with improved running time. They restructured the algorithm of [1] to eliminate one of the three nested loops that steps through all possible c-clusters and added a preprocessing step which speeds up the innermost loop. The resulting time complexity is given by:

Theorem 2 ([10]) *The algorithm of Kannan and Warnow in [10] solves the Perfect Phylogeny Problem in $O(2^{2r} nm^2)$ time.*

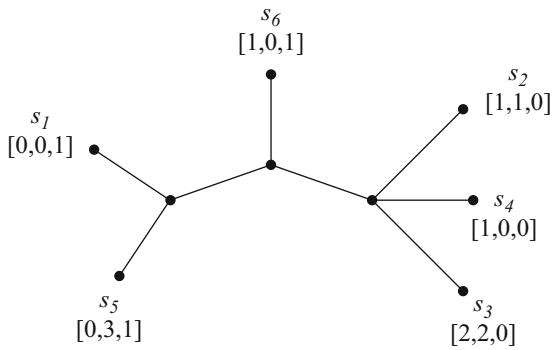
A perfect phylogeny T for (S, C) is called *minimal* if no tree which results by contracting an edge of T is a perfect phylogeny for (S, C). In [10], Kannan and Warnow also showed how to extend their algorithm to enumerate all minimal perfect phylogenies for (S, C) by constructing a directed acyclic graph that implicitly stores the set of all perfect phylogenies for (S, C).

Theorem 3 ([10]) *The extended algorithm of Kannan and Warnow in [10] enumerates the set of all minimal perfect phylogenies for (S, C) so*

a

M	c_1	c_2	c_3
s_1	0	0	1
s_2	1	1	0
s_3	2	2	0
s_4	1	0	0
s_5	0	3	1
s_6	1	0	1

b



Perfect Phylogeny (Bounded Number of States), Fig. 1 (**a**) An example of a character state matrix M for $S = \{s_1, s_2, \ldots, s_6\}$ and $C = \{c_1, c_2, c_3\}$ with $r_1 = 3$, $r_2 = 4$, and $r_3 = 2$, i.e., $r = 4$. (**b**) A perfect phylogeny for (S, C). For convenience, the states of all three characters for each object in S are shown

Perfect Phylogeny (Bounded Number of States), Table 1 The running times of the fastest known algorithms for the Perfect Phylogeny Problem with a bounded number of states

r	Running time	Reference
2	$O(nm)$	[11] together with [7]
3	$\min\{O(nm^2), O(n^2m)\}$	[3, 10] together with [9]
4	$\min\{O(nm^2), O(n^2m)\}$	[10] together with [9]
≥ 5	$O(2^{2r}nm^2)$	[10]

that the maximum computation time between two consecutive outputs is $O(2^{2r}nm^2)$.

For small values of r, even faster algorithms are known. Refer to the table in Table 1 for a summary. If $r = 2$, then the problem can be solved in $O(nm)$ time by reducing it to the *Directed* Perfect Phylogeny Problem for Binary Characters (see, e.g., Encyclopedia entry ▸ Directed Perfect Phylogeny (Binary Characters) for a definition of this variant of the problem) using $O(nm)$ time [7,11] and then applying Gusfield's $O(nm)$-time algorithm [7]. If $r = 3$ or $r = 4$, the problem is solvable in $O(n^2m)$ time by another algorithm by Kannan and Warnow [9], which is faster than the algorithm from Theorem 2 when $n < m$. Also note that for the case $r = 3$, there exists an older algorithm by Dress and Steel [3] whose time complexity coincides with that of the algorithm in Theorem 2.

For other special cases of the Perfect Phylogeny Problem that can be solved efficiently, see Encyclopedia entry ▸ Directed Perfect Phylogeny (Binary Characters) or the survey by Fernández-Baca [5].

Applications

Computational evolutionary biology relies on efficient methods for inferring, from some given data, a phylogenetic tree that accurately describes the evolutionary relationships among a set of objects (e.g., biological species, proteins, genes, etc.) assumed to have been produced by an evolutionary process. One of the most widely used techniques for reconstructing a phylogenetic tree is to represent the objects as vectors of character states and look for a tree that clusters objects which have a lot in common. The Perfect Phylogeny Problem can be regarded as the ideal special case of this approach in which the given data contains no errors, evolution is treelike, and each character state can emerge only once in the evolutionary history.

However, data obtained experimentally seldom admits a perfect phylogeny, so various optimization versions of the problem such as *maximum parsimony* and *maximum compatibility* are often considered in practice. These strategies generally lead to NP-complete problems, but there exist heuristics that work well for most inputs. See, e.g., [4, 5, 12] for a discussion. Nevertheless, algorithms for the Perfect Phylogeny Problem may be useful even when the data does not admit a perfect phylogeny, for example, if there exists a perfect phylogeny for $m - O(1)$ of the characters in C. In fact, in one crucial step of their proposed character-based methodology for determining the evolutionary history of a set of related natural languages, Warnow, Ringe, and Taylor [15] consider all subsets of C in decreasing order of cardinality, repeatedly applying the algorithm of [10] until a largest subset of C which admits a perfect phylogeny is found. The ideas behind the algorithms of [1] and [10] have also been utilized and extended by Fernández-Baca and Lagergren [6] in their algorithm for computing *near-perfect phylogenies* in which the constraints on the output have been relaxed in order to permit non-perfect phylogenies whose so-called penalty score is less than or equal to a prespecified parameter q; see [6] for details. (See also [13] for a fixed-parameter tractable algorithm for this problem variant when $r = 2$.)

The motivation for considering a bounded number of states is that characters based on directly observable traits are, by the way they are defined, naturally bounded by some small number (often 2). When biomolecular data is

used to define characters, the number of allowed states is typically bounded by a constant, e.g., $r = 2$ for SNP markers, $r = 4$ for DNA or RNA sequences, or $r = 20$ for amino acid sequences. (see also Encyclopedia entry ▸ Directed Perfect Phylogeny (Binary Characters)). Moreover, characters with $r = 2$ can be useful in comparative linguistics [8].

Open Problems

An open problem is to determine whether the time complexity of the algorithm of Kannan and Warnow [10] can be improved. As noted in [5], it would be interesting to find out if the Perfect Phylogeny Problem is solvable in $O(2^{2r}nm)$ time for any r, or more generally, in $O(f(r)\cdot nm)$ time, where f is a function of r which does not depend on n or m, since this would match the fastest known algorithm for the special case $r = 2$ (see Table 1). Another open problem is to establish lower bounds on the computational complexity of the Perfect Phylogeny Problem with a bounded number of states.

Cross-References

- ▸ Directed Perfect Phylogeny (Binary Characters)
- ▸ Perfect Phylogeny Haplotyping

Acknowledgments JJ was funded by the Hakubi Project at Kyoto University and KAKENHI grant number 26330014.

Recommended Reading

1. Agarwala R, Fernández-Baca D (1994) A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. SIAM J Comput 23(6):1216–1224
2. Bodlaender HL, Fellows MR, Warnow T (1992) Two strikes against perfect phylogeny. In: Proceedings of the 19th international colloquium on automata, languages and programming (ICALP 1992), Vienna. Lecture notes in computer science, vol 623. Springer, Berlin/Heidelberg, pp 273–283
3. Dress A, Steel M (1992) Convex tree realizations of partitions. Appl Math Lett 5(3):3–6
4. Felsenstein J (2004) Inferring phylogenies. Sinauer Associates, Sunderland
5. Fernández-Baca D (2001) The perfect phylogeny problem. In: Cheng X, Du DZ (eds) Steiner trees in industry. Kluwer Academic, Dordrecht, pp 203–234
6. Fernández-Baca D, Lagergren J (2003) A polynomial-time algorithm for near-perfect phylogeny. SIAM J Comput 32(5):1115–1127
7. Gusfield DM (1991) Efficient algorithms for inferring evolutionary trees. Networks 21:19–28
8. Kanj IA, Nakhleh L, Xia G (2006) Reconstructing evolution of natural languages: Complexity and parameterized algorithms. In: Proceedings of the 12th annual international computing and combinatorics conference (COCOON 2006), Taipei. Lecture notes in computer science, vol 4112. Springer, Berlin/Heidelberg, pp 299–308
9. Kannan S, Warnow T (1994) Inferring evolutionary history from DNA sequences. SIAM J Comput 23(4):713–737
10. Kannan S, Warnow T (1997) A fast algorithm for the computation and enumeration of perfect phylogenies. SIAM J Comput 26(6):1749–1763
11. McMorris FR (1977) On the compatibility of binary qualitative taxonomic characters. Bull Math Biol 39(2):133–138
12. Setubal JC, Meidanis J (1997) Introduction to Computational Molecular Biology. PWS Publishing Company, Boston
13. Sridhar S, Dhamdhere K, Blelloch GE, Halperin E, Ravi R, Schwartz R (2007) Algorithms for efficient near-perfect phylogenetic tree reconstruction in theory and practice. IEEE/ACM Trans Comput Biol Bioinform 4(4):561–571
14. Steel MA (1992) The complexity of reconstructing trees from qualitative characters and subtrees. J Classif 9(1):91–116
15. Warnow T, Ringe D, Taylor A (1996) Reconstructing the evolutionary history of natural langauges. In: Proceedings of the 7th annual ACM-SIAM symposium on discrete algorithms (SODA'96), Atlanta, pp 314–322

Perfect Phylogeny Haplotyping

Giuseppe Lancia
Department of Mathematics and Computer Science, University of Udine, Udine, Italy

Keywords

Alleles phasing

Years and Authors of Summarized Original Work

2005; Ding, Filkov, Gusfield

Problem Definition

In the context of the *perfect phylogeny haplotyping* (PPH) problem, each vector $h \in \{0,1\}^m$ is called a *haplotype*, while each vector $g \in \{0,1,2\}^m$ is called a *genotype*. Haplotypes are binary encodings of DNA sequences, while genotypes are ternary encodings of *pairs* of DNA sequences (one sequence for each of the two homologous copies of a certain chromosome).

Two haplotypes h' and h'' are said to *resolve* a genotype g if, at each position j: (i) if $g_j \in \{0,1\}$ then both $h'_j = g_j$ and $h''_j = g_j$; (ii) if $g_j = 2$ then either $h'_j = 0$ and $h''_j = 1$ or $h'_j = 1$ and $h''_j = 0$. If h' and h'' resolve g, we write $g = h' + h''$. An instance of the PPH problem consists of a set $G = \{g^1, g^2, \ldots, g^n\}$ of genotypes. A set H of haplotypes such that, for each $g \in G$, there are $h', h'' \in H$ with $g = h' + h''$, is called a *resolving set* for G.

A *perfect phylogeny* for a set H of haplotypes is a rooted tree T for which

- the set of leaves is H and the root is labeled by some binary vector r;
- each index $j \in \{1, \ldots, m\}$ labels exactly one edge of T;
- if an edge e is labeled by an index k, then, for each leaf h that can be reached from the root via a path through e, it is $h_k \neq r_k$.

Without loss of generality, it can be assumed that the vector labeling the root is $r = 0$. Within the PPH problem, T is meant to represent the evolution of the sequences at the leaves from a common ancestral sequence (the root). Each edge labeled with an index represents a point in time when a mutation happened at a specific site. This model of evolution is also known as *coalescent* [11]. It can be shown that a perfect phylogeny for H exists if and only if for all choices of four haplotypes $h^1, \ldots, h^4 \in H$ and two indices i, j,

$$\{h^a_i\, h^a_j\,, 1 \leq a \leq 4\} \neq \{00, 01, 10, 11\}\,.$$

Given the above definitions, the problem surveyed in this entry is the following:

Perfect Phylogeny Haplotyping Problem (PPH)

Given a set G of genotypes, find a resolving set H of haplotypes and a perfect phylogeny T for H, or determine that such a resolving set does not exist.

In a slightly different version of the above problem, one may require to find all perfect phylogenies for H instead of just one (in fact, all known algorithms for PPH do find all perfect phylogenies).

The perfect phylogeny problem was introduced by Gusfield [7], who also proposed a nearly linear-time $O(nm\,\alpha(nm))$-algorithm for its solution (where $\alpha()$ is the extremely slowly growing inverse Ackerman function). The algorithm resorted to a reduction to a complex procedure for the *graph realization* problem (Bixby and Wagner [2]), of very difficult understanding and implementation. Later approaches for PPH proposed much simpler, albeit slower, $O(nm^2)$-algorithms (Bafna et al. [1]; Eskin et al. [6]). However, a major question was left open: *does there exist a linear-time algorithm for PPH*? In [7], Gusfield conjectured that this should be the case. The 2005 algorithm by Ding, Filkov, and Gusfield [5] surveyed in this entry settles the above conjecture in the affirmative.

Key Results

The main idea of the algorithm is to find the maximal sub-graphs that are common to all PPH solutions. Let us call *P-class* a maximal sub-graph of all PPH trees for G. The authors show

that each P-class consists of two sub-trees which, in each PPH solution, can appear in either one of two possible ways (called *flips* of the P-class) with respect to any fixed P-class taken as a reference. Hence, if there are k P-classes, there are 2^{k-1} distinct PPH solutions.

The algorithm resorts to an original and effective data structure, called the *shadow tree*, which gives an implicit representation of all P-classes. The data structure is built incrementally, by processing one genotype at a time. The total cost for building and updating the shadow tree is linear in the input size (i.e., in nm). A detailed description of the shadow tree requires a rather large number of definitions, possibly accompanied by figures and examples. Here, we will introduce only its basic features, those that allow us to state the main theorems of [5].

The shadow tree is a particular type of directed rooted tree, which contains both *edges* and *links* (strictly speaking, the latter are just arcs, but they are called links to underline their specific use in the algorithm). The edges are of two types: *tree-edges* and *shadow-edges*, and are associated to the indices $\{1, \ldots, m\}$. For each index i, there is a tree-edge labeled t_i and a shadow-edge labeled s_i. Both edges and links are oriented, with their head closer to the root than their tail. Other than the root, each node of the shadow tree is the endpoint of exactly one tree-edge or one shadow-edge (while the root is the head of two "dummy" links). The links are used to connect certain tree- and shadow-edges. A link can be either *free* or *fixed*. The head of a free link can still be changed during the execution of the algorithm, but once a link is fixed, it cannot be changed any more.

Tree-edges, shadow-edges and *fixed* links are organized into *classes*, which are sub-graphs of the shadow tree. Each fixed link is contained in exactly one class, while each free link connects one class to another, called its *parent*. For each index i, if the tree-edge t_i is in a class X, then the shadow-edge s_i is in X as well, so that a class can be seen as a pair of "twin" sub-trees of the shadow tree. The free links point out from the root of the sub-trees (the *class roots*). Classes change during the running of the algorithm. Specifically, classes are *created* (containing a single tree- and shadow-edge) when a new genotype is processed; a class can be *merged* with its parent, by fixing a pair of free edges; finally, a class can be *flipped*, by switching the heads of the two free links that connect the class roots to the parent class.

A tree T is said to be "*contained in*" a shadow tree if T can be obtained by flipping some classes in the shadow tree, followed by contracting all links and shadow-edges. Let us call *contraction* of a class the sub-graph (consisting of a pair of sub-trees, made of tree-edges only) that is obtained from a class X of the shadow tree by contracting all fixed links and shadow-edges of X. The following are the main results obtained in [5]:

Proposition 1 *Every P-class can be obtained by contraction of a class of the final shadow tree produced by the algorithm. Conversely, every contraction of a class of the final shadow tree is a P-class.*

Theorem 1 *Every PPH solution is contained in the final shadow tree produced by the algorithm. Conversely, every tree contained in the final shadow tree is a distinct PPH solution.*

Theorem 2 *The total time required for building and updating the shadow tree is* $O(nm)$.

Applications

The PPH problem arises in the context of *Single Nucleotide Polymorphisms* (SNP's) analysis in human genomes. A SNP is the site of a single nucleotide which varies in a statistically significant way in a population. The determination of SNP locations and of common SNP patterns (haplotypes) are of uttermost importance. In fact, SNP analysis is used to understand the nature of several genetic diseases, and the international Haplotype Map Project is devoted to SNP study (Helmuth [9]).

The values that a SNP can take are called its *alleles*. Almost all SNPs are bi-allelic, i.e., out of the four nucleotides A, C, T, G, only two are observed at any SNP. Humans are *diploid* organisms, with DNA organized in pairs of chromosomes (of paternal and of maternal origin). The sequence of alleles on a chromosome copy is called a *haplotype*. Since SNPs are bi-allelic, haplotypes can be encoded as binary strings. For a given SNP, an individual can be either *homozygous*, if both parents contributed the same allele, or *heterozygous*, if the paternal and maternal alleles are different.

Haplotyping an individual consists of determining his two haplotypes. Haplotyping a population consists of haplotyping each individual of the population. While it is today economically infeasible to determine the haplotypes directly, there is a cheap experiment which can determine the (less informative and often ambiguous) *genotypes*.

A genotype of an individual contains the *conflated* information about the two haplotypes. For each SNP, the genotype specifies which are the two (possibly identical) alleles, but does not specify their origin (paternal or maternal). The ternary encoding that is used to represent a genotype g has the following meaning: at each SNP j, it is $g_j = 0$ (respectively, 1) if the individual is homozygous for the allele 0 (respectively, 1), and $g_j = 2$ if the individual is heterozygous. There may be many possible pairs of haplotypes that justify a particular genotype (there are 2^{k-1} pairs of haplotypes that can resolve a genotype with k heterozygous SNPs). Given a set of genotypes, in order to determine the correct resolving set out of the exponentially many possibilities, one imposes some "biologically meaningful" constraints that the solution must possess. The perfect phylogeny model (coalescent) requires that the resolving set must fit a particular type of evolutionary tree. That is, all haplotypes should descend from some ancestral haplotype, via mutations that happened (only once) at specific sites over time. The coalescent model is accurate especially for short haplotypes (for longer haplotypes there is also another type of evolutionary event, *recombination*, that should be taken into account).

The linear-time PPH algorithm is of significant practical value in two respects. First, there are instances of the problem where the number of SNPs considered is fairly large (genotypes can extend over several kilo-bases). For these long instances, the advantage of an $O(nm)$ algorithm with respect to the previous $O(nm^2)$ approach is evident. On the other hand, when genotypes are relatively short, the benefit of using the linear-time algorithm is not immediately evident (both algorithms run extremely quickly). Nevertheless, there are situations in which one has to solve a large set of haplotyping problems, where each single problem is defined over short genotypes. For instance, this is the case in which one examines all "small" subsets of SNPs in order to determine the subsets for which there is a PPH solution. In this type of application, the gain of efficiency with the use of the linear-time PPH algorithm is significant (Chung and Gusfield [4]; Wiuf [14]).

Open Problems

A linear-time algorithm is the best possible for PPH, and no open problems are listed in [5].

Experimental Results

The algorithm has been implemented in C and its performance has been compared with the previous fastest PPH algorithm, i.e., DPPH (Bafna et al. [1]). In the case of $m = 2000$ and $n = 1000$, the algorithm is about 250-times faster than DPPH, and is capable of solving an instance in an average time of 2 s, versus almost 8 min needed by DPPH (on a "standard" 2005 Personal Computer). The smaller instances (e.g., with $m = 50$ SNPs) are such that the superior performance of the algorithm is not as evident, with an average running time of 0.07 s versus 0.2 s. However, as already remarked, when the small instances are executed within a loop, the

speed-up turns out to be again of two or more orders of magnitude.

Data Sets

The data sets used in [5] have been generated by the program *ms* (Hudson [12]), which is the widely used standard for instance generation reflecting the coalescent model of SNP sequence evolution. Real-life instances can be found at the HapMap web site http://www.hapmap.org.

URL to Code

http://wwwcsif.cs.ucdavis.edu/~gusfield/lpph/

Cross-References

- Directed Perfect Phylogeny (Binary Characters)
- Perfect Phylogeny (Bounded Number of States)

Recommended Reading

For surveys about computational haplotyping problems in general, see Bonizzoni et al. [3], Gusfield and Orzack [8], Halldorsson et al. [10], and Lancia [13].

1. Bafna V, Gusfield D, Lancia G, Yooseph S (2003) Haplotyping as perfect phylogeny: a direct approach. J Comput Biol 10(3–4):323–340
2. Bixby RE, Wagner DK (1988) An almost linear-time algorithm for graph realization. Math Oper Res 13:99–123
3. Bonizzoni P, Della Vedova G, Dondi R, Li J (2004) The haplotyping problem: an overview of computational models and solutions. J Comput Sci Technol 19(1):1–23
4. Chung RH, Gusfield D (2003) Empirical exploration of perfect phylogeny haplotyping and haplotypes. In: Proceedings of annual international conference on computing and combinatorics (COCOON). Lecture notes in computer science, vol 2697. Springer, Berlin, pp 5–9
5. Ding Z, Filkov V, Gusfield D (2005) A linear-time algorithm for the perfect phylogeny haplotyping problem. In: Proceedings of the annual international conference on computational molecular biology (RECOMB), New York. ACM, New York
6. Eskin E, Halperin E, Karp R (2003) Efficient reconstruction of haplotype structure via perfect phylogeny. J Bioinf Comput Biol 1(1):1–20
7. Gusfield D (2002) Haplotyping as perfect phylogeny: conceptual framework and efficient solutions. In: Myers G, Hannenhalli S, Istrail S, Pevzner P, Waterman M (eds) Proceedings of the annual international conference on computational molecular biology (RECOMB). ACM, New York, pp 166–175
8. Gusfield D, Orzack SH (2005) Haplotype inference. In: Aluru S (ed) Handbook of computational molecular biology. Champman and Hall/CRC, Boca Raton, pp 1–28
9. Helmuth L (2001) Genome research: map of the human genome 3.0. Science 293(5530): 583–585
10. Halldorsson BV, Bafna V, Edwards N, Lippert R, Yooseph S, Istrail S (2004) A survey of computational methods for determining haplotypes. In: Computational methods for SNP and haplotype inference: DIMACS/RECOMB satellite workshop. Lecture notes in computer science, vol 2983. Springer, Berlin, pp 26–47
11. Hudson R (1990) Gene genealogies and the coalescent process. Oxf Surv Evol Biol 7:1–44
12. Hudson R (2002) Generating samples under the wright-fisher neutral model of genetic variation. Bioinformatics 18(2):337–338
13. Lancia G (2008) The phasing of heterozygous traits: algorithms and complexity. Comput Math Appl 55(5):960–969
14. Wiuf C (2004) Inference on recombination and block structure using unphased data. Genetics 166(1): 537–545

P

Performance-Driven Clustering

Rajmohan Rajaraman
Department of Computer Science, Northeastern University, Boston, MA, USA

Keywords

Circuit clustering; Circuit partitioning

Years and Authors of Summarized Original Work

1993; Rajaraman, Wong

Problem Definition

Circuit partitioning consists of dividing the circuit into parts, each of which can be implemented as a separate component (e.g., a chip) that satisfies the design constraints. The work of Rajaraman and Wong [5] considers the problem of dividing a circuit into components, subject to area constraints, such that the maximum delay at the outputs is minimized.

A combinational circuit can be represented as a directed acyclic graph $G = (V, E)$, where V is the set of nodes and E is the set of directed edges. Each node represents a gate in the network and each edge (u, v) in E represents an interconnection between gates u and v in the network. The *fanin* of a node is the number of edges incident into it, and the *fanout* of a node is the number of edges incident out of it. A *primary input* (PI) is a node with fanin 0, while a *primary output* (PO) is a node with fanout 0. Each node has a *weight* and a *delay* associated with it.

Definition 1 A clustering of a network $G = (V, E)$ is a triple (H, ϕ, Σ), where

1. $H = (V', E')$ is a directed acyclic graph.
2. φ is a function mapping V' to V such that
   - For every edge $(u', v') \in E'$, $(\phi(u'), \phi(v')) \in E$.
   - For every node $v' \in V'$ and edge $(u, \phi(v')) \in E$, there exists a unique $u' \in V'$ such that $\phi(u') = u$ and $(u', v') \in E'$.
   - For every PO node $v \in V$, there exists a unique $v' \in V'$ such that $\phi(v') = v$.
3. Σ is a partition of V'.

Let $\Gamma = (H = (V', E'), \phi, \Sigma)$ be a clustering of G. For $v \in V, v' \in V'$, if $\phi(v') = v$, we call v' a *copy* of v. The set V' consists of all the copies of the nodes in V that appear in the clustering. A node v' is a PI (respectively, PO) in Γ if $\phi(v')$ is a PI (respectively, PO) in G. It follows from the definition of φ that H is logically equivalent to G.

The weights and delays on the individual nodes in G yield weights and delays of nodes in H' and a delay for the clustering Γ. The weight (respectively, delay) of a node v' in V' is the weight (respectively, delay) of $\phi(v)$. The weight of any cluster $C \in \Sigma$, denoted by $W(C)$, is the sum of the weights of the nodes in C. The delay of a clustering is given by the general delay model of Murgai et al. [3], which is as follows. The delay of an edge $(u', v') \in E'$ is D (which is a given parameter) if u' and v' belong to different elements of Σ and zero otherwise. The delay along a path in H' is simply the sum of the delays of the edges of the path. Finally, the delay of Γ is the delay of a maximum-delay path in H', among all the paths from a PI node to a PO node in H'.

Definition 2 Given a combinational network $G = (V, E)$ with weight function $w : V \to R^+$, weight capacity M, and a delay function $\delta : V \to R^+$, we say that a clustering $\Gamma = (H, \phi, \Sigma)$ is *feasible* if for every cluster $C \in \Sigma, W(C)$ is at most M. The *circuit clustering problem* is to compute a feasible clustering Γ of G such that the delay of Γ is minimum among all feasible clusterings of G.

An early work of Lawler et al. [2] presented a polynomial-time optimal algorithm for the circuit clustering problem in the special case where all the gate delays are zero (i.e., $\delta(v) = 0$ for all v).

Key Results

Rajaraman and Wong [5] presented an optimal polynomial-time algorithm for the circuit clustering problem under the general delay model.

Theorem 1 *There exists an algorithm that computes an optimal clustering for the circuit clustering problem in $O(n^2 \log n + nm)$ time, where n and m are the vertices and edges, respectively, of the given combinational network.*

This result can be extended to compute optimal clusterings under any monotone clustering constraint. A clustering constraint is monotone if any connected subset of nodes in a feasible cluster is also monotone [2].

Theorem 2 *The circuit clustering problem can be solved optimally under any monotone clustering constraint in time polynomial in the size of the circuit.*

Applications

Circuit partitioning/clustering is an important component of very large scale integration design. One application of the circuit clustering problem formulated above is to implement a circuit on multiple field programmable gate array chips. The work of Rajaraman and Wong focused on clustering combinational circuits to minimize delay under area constraints. Related studies have considered other important constraints, such as pin constraints [1] and a combination of area and pin constraints [6]. Further work has also included clustering sequential circuits (as opposed to combinational circuits) with the objective of minimizing the clock period [4].

Experimental Results

Rajaraman and Wong reported experimental results on five ISCAS (International Symposium on Circuits and Systems) circuits. The number of nodes in these circuits ranged from 196 to 913. They reported the maximum delay of the clusterings and running times of their algorithm on a Sun Sparc workstation.

Cross-References

▸ FPGA Technology Mapping

Recommended Reading

1. Cong J, Ding Y (1992) An optimal technology mapping algorithm for delay optimization in lookup-table based fpga design. In: Proceedings of IEEE international conference on computer-aided design, Santa Clara, pp 48–53
2. Lawler EL, Levitt KN, Turner J (1966) Module clustering to minimize delay in digital networks. IEEE Trans Comput C-18:47–57
3. Murgai R, Brayton RK, Sangiovanni-Vincentelli A (1991) On clustering for minimum delay/area. In: Proceedings of IEEE international conference on computer-aided design, Santa Clara, pp 6–9
4. Pan P, Karandikar AK, Liu CL (1998) Optimal clock period clustering for sequential circuits with retiming. IEEE Trans Comput-Aided Des Integr Circuits Syst 17:489–498
5. Rajaraman R, Wong DF (1995) Optimum clustering for delay minimization. IEEE Trans Comput-Aided Des Integr Circ Syst 14:1490–1495
6. Yang HH, Wong DF (1997) Circuit clustering for delay minimization under area and pinconstraints. IEEE Trans Comput-Aided Des Integr Circ Syst 16:976–986

Permutation Enumeration

Katsuhisa Yamanaka
Department of Electrical Engineering and Computer Science, Iwate University, Iwate, Japan

Keywords

Combinatorial Gray code; Enumeration; Permutation; Partition search; Reverse search

Years and Authors of Summarized Original Work

1962; Trotter
1963; Johnson
1977; Sedgewick
2008; Sekine, Yamanaka, Nakano

Problem Definition

Permutation Enumeration

Let S_n be the set of permutations of $[n] = \{1, 2, \ldots, n\}$. We write a permutation as a sequence of elements in $[n]$ such that each element appears exactly once. A *permutation enumeration* is to list all permutations in S_n. For

example, there are 24 permutations of [4]: 1234, 1243, 1324, 1342, 1423, 1432, 2134, 2143, 2314, 2341, 2413, 2431, 3124, 3142, 3214, 3241, 3412, 3421, 4123, 4132, 4213, 4231, 4312, 4321. The enumeration of permutations is a basic and long-standing enumeration problem, and it was surveyed by Sedgewick [8].

Note that the above example lists all the permutations of [4], and we have listed them in lexicographic order, which is the most natural way to enumerate them. The purpose of this paper is to introduce representative methods for enumeration problems by showing how these methods are applied to permutation enumeration.

Efficiency of Enumeration Algorithms

The efficiency of an algorithm is measured by the time and space complexity for a given input size. However, many enumeration problems have an exponential number of outputs (solutions) for a given input. Hence, an enumeration algorithm may require exponential time in order to output the solutions. Typically, an enumeration algorithm is measured by the "delay time." We say that an enumeration algorithm has delay d if (1) it takes at most d time to output the first object, and (2) it takes at most d time between two consecutive outputs. See [1, 3, 4] for further details. Note that the delay time does not include the time required to output the objects, since this is typically ignored when estimating the time complexity of an enumeration algorithm. The space complexity of an enumeration algorithm is an estimate of the amount of working memory required by the algorithm (as in the usual sense).

Key Results

Enumeration by Partition Search

In the partition search enumeration method, objects are listed by repeatedly partitioning the set of objects. As an example, we will apply the partition search method to permutation enumeration. We will partition S_n by fixing the first element of a permutation. Denote by $S_n(i) \subseteq S_n$ the set of permutations in which i in $[n]$ is the first element. Then, S_n is partitioned into $S_n(1), S_n(2), \ldots, S_n(n)$. Hence, if we have the list of all permutations of $[n] \setminus \{i\}$ for each $i = 1, 2, \ldots, n$, then we can enumerate all permutations in S_n by appending i as the first element to every permutation. This recursive structure gives the algorithm shown as Algorithm 1. To begin, Algorithm 1 is called with the empty sequence and the empty set. The algorithm recursively fixes the first element, and we then obtain all permutations in S_n. Figure 1 illustrates a tree structure of recursive calls of the algorithm. The root corresponds to the empty sequence. Each vertex of the tree corresponds to the prefix of a permutation, and it is obtained by removing the last element of its child. The leaves correspond to the permutations in S_n. Algorithm 1 traverses the tree structure in a depth-first manner.

Algorithm 1: PARTITION-SEARCH(π, S)

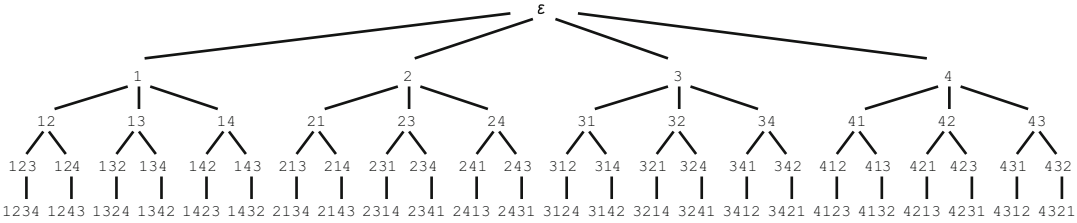
```
1 π is the current subpermutation, and S is the set of
  elements in π;
2 if S = [n] then    /* π is a permutation
  in S_n */
3 |   Output π;
4 |   return;
5 foreach i ∈ [n] \ S do
6 |   PARTITION-SEARCH(π + i, S ∪ {i});
  |         /* The operation '+' is a
  |   concatenation */
```

We now estimate the running time of Algorithm 1. Let T be the running time for traversing an edge of the search tree. Since the depth of the search tree is n, the delay time of the algorithm is $O(nT)$ time in worst case.

Theorem 1 *One can enumerate all permutations by the partition search method. The running time of the algorithm is $O(nT)$, where T is the time for traversing an edge of the search tree. The required working space is $O(n)$.*

Enumeration by Gray Code

A *combinatorial Gray code* is a list of all of the objects in some class such that two consecutive objects in the list differ only by a small amount. Since the list contains all of the ob-



Permutation Enumeration, Fig. 1 Search tree of the partition search method

jects, an algorithm that generates a combinatorial Gray code can be regarded as an enumeration algorithm. There are combinatorial Gray codes for various combinatorial objects, and these have been surveyed [7]. For a permutation, a difference between two consecutive objects would be a swap of two adjacent elements, where we say that the i-th and the $(i+1)$-th element in a permutation are *adjacent*. For permutations, the well-known Steinhaus-Johnson-Trotter algorithm (or Johnson-Trotter algorithm) [5,8,10,11] generates the combinatorial Gray code for the permutations of $[n]$. This algorithm is regarded as an enumeration of permutations.

Let $\pi = p_1 p_2 \dots p_{n-1}$ in S_{n-1}, and denote by $\pi(i) = p_1 p_2 \dots p_i n p_{i+1} \dots p_{n-1}$ for $i = 0, 1, \dots, n-1$ the permutation obtained from π by inserting n between p_i and p_{i+1}. Then, the list of $\pi(0), \pi(1), \dots, \pi(n-1)$ or $\pi(n-1), \pi(n-2), \dots, \pi(0)$ is a combinatorial Gray code for a subset of S_n. Such lists can be defined for all permutations in S_{n-1}, and the lists for all permutations in S_{n-1} contain all permutations in S_n. Assume that we have a combinatorial Gray code for S_{n-1}. Let π_i be the i-th permutation in the list. Then, we construct the list $\pi_i(0), \pi_i(1), \dots, \pi_i(n-1)$ if i is even, and $\pi_i(n-1), \pi_i(n-2), \dots, \pi_i(0)$ if i is odd. The obtained list is a combinatorial Gray code for S_n. Note that if i is even, then $\pi_{i+1}(n-1)$ is obtained from $\pi_i(n-1)$ by swapping two adjacent elements, where π_{i+1} is the $(i+1)$-th permutation in a combinatorial Gray code for S_{n-1}. Similarly, if i is odd, then $\pi_{i+1}(0)$ is obtained from $\pi_i(0)$ by swapping two adjacent elements. By recursively applying this idea, we can design a combinatorial Gray code for S_n.

Now we explain the details of Steinhaus-Johnson-Trotter algorithm. The algorithm first outputs the identity permutation $\iota = 12 \dots n$. Let us consider the case of $\pi = p_1 p_2 \dots p_n$ in $S_n \setminus \{\iota\}$, and let us assume that π is generated from $\pi' = p'_1 p'_2 \dots p'_n$ by swapping two adjacent elements in π'. We construct the next permutation of π by swapping n and its left-adjacent or its right-adjacent element. More precisely, the rule of swapping is as follows.

1. n is the last element of π.
   - 1-1. n is the second-to-last element of π'.
     In this case, π is obtained from π' by swapping $p'_{n-1} = n$ and p'_n by Step 3. We swap two elements in $[n-1]$ by recursively applying this swapping algorithm to the subpermutation $p_1 p_2 \dots p_{n-1}$, which is obtained from π by removing the element n. Let π_s be the obtained subpermutation. Then, we append n to π_s as the last element. The obtained permutation is the next permutation.
   - 1-2. n is also the last element of π'.
     We construct the next permutation of π by swapping p_{n-1} and $p_n = n$.
2. n is the first element of π. Similar to Step 1, we construct the next permutation of π, as follows:
   - 2-1. n is the second element of π'.
     We recursively apply this swapping algorithm to the subpermutation $p_2 p_3 \dots p_n$. Let π_t be the obtained subpermutation. Then, we append n to π_t as the first element. The obtained permutation is the next permutation.

Permutation Enumeration, Table 1 List of S_4 in combinatorial Gray code ($n = 4$ is underlined)

123$\underline{4}$	312$\underline{4}$	231$\underline{4}$
124$\underline{3}$	314$\underline{2}$	234$\underline{1}$
142$\underline{3}$	341$\underline{2}$	243$\underline{1}$
$\underline{4}$123	$\underline{4}$312	$\underline{4}$231
$\underline{4}$132	$\underline{4}$321	$\underline{4}$213
143$\underline{2}$	342$\underline{1}$	241$\underline{3}$
134$\underline{2}$	324$\underline{1}$	214$\underline{3}$
132$\underline{4}$	321$\underline{4}$	213$\underline{4}$

2-2. n is also the first element of π'.

We construct the next permutation of π by swapping $p_1 = n$ and p_2.

3. Otherwise.

We swap $p_i = n$ and p_{i+1} if π is obtained from π' by swapping $p'_{i-1} = n$ and p'_i of π', and swap p_{i-1} and $p_i = n$ if π is obtained from π' by swapping p'_i and $p'_{i+1} = n$ of π'.

Table 1 shows the list of permutations in S_4 enumerated by the above swapping algorithm. Note that any permutation can be obtained by swapping two adjacent elements.

Pseudocodes for the above algorithm are shown in Algorithm 2, which is the main routine, and Algorithm 3, which is a subroutine which generates the next permutation of a given permutation. These pseudocodes assume that a direction vector $d = (d(1), d(2), \ldots, d(n))$ is stored in global memory. Each $d(i)$ for $i = 1, 2, \ldots, n$ represents the direction in which the element i in the current permutation goes to obtain the next permutation. More precisely, an instruction of "left" or "right" is stored in each $d(i)$. By using the direction vector, we know in which direction two adjacent elements were swapped without needing to check the current permutation and the preceding permutation.

Our implementations (Algorithms 2 and 3) are not efficient, but an efficient loopless algorithm was given by Sedgewick [8], as in the following theorem.

Theorem 2 ([8]) *After constructing the identity permutation in $O(n)$ time, one can enumerate all permutations in S_n in the order of a combinatorial Gray code with a constant time delay.*

It can be observed that the combinatorial Gray coder order defined by the algorithm represents a Hamiltonian path of the permutohedron. Figure 2 shows a permutohedron of S_4 and its Hamiltonian path corresponding to the combinatorial Gray code.

Enumeration by Reverse Search

Avis and Fukuda [2] proposed a reverse search enumeration method. The idea of the reverse search method is as follows: We first define a rooted tree structure for the objects such that each vertex corresponds to an object and each edge corresponds to a relation between two objects. Then, by traversing the tree structure, we enumerate all of the objects.

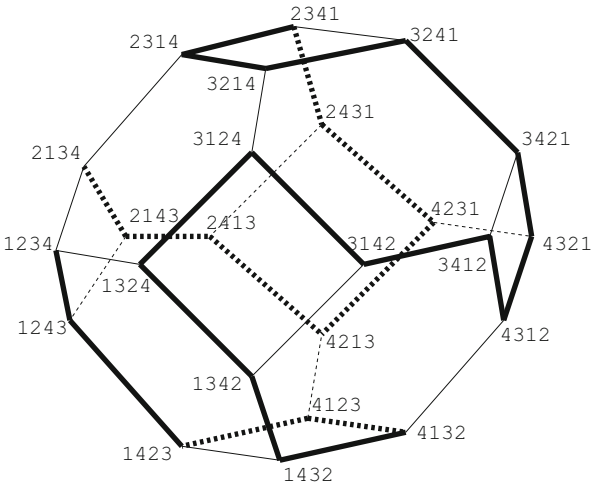
Now, we illustrate the reverse search method by applying it to permutation enumeration [9]. We define a rooted tree structure for S_n, as follows. To construct a tree structure, we define its root and the parent of each permutation

Algorithm 2: GRAY-CODE(n)

```
1  d(i) for i = 1,2,...,n represents in which direction the element i goes;
2  foreach i = 1,2,...,n do
   /* Initialization of direction vector d(i) */
3      d(i) ← left
4  π ← 12...n        /* Set the identity permutation to π. */;
5  foreach i = 1,2,...,n! do
6      Output π;
7      π ← SWAP(n,π);
```

Algorithm 3: SWAP($n, \pi = p_1 p_2 \ldots p_n$)

```
1  if n = 1 then return π
2  else if p_n = n and d(n) = right then
3      π' ← SWAP(n − 1, p_1 p_2 ... p_{n−1});
4      d(n) ← left;
5      return π' + n
6  else if p_1 = n and d(n) = left then
7      π' ← SWAP(n − 1, p_2 p_3 ... p_n);
8      d(n) ← right;
9      return n + π'
10 else return the permutation obtained from π by swapping n and its left-adjacent or its right-adjacent element depending on d(n)
```

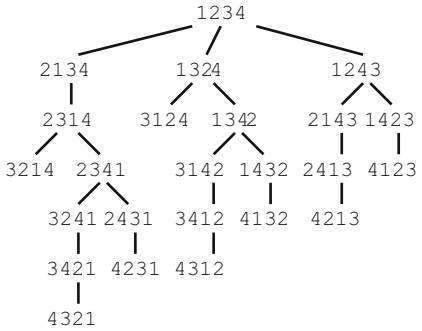


Permutation Enumeration, Fig. 2 Permutohedron of S_4 and its Hamiltonian path

in S_n except the root. We define the identity permutation $\iota = 12\ldots n$ as the *root* of the tree structure. Then, we define the parent of a permutation by an adjacent swap of two elements. Intuitively, the parent is defined so that there is greater similarity between it and ι than there is between the child and ι. A formal definition is as follows. Let $\pi = p_1 p_2 \ldots p_n \in S_n \setminus \{\iota\}$ be a permutation, and let i be the minimum index such that $p_i > p_{i+1}$ holds. The *parent permutation* of π, denoted by $P(\pi)$, is the permutation obtained from π by swapping p_i and p_{i+1}. Then, we call π a *child permutation* of $P(\pi)$. Note that the parent permutation of π is uniquely defined. For example, $P(3421) = 3241$, $P(3241) = 2341$, $P(2341) = 2314$, $P(2314) = 2134$, and $P(2134) = 1234$ are obtained. By repeatedly finding the parent permutations, we have the sequence of permutations in S_n, which ends up with the identity permutation. By merging these sequences, we have the tree structure, called the *family tree* T_n of S_n. Figure 3 shows the family tree T_4.

We next design an algorithm that traverses the family tree by recursively generating all of the child permutations of any permutation. Intuitively, the operation to generate a child permutation is the reverse of finding the parent permutation.

We introduce some notation. Let $\pi = p_1 p_2 \ldots p_n$ be a permutation in S_n. Then, we de-



Permutation Enumeration, Fig. 3 The family tree T_4

note by $\pi[i] = p_1 p_2 \ldots p_{i-1} p_{i+1} p_i p_{i+2} \ldots p_n$, the permutation obtained from π by swapping p_i and p_{i+1}. Note that $\pi[i]$ is a child permutation if and only if $\pi = P(\pi[i])$ holds. A key point to find i with $\pi = P(\pi[i])$ is to maintain the *reverse point* $r(\pi)$, which is the minimum index of π such that $p_{r(\pi)} > p_{r(\pi)+1}$. For convenience, we set $r(\iota) = n$ for the identity permutation in S_n. Note that the subpermutation $p_1 p_2 \ldots p_{r(\pi)}$ is the maximal increasing prefix of π. If we know $r(\pi)$ of π, all child permutations are generated, as follows. For each $i = 1, 2, \ldots, r(\pi) - 1$, $\pi[i]$ is a child permutation of π since $\pi = P(\pi[i])$ holds. If $p_{r(\pi)} < p_{r(\pi)+2}$ holds, then $\pi[r(\pi)+1]$ is a child permutation of π. Otherwise, $\pi[r(\pi) + 1]$ is not a child permutation. For each $i = r(\pi), r(\pi) + 2, r(\pi) + 3, \ldots, n - 1$, $\pi[i]$

Algorithm 4: REVERSE-SEARCH($\pi = p_1 p_2 \ldots p_n$)

1 Let $r(\pi)$ be a reverse point of π;
2 Output π;
3 **for** ***each*** $i = 1, 2, \ldots, r(\pi) - 1$ **do**
4 $\quad$ REVERSE-SEARCH($\pi[i]$)
5 **if** $r(\pi) \leq n - 2$ *and* $p_{r(\pi)} < p_{r(\pi)+2}$ **then** REVERSE-SEARCH($\pi[r(\pi) + 1]$)

is not a child permutation. Based on the above observation, we obtain the enumeration algorithm shown in Algorithm 4. To begin, Algorithm 4 is called with the identity permutation which is the root of the family tree.

By maintaining the reverse point of the current permutation in a traverse of the family tree, we can use a stack to generate each child permutation in O(1) time. To estimate the running time of the algorithm, note that the algorithm can traverse each edge of the family tree in O(1) time. However, the delay time of the algorithm is not bounded by O(1) time for the case that the next permutation is output after deep recursive calls without outputting any permutation. However, by applying the speed-up method proposed by Nakano and Uno [6], we have the following lemma.

Theorem 3 ([9]) *After constructing the root (the identity permutation) in $O(n)$ time, one can enumerate all the permutations in S_n by the reverse search method with a constant time delay. The required working space is $O(n)$.*

Recommended Reading

1. Arimura H, Uno T (2007) An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. J Comb Optim 13:243–262
2. Avis D, Fukuda K (1996) Reverse search for enumeration. Discret Appl Math 65(1–3):21–46
3. Goldberg L (1992) Efficient algorithm for listing unlabeled graphs. J Algorithms 13:128–143
4. Goldberg L (1993) Polynomial space polynomial delay algorithms for listing families of graphs. In: Proceedings of the 25th annual ACM symposium on theory of computing, San Diego, pp 218–225
5. Johnson S (1963) Generation of permutations by adjacent transposition. Math Comput 17:282–285
6. Nakano S, Uno T (2004) Constant time generation of trees with specified diameter. In: Proceedings of the 30th workshop on graph-theoretic concepts in computer science (WG 2004), Bad Honnef. LNCS, vol 3353, pp 33–45
7. Savage C (1997) A survey of combinatorial gray codes. SIAM Rev 39(4):605–629
8. Sedgewick R (1977) Permutation generation methods. Comput Surv 9(2):137–164
9. Sekine K, Yamanaka K, Nakano S (2008) Enumeration of permutations. IEICE Trans Fundam Electron Commun Comput Sci J91-A(5):543–549 (in Japanese)
10. Steinhaus H (1964) One hundred problems in elementary mathematics. Basic Books, New York
11. Trotter H (1962) Perm (algorithm 115). Commun ACM 5(8):434–435

Phylogenetic Tree Construction from a Distance Matrix

Jesper Jansson
Laboratory of Mathematical Bioinformatics, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto, Japan

Keywords

Additive; Dissimilarity matrix; Distance matrix; Phylogenetic tree; Phylogenetic reconstruction; Tree-realizable

Years and Authors of Summarized Original Work

1968; Boesch
1989; Hein
1989; Culberson, Rudnicki
2003; King, Zhang, Zhou

Problem Definition

Let n be a positive integer. A *distance matrix of order n* is a matrix D of size $(n \times n)$ which

satisfies (1) $D_{i,j} > 0$ for all $i, j \in \{1, 2, \ldots, n\}$ with $i \neq j$; (2) $D_{i,j} = 0$ for all $i, j \in \{1, 2, \ldots, n\}$ with $i = j$; and (3) $D_{i,j} = D_{j,i}$ for all $i, j \in \{1, 2, \ldots, n\}$. In the literature, a distance matrix of order n is also called a *dissimilarity matrix of order n*.

Below, all trees are assumed to be unrooted and edge-weighted. For any tree $\mathcal{T}$, the *distance* between two nodes u and v in $\mathcal{T}$ is defined as the sum of the weights of all edges on the unique path in $\mathcal{T}$ between u and v and is denoted by $d^{\mathcal{T}}_{u,v}$. A tree $\mathcal{T}$ is said to *realize* a given distance matrix D of order n if and only if it holds that $\{1, 2, \ldots, n\}$ is a subset of the nodes of $\mathcal{T}$ and $d^{\mathcal{T}}_{i,j} = D_{i,j}$ for all $i, j \in \{1, 2, \ldots, n\}$. Finally, a distance matrix D is called *additive* or *tree-realizable* if and only if there exists a tree which realizes D. See Fig. 1 for an example.

Problem 1 (The Phylogenetic Tree from Distance Matrix Problem)
INPUT: A distance matrix D of order n
OUTPUT: A tree which realizes D and has the smallest possible number of nodes, if D is additive, otherwise *null*

In the time complexities listed below, the time needed to input all of D is not included. Instead, $O(1)$ is charged to the running time whenever an algorithm requests to know the value of any specified entry of D.

Key Results

Several authors have independently shown how to solve the Phylogenetic Tree from Distance Matrix Problem in $O(n^2)$ time. (See [5] for a short survey of older algorithms which do not run in $O(n^2)$ time.)

Theorem 1 ([2, 4, 5, 7, 14]) *There exists an algorithm which solves the Phylogenetic Tree from Distance Matrix Problem in $O(n^2)$ time.*

Although the various existing algorithms are different, it can be proved that:

Theorem 2 ([8, 14]) *For any given distance matrix, the solution to the Phylogenetic Tree from Distance Matrix Problem is unique.*

Furthermore, the algorithms referred to in Theorem 1 have optimal running time since any algorithm for the Phylogenetic Tree from Distance Matrix Problem must in the worst case query all $\Omega(n^2)$ entries of D to make sure that D is additive. However, if it is known in advance that the input distance matrix is additive, then the time complexity improves as follows.

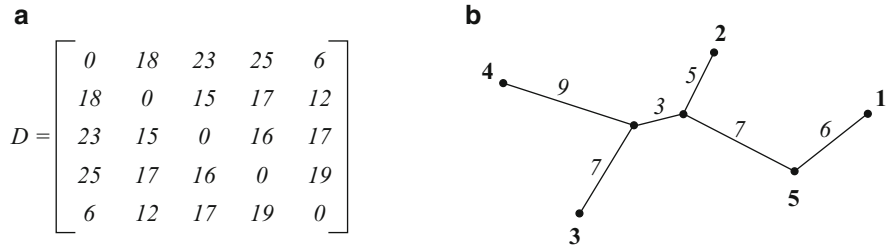
Theorem 3 ([9, 12]) *There exists an algorithm which solves the Phylogenetic Tree from Distance Matrix Problem restricted to additive distance matrices in $O(k n \log_k n)$ time, where k is the maximum degree of the tree that realizes the input distance matrix.*

The algorithm of Hein [9] starts with a tree containing just two nodes and then successively inserts each node i into the tree by repeatedly choosing a pair of existing nodes and computing where on the path between them that i should be attached, until i's position has been determined. The same basic technique is used in the $O(n^2)$-time algorithm of Waterman et al. [14] referenced to by Theorem 1 above, but the algorithm of Hein selects paths which are more efficient at discriminating between the possible positions for i. According to [12], the running time of Hein's algorithm is $O(k n \log_k n)$.

A lower bound that implies the optimality of Theorem 3 is given by the next theorem.

Theorem 4 ([10]) *The Phylogenetic Tree from Distance Matrix Problem restricted to additive distance matrices requires $\Omega(k n \log_k n)$ queries to the distance matrix D, where k is the maximum degree of the tree that realizes D, even if restricted to trees in which all edge weights are equal to* 1.

Independently of [9], Culberson and Rudnicki [5] presented an algorithm for the Phylogenetic Tree from Distance Matrix Problem and claimed it to have $O(k n \log_k n)$ time complexity when restricted to additive distance



Phylogenetic Tree Construction from a Distance Matrix, Fig. 1 (**a**) An additive distance matrix D of order 5. (**b**) A tree $\mathcal{T}$ which realizes D. Here, $\{1, 2, \ldots, 5\}$ forms a subset of the nodes of $\mathcal{T}$

matrices and trees in which all edge weights are equal to 1. As pointed out by Reyzin and Srivastava [12], the algorithm actually runs in $\Theta(n^{3/2}\sqrt{k})$ time. See [12] for a counterexample to [5] and a correct analysis. On the positive side, the following special case is solvable in linear time by the Culberson-Rudnicki algorithm:

Theorem 5 ([5]) *There exists an $O(n)$-time algorithm which solves the Phylogenetic Tree from Distance Matrix Problem restricted to additive distance matrices for which the realizing tree contains two leaves only and has all edge weights equal to* 1.

Applications

The main application of the Phylogenetic Tree from Distance Matrix Problem is in the construction of a tree (a so-called *phylogenetic tree*) that represents evolutionary relationships among a set of studied objects (e.g., species or other taxa, populations, proteins, genes, etc.). Here, it is assumed that the objects are indeed related according to a treelike branching pattern caused by an evolutionary process and that their true pairwise evolutionary distances are proportional to the measured pairwise dissimilarities. See, e.g., [1, 6, 7, 14] for examples and many references as well as discussions on how to estimate pairwise dissimilarities based on biological data. Other applications of the Phylogenetic Tree from Distance Matrix Problem can be found in psychology, for example, to describe semantic memory organization [1], in comparative linguistics to infer the evolutionary history of a set of languages [11], or in the study of the filiation of manuscripts to trace how manuscript copies of a text (whose original version may have been lost) have evolved in order to identify discrepancies among them or to reconstruct the original text [1, 3, 13].

In general, real data seldom forms additive distance matrices [14]. Therefore, in practice, researchers consider optimization versions of the Phylogenetic Tree from Distance Matrix Problem which look for a tree that "almost" realizes D. Many alternative definitions of "almost" have been proposed, and numerous heuristics and approximation algorithms have been developed. A comprehensive description of some of the most popular methods for phylogenetic reconstruction from a non-additive distance matrix such as *Neighbor-joining* [16] as well as more background information can be found in, e.g., Chapter 11 of [6]. See also [1] and [15] and the references therein.

Cross-References

- ▸ Distance-Based Phylogeny Reconstruction (Fast-Converging)
- ▸ Distance-Based Phylogeny Reconstruction: Safety and Edge Radius

Acknowledgments JJ was funded by the Hakubi Project at Kyoto University and KAKENHI grant number 26330014.

Recommended Reading

1. Abdi H (1990) Additive-tree representations. In: Dress A, von Haeseler A (eds) Trees and hierarchical structures: proceedings of a conference held at Bielefeld, FRG, Oct 5–9th, 1987. Lecture Notes in Biomathematics, vol 84. Springer, Berlin/Heidelberg, pp 43–59
2. Batagelj V, Pisanski T, Simões-Pereira JMS (1990) An algorithm for tree-realizability of distance matrices. Int J Comput Math 34(3–4):171–176
3. Bennett CH, Li M, Ma B (2003) Chain letters and evolutionary histories. Sci Am 288(6):76–81
4. Boesch FT (1968) Properties of the distance matrix of a tree. Quart Appl Math 26:607–609
5. Culberson JC, Rudnicki P (1989) A fast algorithm for constructing trees from distance matrices. Inf Process Lett 30(4):215–220
6. Felsenstein J (2004) Inferring phylogenies. Sinauer Associates, Sunderland
7. Gusfield DM (1997) Algorithms on strings, trees, and sequences. Cambridge University Press, New York
8. Hakimi SL, Yau SS (1964) Distance matrix of a graph and its realizability. Quart Appl Math 22:305–317
9. Hein J (1989) An optimal algorithm to reconstruct trees from additive distance data. Bull Math Biol 51(5):597–603
10. King V, Zhang L, Zhou Y (2003) On the complexity of distance-based evolutionary tree construction. In: Proceedings of the 14th annual ACM-SIAM symposium on discrete algorithms (SODA 2003), Baltimore, pp 444–453
11. Nakhleh L, Warnow T, Ringe D, Evans SN (2005) A comparison of phylogenetic reconstruction methods on an Indo-European dataset. Trans Philol Soc 103(2):171–192
12. Reyzin L, Srivastava N (2007) On the longest path algorithm for reconstructing trees from distance matrices. Inf Process Lett 101(3):98–100
13. The Canterbury Tales Project. University of Birmingham, Brigham Young University, University of Münster, New York University, Virginia Tech, and Keio University. Website: http://www.petermwrobinson.me.uk/canterburytalesproject.com/
14. Waterman MS, Smith TF, Singh M, Beyer WA (1977) Additive evolutionary trees. J Theor Biol 64(2):199–213
15. Wu BY, Chao K-M, Tang CY (1999) Approximation and exact algorithms for constructing minimum ultrametric trees from distance matrices. J Combin Optim 3(2–3):199–211
16. Saitou N, Nei M (1987) The Neighbor-joining Method: A New Method for Reconstructing Phylogenetic Trees. Mol Biol Evol 4(4):406–425

Planar Directed k-Vertex-Disjoint Paths Problem

Marcin Pilipczuk
Institute of Informatics, University of Bergen, Bergen, Norway
Institute of Informatics, University of Warsaw, Warsaw, Poland

Keywords

Directed graphs; Fixed-parameter tractability; Graph decomposition; Planar graphs; Vertex-Disjoint Paths problem

Years and Authors of Summarized Original Work

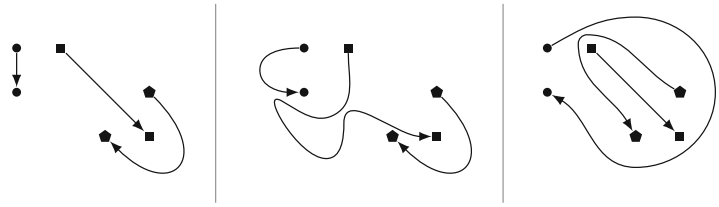
1994; Schrijver
2013; Cygan, Marx, Pilipczuk, Pilipczuk

Problem Definition

In the classic Vertex-Disjoint Paths problem, the input consists of an n-vertex graph G and k pairs of terminals $(s_i, t_i)_{i=1}^{k}$, and the question is whether there exist pairwise Vertex-Disjoint Paths $P_1, P_2, \ldots, P_k$ such that for every $1 \leq i \leq k$, the path P_i starts in s_i and ends in t_i. In this entry we are interested in the complexity of this problem restricted to planar directed graphs.

Key Results

An algorithm for the Vertex-Disjoint Paths problem in undirected graphs with running time $f(k)n^3$ for some function f is one of the key ingredients of the minor testing algorithm of Robertson and Seymour [8]. The approach can be summarized as follows: either the input graph has treewidth bounded by a function of k, in which case we can apply standard dynamic



Planar Directed k-Vertex-Disjoint Paths Problem, Fig. 1 Different homotopy classes of a solution: in the first two figures, the solutions are of the same class, whereas on the third figure the homotopy class is different

programming techniques, or, by the Excluded Grid Theorem, the input graph contains a large grid minor. In the second case, we may deduce that a middle vertex of the grid minor is irrelevant for the problem and can be discarded.

The original proof of the irrelevancy of a middle vertex of a grid minor by Robertson and Seymour [8] is not only highly involved but also leads to an extremely large dependency on k in the running time bound. A more recent algorithm by Kawarabayashi and Wollan [6] improves upon the original approach in both these aspects, but is still very complex.

As already observed by Robertson and Seymour in [7], a situation becomes dramatically simpler if we restrict ourselves to planar graphs. A short self-contained argument of irrelevancy of the middle vertex of a $c^k \times c^k$ grid minor, for some universal constant c, is due to Adler, Kolliopoulos, Krause, Lokshtanov, Saurabh, and Thilikos [1]. It is worth noting that the exponential dependency on k in the irrelevant vertex argument is necessary. The intuitive reason why planarity greatly helps in the Vertex-Disjoint Paths problem is that, on the plane, the solution paths need to correspond to noncrossing curves and one path serves as a separator for other paths. This allows us to use a wide variety of topological arguments.

In directed graphs, the Vertex-Disjoint Paths problem is already NP-hard for two paths ($k = 2$) [3], so we cannot hope for similar results. However, it turns out that in the directed case the planarity assumption is very useful, too. More than 20 years ago, Schrijver showed that the Vertex-Disjoint Paths problem in n-vertex planar directed graphs can be solved in time $n^{\mathcal{O}(k)}$ [9]. Recently, Cygan, Marx, Pilipczuk, and Pilipczuk presented a fixed-parameter algorithm for this problem, running in time $2^{2^{\mathcal{O}(k^2)}} n^{\mathcal{O}(1)}$ [2].

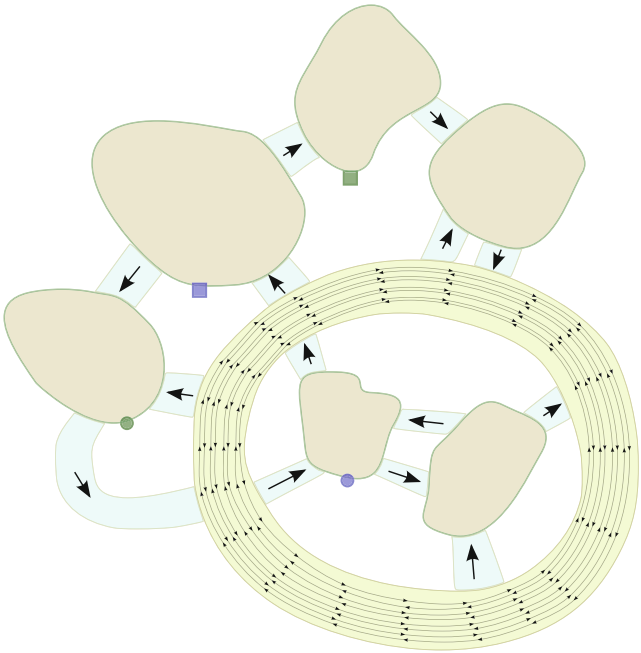
Key Techniques for Planar Directed Graphs

The Schrijver's Algorithm

The approach of Schrijver [9] can be summarized as follows. The main observation is that there are $n^{\mathcal{O}(k)}$ homotopy types of the solution, where two different solutions are considered homotopical if the paths of one solution can be "shifted" (modified by a homotopy) to obtain the second solution, without crossing any face that contains a terminal (without loss of generality, we may assume that all terminals are of degree one, and the notion of a face containing a terminal is well defined). See also Fig. 1 for an illustration. A second ingredient of Schrijver's approach is a polynomial-time algorithm that essentially checks if there exists a solution in one homotopy class. (It should be noted that this statement is a significant simplification, as the Schrijver's algorithm operates on the notions of (co)homologies and in fact searches for a solution in a significant superset of one homotopy class, but that is sufficient for our needs.)

The Fixed-Parameter Algorithm

The first step in the fixed-parameter algorithm of [2] is an appropriate irrelevant vertex rule for the problem. Unfortunately, for directed graphs there is no Excluded Grid Theorem which as convenient as it is in the undirected case. Al-



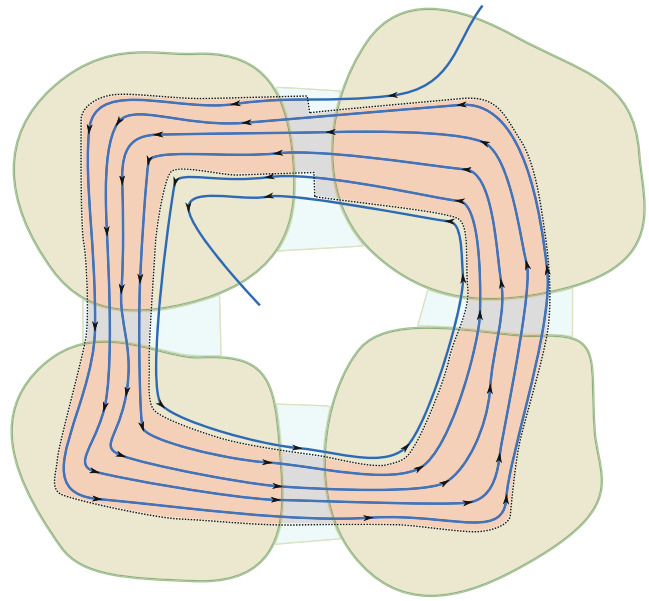
Planar Directed k-VERTEX-DISJOINT PATHS Problem, Fig. 2 An example of a decomposition with six disk components and a single ring component

though the conjecture of Johnson, Robertson, Seymour, and Thomas [4] about the connections between directed treewidth and directed grid minors has been recently proven by Kawarabayashi and Kreutzer for graph excluding a fixed minor [5], neither directed treewidth seems well suited for dynamic programming algorithm for the VERTEX-DISJOINT PATHS problem [10], nor it is clear whether a directed grid minor can be useful for an irrelevant vertex argument. In [2], it is proven that a family of sufficiently many concentric cycles with alternating direction, without any terminal enclosed by the outermost cycle, is sufficient to make an irrelevant vertex argument.

In the light of such an irrelevant vertex rule, the next question is: what is the structure of a graph without many concentric cycles with alternating directions? The answer provided in [2] can be informally stated as follows: such a graph can be decomposed into a bounded (in k) number of disk and ring components, connected by a bounded number of *bundles*, where every bundle is a set of edges in one direction that lie close to each other on the plane (see Fig. 2).

Unfortunately, it is not easy to make algorithmic use of such a decomposition; this should be contrasted with the undirected case, where bounded treewidth immediately yields efficient algorithms via standard dynamic programming approach. The approach taken in [2] is to make use of Schrijver's approach and use the decomposition to enumerate only $2^{2^{\mathcal{O}(k^2)}} n^{\mathcal{O}(1)}$ "reasonable" homotopy types of the solution.

Recall that the $n^{\mathcal{O}(k)}$ term in the time complexity of Schrijver's algorithm comes from the number of homotopy classes. It is quite easy to see that this bound cannot be improved: each of k solution paths can "wind" arbitrary number of times in some part of the graph, leading to different homotopy classes. This is also the case in the decomposition, as can be seen on Fig. 3. To deal with this issue, an involved technical argumentation is developed in [2] to show that for any such place in the graph as on Fig. 3, there is some "canonical" number of turns, and we may assume that the solution will take approximately this number of turns, up to an additive $f(k)$ factor, for some computable function f. This leads to an FPT bound on the number of "reasonable" homotopy classes to consider and, consequently, to an FPT running time bound.



Planar Directed k-Vertex-Disjoint Paths Problem, Fig. 3 An example of a solution path winding many times between four disk components

Cross-References

▸ Bidimensionality

Recommended Reading

1. Adler I, Kolliopoulos SG, Krause PK, Lokshtanov D, Saurabh S, Thilikos DM (2011) Tight bounds for linkages in planar graphs. In: Aceto L, Henzinger M, Sgall J (eds) ICALP (1), Zurich. Lecture notes in computer science, vol 6755, pp 110–121. Springer
2. Cygan M, Marx D, Pilipczuk M, Pilipczuk M (2013) The planar directed k-Vertex-Disjoint Paths problem is fixed-parameter tractable. In: FOCS, Berkeley. IEEE Computer Society, pp 197–206
3. Fortune S, Hopcroft J, Wyllie J (1980) The directed subgraph homeomorphism problem. Theor Comput Sci 10(2):111–121
4. Johnson T, Robertson N, Seymour PD, Thomas R (2001) Directed tree-width. J Comb Theory Ser B 82(1):138–154
5. Kawarabayashi K, Kreutzer S (2014) An excluded grid theorem for digraphs with forbidden minors. In: Chekuri C (ed) SODA, Portland. SIAM, pp 72–81
6. Kawarabayashi K, Wollan P (2010) A shorter proof of the graph minor algorithm: the unique linkage theorem. In: Schulman LJ (ed) STOC, Cambridge. ACM, pp 687–694
7. Robertson N, Seymour PD (1988) Graph minors. VII. Disjoint paths on a surface. J Comb Theory Ser B 45(2):212–254
8. Robertson N, Seymour PD (1995) Graph minors. XIII. The disjoint paths problem. J Comb Theory Ser B 63(1):65–110
9. Schrijver A (1994) Finding k disjoint paths in a directed planar graph. SIAM J Comput 23(4):780–788
10. Slivkins A (2010) Parameterized tractability of edge-disjoint paths on directed acyclic graphs. SIAM J Discret Math 24(1):146–157

Planar Geometric Spanners

Joachim Gudmundsson[1,2], Giri Narasimhan[3,5], and Michiel Smid[4]

[1]DMiST, National ICT Australia Ltd, Alexandria, Australia

[2]School of Information Technologies, University of Sydney, Sydney, NSW, Australia

[3]Department of Computer Science, Florida International University, Miami, FL, USA

[4]School of Computer Science, Carleton University, Ottawa, ON, Canada

[5]School of Computing and Information Sciences, Florida International University, Miami, FL, USA

Keywords

Dilation; Detour; Geometric network

Years and Authors of Summarized Original Work

2005; Bose, Smid, Gudmundsson

Problem Definition

Let S be a set of n points in the plane and let G be an undirected graph with vertex set S, in which each edge (u, v) has a weight, which is equal to the Euclidean distance $|uv|$ between the points u and v. For any two points p and q in S, their shortest-path distance in G is denoted by $\delta_G(p, q)$. If $t \geq 1$ is a real number, then G is a *t-spanner* for S if $\delta_G(p, q) \leq t|pq|$ for any two points p and q in S. Thus, if t is close to 1, then the graph G contains close approximations to the $\binom{n}{2}$ Euclidean distances determined by the pairs of points in S. If, additionally, G consists of $O(n)$ edges, then this graph can be considered a sparse approximation to the complete graph on S. The smallest value of t for which G is a t-spanner is called the *stretch factor* (or *dilation*) of G. For a comprehensive overview of geometric spanners, see the book by Narasimhan and Smid [16].

Assume that each edge (u, v) of G is embedded as the straight-line segment between the points u and v. The graph G is said to be *plane* if its edges intersect only at their common vertices.

In this entry, the following two problems are considered:

Problem 1 Determine the smallest real number $t > 1$ for which the following is true: For every set S of n points in the plane, there exists a plane graph with vertex set S, which is a t-spanner for S. Moreover, design an efficient algorithm that constructs such a plane t-spanner.

Problem 2 Determine the smallest positive integer D for which the following is true: There exists a constant t, such that for every set S of n points in the plane, there exists a plane graph with vertex set S and maximum degree at most D, which is a t-spanner for S. Moreover, design an efficient algorithm that constructs such a plane t-spanner.

Key Results

Let S be a finite set of points in the plane that is in *general position*, i.e., no three points of S are on a line and no four points of S are on a circle. The *Delaunay triangulation* of S is the plane graph with vertex set S, in which (u, v) is an edge if and only if there exists a circle through u and v that does not contain any point of S in its interior. (Since S is in general position, this graph is a triangulation.) The Delaunay triangulation of a set of n points in the plane can be constructed in $O(n \log n)$ time. Dobkin, Friedman and Supowit [10] were the first to show that the stretch factor of the Delaunay triangulation is bounded by a constant: They proved that the Delaunay triangulation is a t-spanner for $t = \pi(1 + \sqrt{5})/2$. The currently best known upper bound on the stretch factor of this graph is due to Keil and Gutwin [12]:

Theorem 1 *Let S be a finite set of points in the plane. The Delaunay triangulation of S is a t-spanner for S, for $t = 4\pi\sqrt{3}/9$.*

A slightly stronger result was proved by Bose et al. [3]. They proved that for any two points p and q in S, the Delaunay triangulation contains a path between p and q, whose length is at most $(4\pi\sqrt{3}/9)|pq|$ and all edges on this path have length at most $|pq|$.

Levcopoulos and Lingas [14] generalized the result of Theorem 1: Assume that the Delaunay triangulation of the set S is given. Then, for any real number $r > 0$, a plane graph G with vertex set S can be constructed in $O(n)$ time, such that G is a t-spanner for S, where $t = (1+1/r)4\pi\sqrt{3}/9$, and the total length of all edges in G is at most $2r + 1$ times the weight of a minimum spanning tree of S.

The Delaunay triangulation can alternatively be defined to be the dual of the *Voronoi diagram* of the set S. By considering the Voronoi diagram for a metric other than the Euclidean metric, a corresponding Delaunay triangulation is obtained. Chew [7] has shown that the Delaunay triangulation based on the Manhattan-metric is a $\sqrt{10}$-spanner (in this spanner, path-lengths are

measured in the Euclidean metric). The currently best result for Problem 1 is due to Chew [8]:

Theorem 2 *Let S be a finite set of points in the plane, and consider the Delaunay triangulation of S that is based on the convex distance function defined by an equilateral triangle. This plane graph is a 2-spanner for S (where path-lengths are measured in the Euclidean metric).*

Das and Joseph [9] have generalized the result of Theorem 1 in the following way (refer to Fig. 1). Let G be a plane graph with vertex set S and let α be a real number with $0 < \alpha < \pi/2$. For any edge e of G, let Δ_1 and Δ_2 be the two isosceles triangles with base e and base angle α. The edge e is said to satisfy the *α-diamond property*, if at least one of the triangles Δ_1 and Δ_2 does not contain any point of S in its interior. The plane graph G is said to satisfy the *α-diamond property*, if every edge e of G satisfies this property. For a real number $d \geq 1$, G satisfies the *d-good polygon property*, if for every face f of G, and for every two vertices p and q on the boundary of f, such that the line segment joining them is completely inside f, the shortest path between p and q along the boundary of f has length at most $d|pq|$. Das and Joseph [9] proved that any plane graph satisfying both the α-diamond property and the d-good polygon property is a t-spanner, for some real number t that depends only on α and d. A slight improvement on the value of t was obtained by Lee [13]:

Theorem 3 *Let $\alpha \in (0, \pi/2)$ and $d \geq 1$ be real numbers, and let G be a plane graph that satisfies the α-diamond property and the d-good polygon property. Then, G is a t-spanner for the vertex set of G, where*

$$t = \frac{8(\pi - \alpha)^2 d}{\alpha^2 \sin^2(\alpha/4)}.$$

To give some examples, it is not difficult to show that the Delaunay triangulation satisfies the α-diamond property with $\alpha = \pi/4$. Drysdale et al. [11] have shown that the minimum weight triangulation satisfies the α-diamond property with $\alpha = \pi/4.6$. Finally, Lee [13] has shown that the greedy triangulation satisfies the α-diamond property with $\alpha = \pi/6$. Of course, any triangulation satisfies the d-good polygon property with $d = 1$.

Now consider Problem 2, that is, the problem of constructing plane spanners whose maximum degree is small. The first result for this problem is due to Bose et al. [2]. They proved that the Delaunay triangulation of any finite point set contains a subgraph of maximum degree at most 27, which is a t-spanner (for some constant t). Li and Wang [15] improved this result, by showing that the Delaunay triangulation contains a t-spanner of maximum degree at most 23. Given the Delaunay triangulation, the subgraphs in [2, 15] can be constructed in $O(n)$ time. The currently best result for Problem 2 is by Bose et al. [6]:

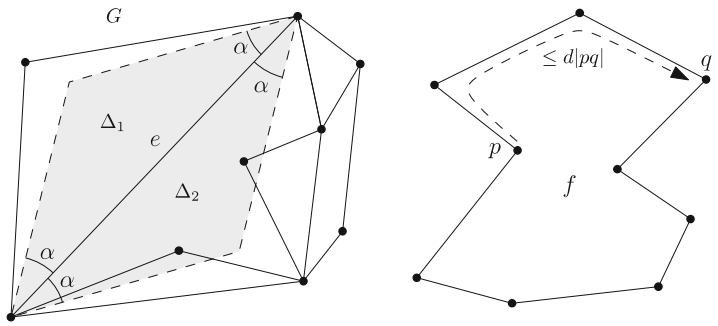
Theorem 4 *Let S be a set of n points in the plane. The Delaunay triangulation of S contains a subgraph of maximum degree at most 17, which is a t-spanner for S, for some constant t. Given the Delaunay triangulation of S, this subgraph can be constructed in $O(n)$ time.*

In fact, the result in [6] is more general:

Theorem 5 *Let S be a set of n points in the plane, let $\alpha \in (0, \pi/2)$ be a real number, and let G be a triangulation of S that satisfies the α-diamond property. Then, G contains a subgraph of maximum degree at most $14 + \lceil 2\pi/\alpha \rceil$, which is a t-spanner for S, where t depends only on α. Given the triangulation G, this subgraph can be constructed in $O(n)$ time.*

Applications

Plane spanners have applications in on-line path-finding and routing problems that arise in, for example, geographic information systems and communication networks. In these application areas, the complete environment is not known, and routing has to be done based only on the source, the destination, and the neighborhood of the current position. Bose and Morin [4, 5] have shown that, in this model, good routing strategies



Planar Geometric Spanners, Fig. 1 On the left, the α-diamond property is illustrated. At least one of the triangles Δ_1 and Δ_2 does not contain any point of S in its interior. On the right, the d-good polygon property is illustrated. p and q are two vertices on the same face f which can see each other. At least one of the two paths between p and q along the boundary of f has length at most $d|pq|$

exist for plane graphs, such as the Delaunay triangulation and graphs that satisfy both the α-diamond property and the d-good polygon property. These strategies are competitive, in the sense that the paths computed have lengths that are within a constant factor of the Euclidean distance between the source and destination. Moreover, these routing strategies use only a limited amount of memory.

Open Problems

None of the results for Problems 1 and 2 that are mentioned in section "Key Results" seem to be optimal. The following problems are open:

1. Determine the smallest real number t, such that the Delaunay triangulation of any finite set of points in the plane is a t-spanner. It is widely believed that $t = \pi/2$. By Theorem 1, $t \leq 4\pi\sqrt{3}/9$.
2. Determine the smallest real number t, such that a plane t-spanner exists for any finite set of points in the plane. By Theorem 2, $t \leq 2$. By taking S to be the set of four vertices of a square, it follows that t must be at least $\sqrt{2}$.
3. Determine the smallest integer D, such that the Delaunay triangulation of any finite set of points in the plane contains a t-spanner (for some constant t) of maximum degree at most D. By Theorem 4, $D \leq 17$. It follows from results in Aronov et al. [1] that the value of D must be at least 3.
4. Determine the smallest integer D, such that a plane t-spanner (for some constant t) of maximum degree at most D exists for any finite set of points in the plane. By Theorem 4 and results in [1], $3 \leq D \leq 17$.

Cross-References

- Applications of Geometric Spanner Networks
- Dilation of Geometric Networks
- Geometric Spanners
- Sparse Graph Spanners

Recommended Reading

1. Aronov B, de Berg M, Cheong O, Gudmundsson J, Haverkort H, Vigneron A (2005) Sparse geometric graphs with small dilation. In: Proceedings of the 16th international symposium on algorithms and computation. Lecture notes in computer science, vol 3827. Springer, Berlin, pp 50–59
2. Bose P, Gudmundsson J, Smid M (2005) Constructing plane spanners of bounded degree and low weight. Algorithmica 42:249–264
3. Bose P, Maheshwari A, Narasimhan G, Smid M, Zeh N (2004) Approximating geometric bottleneck shortest paths. Comput Geom Theory Appl 29:233–249
4. Bose P, Morin P (2004a) Competitive online routing in geometric graphs. Theor Comput Sci 324:273–288
5. Bose P, Morin P (2004b) Online routing in triangulations. SIAM J Comput 33:937–951
6. Bose P, Smid M, Xu D (2006) Diamond triangulations contain spanners of bounded degree. In: Pro-

ceedings of the 17th international symposium on algorithms and computation. Lecture notes in computer science, vol 4288. Springer, Berlin, pp 173–182
7. Chew LP (1986) There is a planar graph almost as good as the complete graph. In: Proceedings of the 2nd ACM symposium on computational geometry, pp 169–177
8. Chew LP (1989) There are planar graphs almost as good as the complete graph. J Comput Syst Sci 39:205–219
9. Das G, Joseph D (1989) Which triangulations approximate the complete graph? In: Proceedings of the international symposium on optimal algorithms. Lecture notes in computer science, vol 401. Springer, Berlin, pp 168–192
10. Dobkin DP, Friedman SJ, Supowit KJ (1990) Delaunay graphs are almost as good as complete graphs. Discret Comput Geom 5:399–407
11. Drysdale RL, McElfresh S, Snoeyink JS (2001) On exclusion regions for optimal triangulations. Discret Appl Math 109:49–65
12. Keil JM, Gutwin CA (1992) Classes of graphs which approximate the complete Euclidean graph. Discret Comput Geom 7:13–28
13. Lee AW (2004) Diamonds are a plane graph's best friend. Master's thesis, School of Computer Science, Carleton University, Ottawa
14. Levcopoulos C, Lingas A (1992) There are planar graphs almost as good as the complete graphs and almost as cheap as minimums panning trees. Algorithmica 8:251–256
15. Li X-Y, Wang Y (2004) Efficient construction of low weighted bounded degree planar spanner. Int J Comput Geom Appl 14:69–84
16. Narasimhan G, Smid M (2007) Geometric spanner networks. Cambridge University Press, Cambridge

Planar Maximum Flow – Multiple-Source Multiple-Sink Maximum Flow in Directed Planar Graphs

Glencora Borradaile
Department of Computer Science, Brown University, Providence, RI, USA
School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA

Keywords

Maximum flow; Planar graphs

Years and Authors of Summarized Original Work

2011; Borradaile, Klein, Mozes, Nussbaum, Wulff-Nilsen

Problem Definition

Given a directed, planar graph $G = (V, E)$ with arc capacities $c : E \to \Re^+$, a subset S of source vertices, and a subset T of sink vertices, the goal is to find a maximum flow from the source vertices to the sink vertices:

$$\max \sum_{su : s \in S, su \in E} f_{su}$$

$$\text{s.t.} \sum_{uv : uv \in E} f_{uv} - \sum_{vw : vw \in E} f_{vw} = 0 \qquad \forall v \in V \setminus (S \cup T) \tag{1}$$

$$0 \le f_e \le c_e \qquad \forall e \in E \tag{2}$$

Key Results

In general (i.e., nonplanar) graphs, multiple sources and sinks can be reduced to the single-source, single-sink case by introducing an artificial source and sink and connecting them to all the sources and sinks, respectively, but this reduction does not preserve planarity. Using Orlin's algorithm for sparse graphs [21] leads to a running time of $O(n^2/\log n)$. For integer capacities less than U, one could instead use the algorithm of Goldberg and Rao [9], which leads to a running time of $O(n^{1.5} \log n \log U)$.

Maximum flow in planar graphs with multiple sources and sinks was first studied by Miller and Naor [19]. They gave a divide-and-conquer algorithm for the case where all the sinks and the sources are on the boundary of a single face. Plugging in the linear-time shortest-path algorithm of Henzinger et al. [12] yields a running time of $O(n \log n)$. Borradaile and Harutyunyan

have given an iterative algorithm with the same running time [2]. Miller and Naor also gave an algorithm for the case where the sources and the sinks reside on the boundaries of k different faces. Using the $O(n \log n)$ time single-source, single-sink maximum flow algorithm of Borradaile and Klein [3] yields a running time of $O(k^2 n \log^2 n)$. Miller and Naor show that, when it is known how much of the commodity is produced/consumed at each source and each sink, finding a consistent routing of flow that respects arc capacities can be reduced to negative-length shortest paths [19], which can be solved in planar graphs in $O(n \log^2 n / \log \log n)$ time [20].

Near-Linear Time Algorithm

Borradaile et al. gave the first $O(n \text{poly} \log n)$ time algorithm for the multiple-source, multiple-sink maximum flow problem in directed planar graphs. The approach uses pseudoflows [10, 14] (flows which may violate the balance constraints (1) in a limited way) and a divide-and-conquer scheme influenced by that of Johnson and Venkatesan [15] and that of Miller and Naor [19], using the separators introduced by Miller: a (triangulated) planar graph G can be separated by a simple cycle C of $O(\sqrt{n})$ vertices [18].

In each of the two subgraphs, a more general problem is solved in which, after the two recursive calls have been executed, within each of the two subgraphs, there is no residual path from any source to any sink nor from any source to C or from C to any sink. Then, since C is a separator, there is no residual path from any source to any sink in G, but, however, the balance constraints (1) may not be satisfied for vertices in C. The flow is then balanced among the vertices in C by augmenting the flow so that there is no residual path in G from a vertex with positive inflow to a vertex with positive outflow. The resulting flow can then be turned into a maximum flow in linear time.

The core of the algorithm is this final balancing procedure which involves a series of $|C| - 1$ max-flow computations in G. Since $|C|$ is $O(\sqrt{n})$, the challenge is carrying out all these max-flow computations in near-linear time. The procedure uses a succinct representation to keep track of the changes to the pseudoflow without explicitly storing the changes. The representation relies on the relationship between circulations in G and shortest paths in the dual, and the computations make use of an adaptation of Fakcharoenphol and Rao's efficient implementation of Dijkstra's algorithm [7]. The resulting running time to balance the flow is $O(n \log^2 n)$ time for an overall running time of $O(n \log^3 n)$ time for the original multiple-source, multiple-sink maximum flow problem.

Applications

Multiple-source, multiple-sink min-cut arises in several computer vision problems including image segmentation (or binary labeling) [11]. For the case of more than two labels, there is a powerful and effective heuristic [5] using a very large-neighborhood [1] local search; the inner loop consists of solving the two-label case.

Maximum matching in a bipartite planar graph reduces to multiple-source, multiple-sink maximum flow. Multiple-source, multiple-sink maximum flow can also be used for finding *orthogonal drawings of planar graphs with a minimum number of bends* [6] and *uniformly monotone subdivisions of polygons* [23].

Recommended Reading

1. Ahuja R, Ergun O, Orlin J, Punnen A (2002) A survey of very large scale neighborhood search techniques. Discret Appl Math 23:75–102. doi:10.1016/S0166-218X(01)00338-9
2. Borradaile G, Harutyunyan A (2013) Boundary-to-boundary flows in planar graphs. In: Proceedings of IWOCA, Rouen, pp 67–80. doi:10.1007/978-3-642-45278-9_7
3. Borradaile G, Klein PN (2009) An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. J ACM 56(2):9:1–9:30. doi:10.1145/1502793.1502798, http://doi.acm.org/10.1145/1502793.1502798
4. Boykov Y, Kolmogorov V (2004) An experimental comparison of min-cut/max-flow algorithms for en-

ergy minimization in vision. IEEE Trans Pattern Anal Mach Intell 26(9):1124–1137. doi:http://dx.doi.org/10.1109/TPAMI.2004.60
5. Boykov Y, Veksler O, Zabih R (2001) Efficient approximate energy minimization via graph cuts. IEEE Trans Pattern Anal Mach Intell 20(12):1222–1239. doi:10.1109/TPAMI.2003.1233908
6. Cornelsen S, Karrenbauer A (2012) Accelerated bend minimization. J Graph Algorithm Appl 16(3):635–650
7. Fakcharoenphol J, Rao S (2006) Planar graphs, negative weight edges, shortest paths, and near linear time. J Comput Syst Sci 72(5):868–889. doi:http://dx.doi.org/10.1016/j.jcss.2005.05.007
8. Geman S, Geman D (1984) Stochastic relaxation, Gibbs distributions, and the Bayesian relation of images. IEEE Trans Pattern Anal Mach Intell 6(6):721–742
9. Goldberg A, Rao S (1998) Beyond the flow decomposition barrier. J ACM 45(5):783–797. doi:10.1145/290179.290181
10. Goldberg A, Tarjan R (1988) A new approach to the maximum-flow problem. J ACM 35(4):921–940
11. Greig D, Porteous B, Seheult A (1989) Exact maximum a posteriori estimation for binary images. J R Stat Soc B 51(2):271–279
12. Henzinger MR, Klein PN, Rao S, Subramanian S (1997) Faster shortest-path algorithms for planar graphs. J Comput Syst Sci 55(1):3–23. doi:10.1145/195058.195092
13. Hochbaum DS (2001) An efficient algorithm for image segmentation, Markov random fields and related problems. J ACM 48(4):686–701
14. Hochbaum DS (2008) The pseudoflow algorithm: a new algorithm for the maximum-flow problem. Oper Res 56(4):992–1009
15. Johnson DB, Venkatesan SM (1983) Partition of planar flow networks. In: Proceeding of 24th FOCS, Tucson, pp 259–264. doi:10.1109/SFCS.1983.44
16. Kleinberg J, Tardos E (2002) Approximation algorithms for classification problems with pairwise relationships: metric labeling and markov random fields. J ACM 49(5):616–639. doi:10.1145/585265.585268, http://doi.acm.org/10.1145/585265.585268
17. Kohli P, Torr PHS (2007) Dynamic graph cuts for efficient inference in Markov random fields. IEEE Trans Pattern Anal Mach Intell 29(12):2079–2088
18. Miller GL (1986) Finding small simple cycle separators for 2-connected planar graphs. J Comput Syst Sci 32(3):265–279. doi:10.1016/0022-0000(86)90030-9
19. Miller GL, Naor J (1995) Flow in planar graphs with multiple sources and sinks. SIAM J Comput 24(5):1002–1017. doi:10.1137/S0097539789162997
20. Mozes S, Wulff-Nilsen C (2010) Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log\log n)$ time. In: Proceedings of 18th ESA, Liverpool, pp 206–217. doi:10.1007/978-3-642-15781-3_18
21. Orlin J (2013) Max flows in $O(nm)$ time, or better. In: Proceedings of STOC, Palo Alto, pp 765–774
22. Schmidt FR, Toeppe E, Cremers D (2009) Efficient planar graph cuts with applications in computer vision. In: Proceedings of CVPR, Miami, pp 351–356
23. Wei X, Joneja A, Mount DM (2012) Optimal uniformly monotone partitioning of polygons with holes. Comput Aided Des 44(12):1235–1252

Planar Maximum *s-t* Flow

Glencora Borradaile
Department of Computer Science, Brown University, Providence, RI, USA
School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA

Keywords

Maximum flow; Planar graphs

Years and Authors of Summarized Original Work

2006; Borradaile, Klein
2009; Borradaile, Klein
2010; Erickson

Problem Definition

Given a directed, planar graph $G = (V, E)$ with arc capacities $c : E \to \Re^+$, a source vertex s, and a sink vertex t, the goal is to find a flow assignment f_e for each arc $e \in E$ such that

$$\max \sum_{su : su \in E} f_{su}$$

$$\text{s.t.} \sum_{uv : uv \in E} f_{uv} - \sum_{vw : vw \in E} f_{vw} = 0 \qquad \forall v \in V \setminus \{s, t\} \tag{1}$$

$$0 \le f_e \le c_e \qquad \forall e \in E \tag{2}$$

Key Results

In the paper proposing the maximum flow problem in general graphs, Ford and Fulkerson [5] gave a generic method for computing a maximum flow: the augmenting-path algorithm. The algorithm is iterative: find a path P from the source to the sink such that capacity constraint (2) is loose for each arc on P (*residual*); increase the flow on each arc in P by a constant chosen so that at least one of the capacity constraints become tight; update the capacities of each arc, making note that the reverse of these arcs now have *residual capacity*; and repeat until there is no path from the source to the sink along which the flow can be augmented. By augmenting the flow along a path, the balance constraints (1) are always satisfied.

st-Planar Graphs

Ford and Fulkerson further showed that, in the case of planar graphs when the source and the sink are on a common face (st-planar graphs), by selecting the augmenting paths to be as far to the left as possible in each iteration (viewing s on the bottom and t on the top), each arc is saturated at most once, resulting in at most $|E|$ iterations [5]. In 1979, Itai and Shiloach showed that each iteration of this algorithm could be implemented in $O(\log n)$ time using a priority queue and gave a simple example showing that any implementation of this algorithm is capable of sorting n numbers [11]. In 1991, Hassin demonstrated that such a maximum st-flow could be derived from shortest-path distances in the planar dual G^* of G where capacities in G are interpreted as lengths in G^* [7]. Faster algorithms for computing shortest paths in planar graphs culminated in a linear-time algorithm for this case of maximum st-flow in planar graphs with s and t on a common face [9].

Undirected Planar Graphs

For undirected planar graphs, Reif gave an algorithm for computing the maximum st-flow where s and t need not be on a common face, by way of several shortest-path computations in the dual [19]. The algorithm finds a shortest path P in G^* from a vertex adjacent to the face corresponding to s to a vertex adjacent to the face corresponding to t. Reif proves that C only crosses P once; by finding the minimum separating cycle C_v through each vertex v of P, we will surely find C: C is the minimum of the cycles C_v. These cycles can be found in time $\log n$ times the time for one shortest-path computation via divide and conquer over the length of P. Hassin and Johnson show that the corresponding maximum flow can be computed within this framework by computing shortest-path distances between the nested cycles C_v [8]. The shortest-path algorithms of Henzinger et al. [9] or Klein [15] can be used to reimplement these algorithms in $O(n \log n)$ time. Italiano et al. [12] further improved this running time to $O(n \log \log n)$ by using an r-division to break the graph into sufficiently small pieces through which shortest paths can be efficiently computed.

If the capacities are all units, the maximum st-flow can be computed in linear time [1].

Directed Planar Graphs

Maximum st-flow in directed graphs is more general since the problem of maximum st-flow in an undirected graph can be converted to a directed problem by introducing two oppositely oriented arcs of equal capacity for each edge. Johnson and Venkatesan gave a divide-and-conquer algorithm that finds a flow of input value v in $O(n^{1.5} \log n)$ time [13]. The algorithm divides the graph using balanced separators, finding a flow in each side of value v. However, the flow on the $O(\sqrt{n})$-boundary edges of each subproblem might not be feasible. Each boundary edge is made feasible via an st-planar flow computation. Miller and Naor showed that finding a directed st-flow of value v could be reduced to computing shortest-path distances in a graph with positive and negatives lengths [17]. Here, v units of flow are routed (perhaps violating the capacity constraints) along any s-to-t path P. For those arcs whose capacity are violated, we must route the excess flow through the rest of the graph. This is a feasible circulation problem and can be solved using shortest-path distances in the dual graph, where lengths may be negative (representing the negative or violated capacities). Using an $O(n \text{ poly} \log n)$-time algorithm for computing shortest paths in a planar

graph with negative edge lengths [4, 16, 18] gives an $O(n \operatorname{poly} \log n \log C)$-time algorithm where C is the sum of the capacities.

If the capacities are all unit, the maximum st-flow can be computed in linear time [21].

Leftmost-Path Algorithm

Borradaile and Klein gave an augmenting-path, $O(n \log n)$-time algorithm for the maximum st-flow problem in directed planar graphs. The algorithm is a generalization of the algorithm for the st-planar case, augmenting flow repeatedly along the leftmost path from s to t. However, with s and t not on a common face, what leftmost is not clear. With the graph embedded such that t is on the external face and the clockwise cycles saturated, a leftmost path is well-defined and can be found with a left-first, depth-first search into t. Clockwise cycles can be initially saturated with a circulation defined by potentials on the faces given by shortest-path distances in the dual graph [14], and clockwise cycles remain saturated under leftmost augmentations. Borradaile and Klein, and Erickson improved the analysis [3] showed that under these conditions an arc and its reverse can be saturated at most once, resulting in at most $2n$ augmentations. Augmentations can be performed in $O(\log n)$ time using a dynamic tree data structure, resulting in an $O(n \log n)$ running time.

Applications

Maximum st-flow in directed planar graphs has applications to computer vision problems. Schmidt et al. [20] use it as a black box for image segmentation and Greig et al. [6] provide an example for smoothing noisy images.

Open Problems

Currently, maximum st-flow in undirected planar graphs can be computed more quickly than in directed. Can this gap be closed?

Experimental Results

Schmidt et al. [20] have implemented this algorithm and compared its performance on an image segmentation problem.

URLs to Code and Data Sets

Hoch and Wang have provided an open-source implementation of the algorithm [10]. Eisenstat has an implementation of the linear-time algorithm for unit-capacity graphs [2].

Cross-References

▸ Planar Maximum Flow – Multiple-Source Multiple-Sink Maximum Flow in Directed Planar Graphs

Recommended Reading

1. Coupry L (1997) A simple linear algorithm for the edge-disjoint (s,t)-paths problem in undirected planar graphs. Inf Process Lett 64:83–86
2. Eisenstat D (2013) Trickle: linear-time maximum flow in planar graphs with unit capacities (java). http://www.davideisenstat.com/trickle/
3. Erickson J (2010) Maximum flows and parametric shortest paths in planar graphs. In: 21st SODA, Austin, pp 794–804
4. Fakcharoenphol J, Rao S (2006) Planar graphs, negative weight edges, shortest paths, and near linear time. J Comput Syst Sci 72(5):868–889. doi:http://dx.doi.org/10.1016/j.jcss.2005.05.007
5. Ford C, Fulkerson D (1956) Maximal flow through a network. Can J Math 8:399–404
6. Greig D, Porteous B, Seheult A (1989) Exact maximum a posteriori estimation for binary images. J R Stat Soc B 51(2):271–279
7. Hassin R (1981) Maximum flow in (s,t) planar networks. Inf Process Lett 13(3):107
8. Hassin R, Johnson DB (1985) An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks. SIAM J Comput 14:612–624. doi:http://locus.siam.org/SICOMP/volume-14/art_0214045.html
9. Henzinger MR, Klein PN, Rao S, Subramanian S (1997) Faster shortest-path algorithms for planar graphs. J Comput Syst Sci 55(1):3–23. doi:10.1145/195058.195092
10. Hoch J, Wang J (2012) Max flow in a directed planar graph. https://github.com/jrshoch/msmsmaxflow

11. Itai A, Shiloach Y (1979) Maximum flow in planar networks. SIAM J Comput 8:135–150
12. Italiano GF, Nussbaum Y, Sankowski P, Wulff-Nilsen C (2011) Improved algorithms for min cut and max flow in undirected planar graphs. In: 43rd STOC, San Jose, pp 313–322
13. Johnson DB, Venkatesan SM (1983) Partition of planar flow networks. In: 24th FOCS, Tucson, pp 259–264. doi:10.1109/SFCS.1983.44
14. Khuller S, Naor J, Klein P (1993) The lattice structure of flow in planar graphs. SIAM J Discret Math 6(3):477–490. doi:10.1137/0406038
15. Klein PN (2005) Multiple-source shortest paths in planar graphs. In: 16th SODA, Vancouver, pp 146–155. doi:10.1145/1070454
16. Klein PN, Mozes S, Weimann O (2010) Shortest paths in directed planar graphs with negative lengths: a linear-space $O(n \log^2 n)$-time algorithm. TALG 6(2):1–18. doi:http://doi.acm.org/10.1145/1721837.1721846
17. Miller GL, Naor J (1995) Flow in planar graphs with multiple sources and sinks. SIAM J Comput 24(5):1002–1017. doi:10.1137/S0097539789162997
18. Mozes S, Wulff-Nilsen C (2010) Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log \log n)$ time. In: 18th ESA, Liverpool, pp 206–217
19. Reif J (1983) Minimum s-t cut of a planar undirected network in $O(n \log^2 n)$ time. SIAM J Comput 12:71–81. doi:http://locus.siam.org/SICOMP/volume-12/art_0212005.html
20. Schmidt FR, Toeppe E, Cremers D (2009) Efficient planar graph cuts with applications in computer vision. In: CVPR, Miami, pp 351–356
21. Weihe K (1994) Edge-disjoint (s,t)-paths in undirected planar graphs in linear time. In: Proceedings of the European symposium on algorithms, Edinburgh. LNCS, vol 855, pp 130–137

Planarisation and Crossing Minimisation

Markus Chimani
Faculty of Mathematics/Computer Science, Theoretical Computer Science, Osnabrück University, Osnabrück, Germany

Keywords

Approximations; Exact algorithms; Graph theory; Heuristics; Insertion problems; Mathematical programming

Years and Authors of Summarized Original Work

2005; Gutwenger, Mutzel, Weiskircher
2008; Chimani, Mutzel, Bomze

Problem Definition

Given a graph $G = (V, E)$, the *crossing number* $cr(G)$ of G is the smallest number of edge crossings possible for any drawing of G into the plane. Since its introduction in the mid-1940s, the crossing number problem has proved to be notoriously difficult. Even some of the oldest and seemingly simplest questions remain unanswered, despite the large amount of research.

Already the problem definition is more ambiguous than it may seem. We will usually only consider drawings where vertices are mapped to distinct points in the plane and edges to continuous non-self-intersecting curves between their end vertices. Any non-vertex point may only be contained in at most two edge curves, in which these curves have to meet transversally (i.e., *cross*). There are several different and specialized related crossing number variants; see [16] for a comprehensive annotated list.

A *planarization* of a nonplanar graph G is a planar graph obtained from G by drawing G into the plane and replacing the crossings by dummy vertices of degree 4. Observe that in other literature, the term *planarization* is also sometimes used to denote a (large) planar spanning subgraph of G.

Key Results

The (decision version of the) crossing number problem is NP-complete, even when all vertices have degree at most three [13] or when the removal of a single edge would give a planar graph [3]. There is, however, a fixed parameter tractable (FPT) algorithm to test in linear time

for a *constant* k (not part of the input), whether G allows a crossing number of at most k [15]. The dependency on k, however, is doubly exponential, and the algorithm is far from being applicable in practice. Most questions regarding the problem's approximability remain open; see below for details.

Planarization Algorithms

The practically strongest heuristic is the *planarization approach*, cf. Fig. 1: First, we seek a maximal planar subgraph (observe that finding a maximum planar subgraph is already NP-hard). In other words, we temporarily remove edges from G until it becomes planar. Then, we reinsert those edges with as few crossings as possible one after another. After each step, crossings are replaced by dummy vertices, such that we can consider a sequence of *edge insertion problems*: Given a planar graph H and an edge $e \notin E(H)$, let $H_e := H + e$. Since $cr(H_e)$ is still NP-hard to obtain, we ask for a crossing-minimum solution under the side-constraint that H is drawn planarly. An *embedding* is an equivalence class over planar drawings, based on the cyclic order of the edges around their incident vertices. For a fixed embedding of H, the insertion problem is trivially solvable via a breadth-first search in H's dual graph. However, the number of embeddings of H is exponential in general.

The seminal paper [12] shows that it is possible to find the best embedding *in linear time* using SPR-trees, or formally:

Theorem 1 *Let G be a planar graph and $v, w \in V(G)$ two vertices. We can find a planar embedding of G in $\mathcal{O}(|V(G)|)$ time, into which an edge $e = (v, w)$ can be inserted with the least possible number of edge crossings over all possible embeddings.*

To discuss this result, we first need to describe SPR-trees, which are used to decompose graphs into their triconnectivity structures. While their graph-theoretic foundation is based on Tutte [17], the data structure was first suggested by Di Battista and Tamassia [11] under the name *SPQR-tree*. Nowadays, we often drop the "Q" from the abbreviation, as the corresponding node type is not necessary, and use the following contemporary definition (see, e.g., [7]), illustrated in Fig. 2.

Definition 1 (SPR-tree) The *SPR-tree* $\mathcal{T}$ of a biconnected graph G is the unique smallest tree satisfying the following properties:

1. Each node ν in $\mathcal{T}$ holds a graph $S_\nu = (V_\nu, E_\nu)$, $V_\nu \subseteq V(G)$, called *skeleton*. Each edge of E_ν is either a *real* edge from $E(G)$ or a *virtual* edge $f = (u, v)$ where $\{u, v\}$ forms a 2-cut (a *split pair*) in G.
2. $\mathcal{T}$ has only three different node types with the following skeleton structures:
   - **S:** S_ν is a simple cycle – it represents a *serial* component.
   - **P:** S_ν consists of two vertices connected by multiple edges – a *parallel* component.
   - **R:** S_ν is a simple triconnected graph.
3. For every edge (ν, μ) in $\mathcal{T}$, S_ν (S_μ) contains a specific virtual edge e_μ (e_ν) which "represents" S_μ (S_ν, respectively). Both edges e_μ and e_ν connect the same vertices.
4. The original graph G can be obtained by recursively applying the following operation: For the edge (ν, μ) in $\mathcal{T}$, let e_μ, e_ν be the virtual edges as in (3) connecting the same vertices u, v. A merged graph $(S_\nu \cup S_\mu) - e_\mu - e_\nu$ is obtained by gluing the skeletons together at u, v and removing e_μ, e_ν.

There are several essential properties of any SPR-tree: First, it has linear size and can be constructed in linear time. Second, a simple triconnected graph – the skeleton of an R-node – allows only a unique embedding and its mirror; the embedding of an S-node skeleton is unique, and the possible embeddings of a P-node are precisely all cyclic permutations of its edges. Moreover, each embedding of G can be precisely described via the subembeddings of the skeletons.

In order to obtain Theorem 1, it is shown that we only need to consider the unique shortest path P in G's SPR-tree from any node whose skeleton contains v to any node whose skeleton contains w. We will specify an embedding for each skeleton S_μ of the nodes $\mu \in P$; the embedding of all other skeletons is irrelevant. Moreover, the

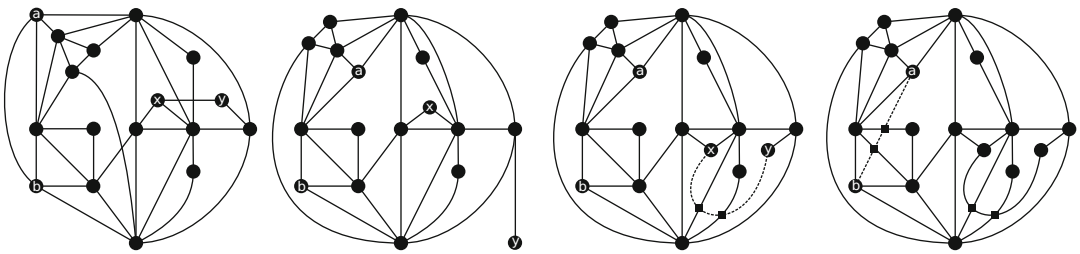
optimum embedding of each of the former skeletons can be chosen independent of the others: For each S_μ, we can specify a source s and a target t and ask for an embedding such that edge (s,t) can be inserted with the least possible number of edge crossings into planar S_μ: In the first (last) skeleton along P, the source (target) simply is v (w, respectively). In the other cases, the source (target) is the virtual edge corresponding to the predecessor (successor) along P. For simplicity, it may be helpful to consider a subdivision of such a virtual edge, such that each source and target can be represented by a vertex. If μ is an S-node, its skeleton has a unique embedding, (s,t) will require no crossings, and there is nothing to specify. If μ is a P-node, both the source and the target are virtual edges due to minimality of P; we pick any embedding where the two virtual edges appear consecutively, so that (s,t) again requires no crossings. Finally, if μ is an R-node, its skeleton allows only a unique embedding and its mirror and hence has a unique dual graph. We compute a shortest path between s and t via a simple breadth-first-search in the dual of S_μ as for the fixed embedding case.

Finally, traverse P and let μ, μ' be any two consecutive nodes. We want to establish that if the insertion path in S_μ enters its target virtual edge "from the left" (corresponding to some arbitrary predefined orientation of the virtual edges), it leaves the source virtual edge in $S_{\mu'}$ "to the right" or vice versa. If this is not already the case, it suffices to flip the embedding of $S_{\mu'}$ (we can ignore the case when μ' is an S-node). This establishes a suitable embedding of G.

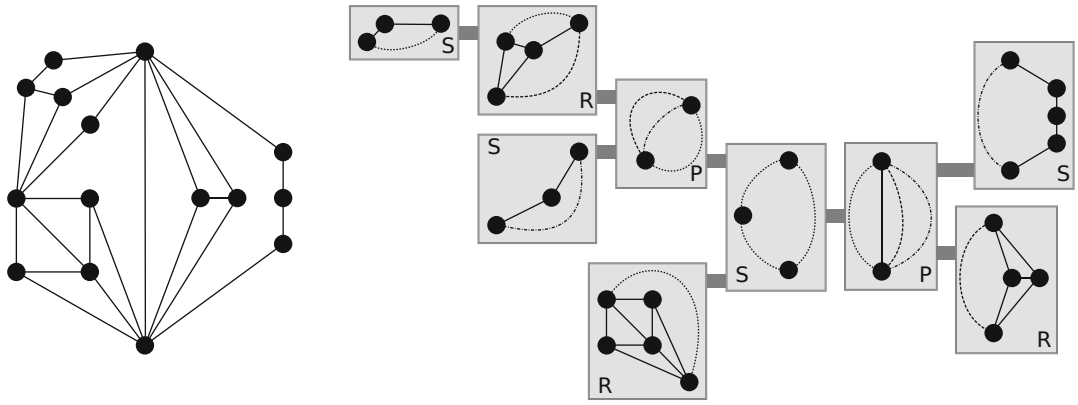
This algorithmic breakthrough has allowed the planarization heuristic to perform extraordinarily well in practice, both in terms of running time and of solution quality. The algorithm and its proofs also give rise to several strong pre- and postprocessing routines. Furthermore, it is pivotal to several later results such as insertion of stars [6] or insertion-based approximation algorithms [2, 7, 8]. The strongest of the latter considers the problem of inserting several edges F simultaneously into a planar graph G [7]: it can be shown that this multi-edge-insertion problem approximates $cr(G + F)$ [8], but is unfortunately itself already NP-hard. However, this insertion problem can in turn also be approximated. This approximation chain gives the currently only practically relevant approximation algorithm. In fact, it arguably gives the best running time vs. solution-quality trade-off among all known algorithms in practice [5].

Exact Approaches

In certain cases, a heuristic or approximate solution is not good enough, e.g., when the result is to be used as a base case in some formal graph-theoretic proof. There are exact approaches based on integer linear programs. The currently strongest one – constituting the second central paper of this entry – is able to solve typical "real-world graph drawing" instances (i.e., relatively



Planarisation and Crossing Minimisation, Fig. 1 The planarization heuristic: the first figure shows a graph drawn with the optimal number of three crossings. The planarization heuristic starts with a maximal (in the figure, in fact, maximum) planar subgraph (second figure) where the edges (x, y) and (a, b) are removed. Then, iteratively, we find optimal embeddings to reinsert these edges into the planarly drawn graph, simulating crossings by dummy vertices (shown as *squares*). Although the insertion problems are solved optimally, the result requires four crossings



Planarisation and Crossing Minimisation, Fig. 2 A graph (to the *left*) and its decomposition into triconnectivity structures, arranged within an SPR-tree. *Thick dashed* or *dotted edges* are virtual

sparse graphs with up to 80–100 vertices) to optimality in a couple of minutes [9]. The idea is to introduce binary indicator variables $x_{e,f}$ for each edge pair $e, f \in E$, which are 1 if and only if the two edges cross in the optimum solution. Minimizing the sum of these variables gives the desired objective function.

It remains to ensure that the variables are set correctly. To this end, we introduce *Kuratowski constraints*. The famous Kuratowski theorem states that planar graphs are characterized by the absence of subdivisions of certain small subgraphs (namely, K_5 and $K_{3,3}$). In other words, for each such subgraph K, we can require that at least one edge pair $e, f \in E(K)$ crosses. Unfortunately, there may be an exponential number of such subgraphs in G, and we also have to take care of Kuratowski subgraphs that only arise because certain other edge pairs cross (establishing a dummy vertex). Even when solving all these challenges, a further crucial problem remains: Consider a (presumably optimum) 0/1-assignment to the x-variables, satisfying all Kuratowski constraints. It is still NP-complete to decide whether this solution is at all feasible! Consider an edge e that is crossed by edges f and g. Our x-variables establish these crossings, but we do not know the order of f and g along e. Deciding whether any feasible order exists is what makes the problem still hard. There are two methods to solve the problem: We can subdivide each edge e sufficiently often and allow at most one crossing per edge segment. Alternatively we can introduce additional variables $y_{e,f,g}$ to explicitly describe a linear ordering of all edges crossing edge e, for all edges e. Of course, we have to modify the Kuratowski constraints accordingly.

While the second modeling approach is practically more efficient, we like to showcase the central ideas only with a simplified version of the first model here. Let U be any upper bound on the crossing number of G, e.g., found via the planarization heuristic described above. Clearly, any optimum solution will have at most U crossings on any edge. Therefore, let H be the graph obtained from G by subdividing each edge into U *edge segments*. Now, we solve the problem on H instead of G, where we can require that each segment is crossed at most once:

$$\min \sum_{e,f \in E(H), e \neq f} x_{\{e,f\}}, \text{ subject to} \quad (1)$$

$$\sum_{f \in E(H) \setminus \{e\}} x_{\{e,f\}} \leq 1 \qquad \forall e \in E(H) \quad (2)$$

$$x_{\{e,f\}} \in \{0,1\} \qquad \forall e, f \in E(H), e \neq f \quad (3)$$

This model minimizes the sum of the variables indicating a crossing between a pair of edge segments and allows at most one crossing per segment. It remains to discuss the Kuratowski

constraints to establish that only graph-theoretically feasible solutions are allowed. Let $\mathcal{X}_s$ denote the set of all binary solution vectors satisfying (2), and consider any solution vector $\bar{x} \in \mathcal{X}_s$. Furthermore, let $R(\bar{x})$ be the set of edge pairs $\{e, f\}$ for which $\bar{x}_{\{e,f\}} = 1$. Starting with H, we can *realize* $\bar{x}$ by, for each $\{e, f\} \in R(\bar{x})$, subdividing e and f and identifying the two new vertices. This vertex may be called a *dummy vertex*, representing the crossing. We obtain a final graph $H[\bar{x}]$ and let $\mathcal{K}(\bar{x})$ denote the set of all Kuratowski subdivisions in $H[\bar{x}]$. Intuitively, for any subdivision $K \in \mathcal{K}(\bar{x})$, we need to require at least one crossing on K, if the crossings $R(\bar{x})$ exist. Formally, this establishes the final constraint class:

$$\sum_{e,f \in K, e \neq f} x_{\{e,f\}} \geq 1 - \sum_{\{e,f\} \in R(\bar{x})} (1 - x_{\{e,f\}})$$
$$\forall \bar{x} \in \mathcal{X}_s, K \in \mathcal{K}(\bar{x}) \qquad (4)$$

Independent on whether we use the just described subdivision-based model or the stronger one based on linear orderings, the obtained ILP models are much too large to solve directly. We require both special separation and column-generation routines, in order to produce the actually necessary constraints and variables on the fly. The approaches' practical applicability is furthermore only possible due to the strong heuristics described above (which often give optimum upper bounds early on), heavy preprocessing [4], and efficient planarity testing routines.

Open Problems

The area of crossing numbers is filled with interesting open questions. Let us pinpoint two of them:

The original question, as stated in 1944 by Pál Turán, was for the crossing number of complete bipartite graphs. We still do not know the answer for this graph class, nor do we for complete graphs. While we have upper bounds, which are conjectured to be optimal, we are stuck with partial proofs and positive results for small graphs.

We know that crossing number is APX-hard [1], i.e., there cannot be a polynomial approximation scheme. However, even for graphs with bounded maximum degree, the best approximation ratios are only slightly sublinear in $|V|$ [10] or dependent on parameters like the graph's genus [14] or the number of edges required to remove in order to become planar [7]. Does there exist a constant factor approximation to the crossing number problem? At least for graphs with bounded degree?

URLs to Code and Data Sets

The free (GPL) Open Graph Drawing Framework (OGDF) contains implementations of the strongest planarization heuristics and the exact algorithms: http://www.ogdf.net.

A web front-end to the exact crossing minimizer is freely available at http://crossings.uos.de.

Recommended Reading

1. Cabello S (2013) Hardness of approximation for crossing number. Discret Comput Geom 49(2):348–358
2. Cabello S, Mohar B (2011) Crossing number and weighted crossing number of near-planar graphs. Algorithmica 60(3):484–504
3. Cabello S, Mohar B (2013) Adding one edge to planar graphs makes crossing number and 1-planarity hard. SIAM J Comput 42(5):1803–1829
4. Chimani M, Gutwenger C (2009) Non-planar core reduction of graphs. Discret Math 309(7):1838–1855
5. Chimani M, Gutwenger C (2012) Advances in the planarization method: effective multiple edge insertions. J Graph Algorithms Appl 16(3):729–757
6. Chimani M, Gutwenger C, Mutzel P, Wolf C (2009) Inserting a vertex into a planar graph. In: SODA, New York, pp 375–383
7. Chimani M, Hlinený P (2011) A tighter insertion-based approximation of the crossing number. In: Aceto L, Henzinger M, Sgall J (eds) ICALP (1), Zurich. Lecture notes in computer science, vol 6755. Springer, pp 122–134
8. Chimani M, Hlinený P, Mutzel P (2012) Vertex insertion approximates the crossing number of apex graphs. Eur J Comb 33(3):326–335

9. Chimani M, Mutzel P, Bomze I (2008) A new approach to exact crossing minimization. In: Halperin D, Mehlhorn K (eds) ESA, Karlsruhe. Lecture notes in computer science, vol 5193. Springer, pp 284–296
10. Chuzhoy J (2011) An algorithm for the graph crossing number problem. In: Proceedings of the 43rd annual ACM symposium on theory of computing (STOC'11), San Jose. ACM, pp 303–312
11. Di Battista G, Tamassia R (1996) On-line planarity testing. SIAM J Comput 25:956–997
12. Gutwenger C, Mutzel P, Weiskircher R (2005) Inserting an edge into a planar graph. Algorithmica 41:289–308
13. Hliněný P (2006) Crossing number is hard for cubic graphs. J Comb Theory Ser B 96:455–471
14. Hliněný P, Chimani M (2010) Approximating the crossing number of graphs embeddable in any orientable surface. In: Proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms (SODA '10), Austin. Society for Industrial and Applied Mathematics, pp 918–927
15. Kawarabayashi K-i, Reed B (2007) Computing crossing number in linear time. In: Proceedings of the thirty-ninth annual ACM symposium on theory of computing (STOC '07), San Diego. ACM, pp 382–390
16. Schaefer M (2013) The graph crossing number and its variants: a survey. Electron J Comb Dyn Sur #21
17. Tutte WT (1966) Connectivity in graphs. Mathematical expositions, vol 15. University of Toronto Press, Toronto

Planarity Testing

Glencora Borradaile
Department of Computer Science, Brown University, Providence, RI, USA
School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA

Keywords

Planar embedding; Planarity testing

Years and Authors of Summarized Original Work

1976; Booth, Lueker

Problem Definition

The problem is to determine whether or not the input graph G is planar. The definition pertinent to planarity-testing algorithms is: G is planar if there is an *embedding* of G into the plane (vertices of G are mapped to distinct points and edges of G are mapped to curves between their respective endpoints) such that edges do not cross. Algorithms that test the planarity of a graph can be modified to obtain such an embedding of the graph.

Key Results

Theorem 1 *There is an algorithm that given a graph G with n vertices, determines whether or not G is planar in O(n) time.*

The first linear-time algorithm was obtained by Hopcroft and Tarjan [5] by analyzing an iterative version of a recursive algorithm suggested by Auslander and Parter [1] and corrected by Goldstein [4]. The algorithm is based on the observation that a connected graph is planar if and only if all its biconnected components are planar. The recursive algorithm works with each biconnected component in turn: find a separating cycle C and partition the edges of G not in C; define a component of the partition as consisting of edges connected by a path in G that does not use an edge of C; and, recursively consider each cyclic component of the partition. If each component of the partition is planar and the components can be combined with C to give a planar graph, then G is planar.

Another method for determining planarity was suggested by Lempel, Even, and Cederbaum [6]. The algorithm starts with embedding a single vertex and the edges adjacent to this vertex. It then considers a vertex adjacent to one of these edges. For correctness, the vertices must be considered in a particular order. This algorithm was first implemented in $O(n)$ time by Booth and Lueker [2] using an efficient implementation of the PQ-trees data structure. Simpler implementations of this algorithm have been given by Boyer and Myrvold [3] and Shih and Hsu [8].

Tutte gave an algebraic method for giving a *straight-line embedding* of a graph that, if the input graph is 3-connected and planar, is guaranteed to generate a planar embedding. The key idea is to fix the vertices of one face of the graph to be the corners of a convex polygon and then embed every other vertex as the geometric average of its neighbors.

Applications

Planarity testing has applications to computer-aided circuit design and VLSI layout by determining whether a given network can be realized in the plane.

URL to Code

LEDA has an efficient implementation of the Hopcroft and Tarjan planarity testing algorithm [7]: http://www.algorithmic-solutions.info/leda_guide/graph_algorithms/planar_kuratowski.html

Cross-References

▸ Fully Dynamic Planarity Testing

Recommended Reading

1. Auslander L, Parter SV (1961) On imbedding graphs in the plane. J Math Mech 10:517–523
2. Booth KS, Lueker GS (1976) Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J Comput Syst Sci 13:335–379
3. Boyer J, Myrvold W (1999) Stop minding your P's and Q's: a simplified O(n) planar embedding algorithm. In: SODA'99: proceedings of the tenth annual ACM-SIAM symposium on discrete algorithms, Philadelphia. Society for Industrial and Applied Mathematics, pp 140–146
4. Goldstein AJ (1963) An efficient and constructive algorithm for testing whether a graph can be embedded in the plane. In: Graph and combinatorics conference
5. Hopcroft J, Tarjan R (1974) Efficient planarity testing. J ACM 21:549–568
6. Lempel A, Even S, Cederbaum I (1967) An algorithm for planarity testing of graphs. In: Rosentiehl P (ed) Theory of graphs: international symposium. Gordon and Breach, New York, pp 215–232
7. Mehlhorn K, Mutzel P, Näher S (1993) An implementation of the hopcroft and tarjan planarity test. Technical report, MPI-I-93-151, Saarbrücken
8. Shih W-K, Hsu W-L (1999) A newplanarity test. Theor Comput Sci 223:179–191

Point Location

Marcel Roeloffzen
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Keywords

Jump and walk; Nearest neighbor; Point location; Trapezoidal decomposition; Triangulation

Years and Authors of Summarized Original Work

1983; Kirkpatrick
1991; Seidel
2009; Haran, Halperin

Problem Definition

Point location is a well-studied problem in computational geometry with many applications in geometric information systems and computer-aided design. In general terms, the problem is to find which element of a given object contains a given query point. More precisely, we are given a subdivision $\mathcal{S}$ of a metric space, usually the Euclidean plane. The goal is then to preprocess $\mathcal{S}$ so that for a query point p we can determine which region of the subdivision contains p. There are many variants of the problem, with different constraints on the subdivision. The most common

variant and the focus of this overview is planar point location, where $\mathcal{S}$ is a polygonal subdivision of the Euclidean plane. In fact, for several data structures, we assume that the input is a triangulation. (Note that a polygonal subdivision can be triangulated in linear time [1].) As for most query data structures, the efficiency of a point location structure is measured by its space requirement, the time needed to preprocess the input, and the query time. The efficiency of the algorithms and data structures described here will be expressed as a function of n, the number of vertices in the subdivision.

Key Results

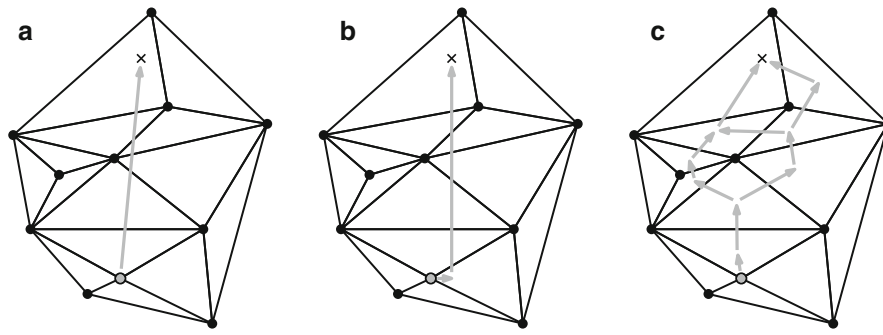
Most solutions for the point location problem build upon one (or a combination) of three basic ideas: walking in a triangulation, a trapezoidal decomposition, or a hierarchical triangulation. The three methods provide a basic trade-off between space usage and query time and are described in more detail below.

Walking in a Triangulation

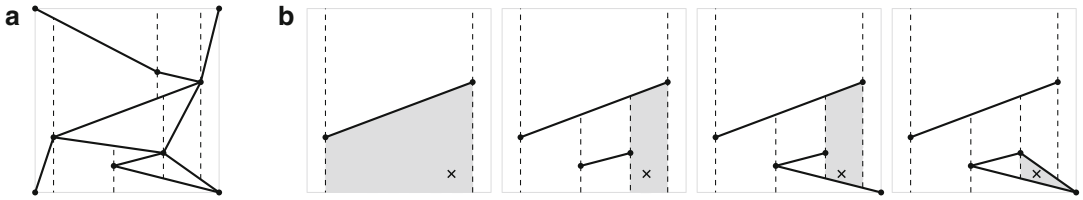
This approach is the simplest of the three and requires no additional storage or preprocessing if the input is provided in a suitable format in which it takes constant time to access the neighbors of a triangle. For each query, we start in some triangle or vertex of the triangulation and walk to the query point by traversing the triangulation, walking from the current triangle to a neighbor in each step. The triangle to visit next is determined by a walking strategy. Devillers et al. [3] describe three walking strategies that are described below and illustrated in Fig. 1.

- *Straight-line walk.* Starting at a vertex of the triangulation we walk along a straight line toward the query point.
- *Orthogonal walk.* Instead of walking along a straight line, we first walk parallel to the x-axis and then parallel to the y-axis.
- *Visibility walk.* For each triangle visited, we pick an edge of the triangle (at random or in some specific order) and test if its supporting line separates the query point from the triangle. If this is the case, then we continue our walk by crossing that edge into a new triangle.

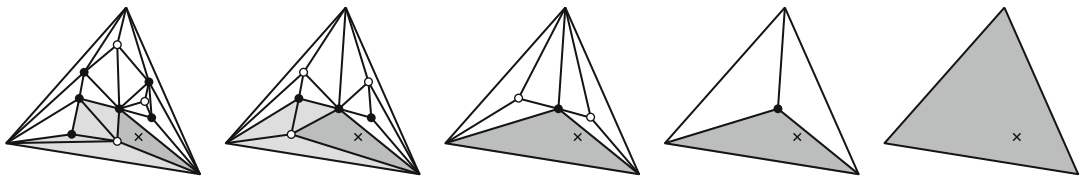
The worst-case behavior of each of the walking strategies is very bad as there are triangulations where each walk visits $\Omega(n)$ triangles in expectation given a random starting point and query point. In fact, the visibility walk is not guaranteed to reach the query point if edges are picked in a fixed order and the randomized version may require exponential time [3]. However, this seems to require many long and thin triangles, which do not occur in most practical applications, where the subdivision is often closer to a Delaunay triangulation. Experiments on Delaunay triangulations of random point sets suggest that the methods are comparable in total query time, though they provide a trade-off between the number of triangles visited and the time spent per triangle.



Point Location, Fig. 1 The paths traversed by (**a**) a straight-line walk, (**b**) an orthogonal walk, and (**c**) a visibility walk. Note that any path following the *arrows* is possible in the visibility walk



Point Location, Fig. 2 (**a**) Part of a trapezoidal decomposition with (**b**) a sequence of trapezoids visited by a query



Point Location, Fig. 3 The triangulations of the hierarchy from left to right. Triangles containing the query point are marked with *dark gray* and other triangles inspected with *light gray*

That is, the straight-line walk visits the fewest triangles, but needs most time to determine which triangle to visit next, whereas the visibility walk visits the most triangles, but spends the least time per triangle.

Trapezoidal Decomposition

The first data structure for point location to achieve $O(\log n)$ query time for point was provided by Dobkin and Lipton [6]. The structure is created by cutting the subdivision into slices using vertical lines through each of its vertices. Point location can then be done using two binary searches, first on the slices and then within a slice. This creates $O(n^2)$ trapezoids, but Sarnak and Tarjan [11] show we don't have to explicitly store all of them and instead only need $O(n)$ space and $O(n \log n)$ time to store them implicitly in a search structure.

A subdivision into trapezoids can also be created in a more careful way so that only $O(n)$ trapezoids are created. For a polygonal subdivision, its *trapezoidal decomposition* is defined as the result of shooting vertical rays upward and downward from each vertex of the subdivision until they hit an edge of the subdivision – if they do not hit such an edge, they extend to infinity (see also Fig. 2). A trapezoidal decomposition can be constructed by incrementally adding the edges of the subdivision [10]. Each insertion of an edge can interrupt some rays and introduces two new ones, namely, the rays from the endpoints of the edge. Seidel [12] showed that the history of this process can be recorded in a directed acyclic graph, which can be used for point location. Using randomized incremental construction, that is, adding the edges in random order, the longest path in the graph has an expected length of $O(\log n)$. Building the structure itself takes $O(n \log n)$ expected time and $O(n)$ expected space. For a guaranteed query time of $O(\log n)$, we can find the longest path and reconstruct if needed resulting in $O(n \log n)$ expected preprocessing time and $O(\log n)$ guaranteed query time [13].

Hierarchical Triangulation

Point location can also be done using a so-called hierarchical triangulation. We start with a triangulation of the subdivision and in each level of the hierarchy, we remove a subset of the points and compute a triangulation of the remainder. Each triangle of this new triangulation then stores a list of triangles from the previous triangulation that intersect it as illustrated in Fig. 3. This process is repeated until only one triangle remains. (Here, we assume that the outer face of the input triangulation is a triangle and these vertices are never removed.) To query for a point p, we traverse this hierarchy of triangles starting

at the top level consisting of only one triangle. At each level, we know the triangle T that contains p and find the triangle T' from the previous level that contains p by a linear search on the triangles that intersect T (see Fig. 3 from right to left).

The query time depends on the number of levels and the number of intersections each triangle has with the previous level. Kirkpatrick [9] showed that it is possible to find a constant size set of points, such that its removal creates triangles that each intersect a constant number of triangles from the previous level. He also shows that such a set of points can be found in $O(n)$ time. By picking and removing points from each level in this way, we create at most $O(\log n)$ levels and each triangle will intersect at most $O(1)$ triangles from the previous level. As a result, a point location query takes $O(\log n)$ time in total, while the structure requires $O(n)$ space.

Hybrid and Refined Approaches

The walking strategy requires no additional memory and is fast on small instances, but has very poor performance on larger instances, both in theory and in practice. To make the walking approach more feasible for larger inputs, several structures have been proposed that combine walking strategies with other methods to obtain fast query times with low overhead costs in terms of space and preprocessing time.

Delaunay Hierarchy. Hierarchical triangulations can be combined with walking algorithms to reduce the number of levels in the hierarchy. Finding the correct triangle in the next level of the hierarchy is done using a walking algorithm as opposed to a linear search. Devillers [2,3] showed that this results in a very fast query time without requiring a lot of preprocessing time or space.

Jump and Walk. In most walking strategies, the starting point is chosen at random. In the jump-and-walk approach, we use a set S of several starting points and start our walk from the one that is nearest to the query point. The set S can either be chosen at random for each new query [5] or picked more carefully and stored in a data structure for nearest-neighbor searches [4]. Depending on the size of S and the complexity of the search structure, query times between $O(\sqrt[3]{n})$ and $O(\log n)$ can be achieved.

Experimental Results

Several solutions have been implemented in CGAL, the Computational Geometry Algorithm Library. Haran and Halperin [7] compared several of the implementations from CGAL. Their results show that the various methods provide a trade-off between how much memory and preprocessing time is used and the resulting query time. Overall, they conclude that a jump-and-walk algorithm performs well, if the set of potential starting points is carefully chosen and stored in an efficient search structure. Recently, the CGAL variant of the trapezoidal decomposition approach has received a major overhaul [8]. Unlike some of the other variants, this implementation guarantees a $O(\log n)$ query time, while experiments show that the implementation is still competitive with other approaches.

Cross-References

- Triangulation Data Structures
- Voronoi Diagrams and Delaunay Triangulations

Recommended Reading

1. Chazelle B (1991) Triangulation a simple polygon in linear time. Discret Comput Geom 6(1):485–524
2. Devillers O (2002) The Delaunay hierarchy. Int J Found Comput Sci 13:163–181
3. Devillers O, Pion S, Teillaud M (2002) Walking in a triangulation. Int J Found Comput Sci 13:181–199
4. Devroye L, Lemaire C, Moreau JM (2004) Expected time analysis for Delaunay point location. Comput Geom Theory Appl 29:61–89
5. Devroye L, Mücke E, Zhu B (1998) A note on point location in Delaunay triangulations of random points. Algorithmica (Spec Issue Aver Case Anal Algorithms) 22(4):477–482
6. Dobkin D, Lipton RJ (1976) Multidimensional searching problems. SIAM J Comput 5(2):181–186

7. Haran I, Halperin D (2009) An experimental study of point location in planar arrangements in CGAL. J Exp Algorithm 13:Article 3
8. Hemmer M, Kleinbort M, Halperin D (2012) Improved implementation of point location in general two-dimensional subdivisions. In: Proceedings of the 20th European symposium on algorithms (ESA), Ljubljana, pp 611–623
9. Kirkpatrick D (1983) Optimal search in planer subdivisions. SIAM J Comput 12(1):28–35
10. Mulmuley K (1990) A fast planar partition algorithm, I. J Symb Comput 10(3):253–280
11. Sarnak N, Tarjan RE (1986) Planar point location using persistent search trees. Commun ACM 29(7):669–679
12. Seidel R (1991) A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. Comput Geom: Theory Appl 1(1):51–64
13. Seidel R, Adamy U (2000) On the exact worst case query complexity of planar point location. J Algorithms 37:189–217

Point Pattern Matching

Veli Mäkinen and Esko Ukkonen
Department of Computer Science, Helsinki Institute for Information Technology (HIIT), University of Helsinki, Helsinki, Finland

Keywords

Geometric alignment; Geometric matching; Largest common point set; Point set matching

Years and Authors of Summarized Original Work

2003; Ukkonen, Lemström, Mäkinen

Problem Definition

Let $\mathbb{R}$ denote the set of reals and $\mathbb{R}^d$ the d-dimensional real space. A finite subset of $\mathbb{R}^d$ is called a *point set*. The set of all point sets (subsets of $\mathbb{R}^d$) is denoted $\mathcal{P}(\mathbb{R}^d)$.

Point pattern matching problems ask for finding similarities between point sets under some transformations. In the basic set–up a *target* point set $T \subset \mathbb{R}^d$ and a *pattern* point set (*point pattern*) $P \subset \mathbb{R}^d$ are given, and the problem is to locate a subset I of T (if it exists) such that P *matches* I. Matching here means that P becomes exactly or approximately equal to I when a *transformation* from a given set $\mathcal{F}$ of transformations is applied on P.

Set $\mathcal{F}$ can be, for example, the set of all *translations* (a constant vector added to each point in P), or all compositions of translations and *rotations* (after a translation, each point is rotated with respect to a common origin; this preserves the distances and is also called a *rigid movement*), or all compositions of translations, rotations, and *scales* (after translating and rotating, distances to the common origin are multiplied by a constant).

The problem variant with exact matching, called the *Exact Point Pattern Matching (EPPM)* problem, requires that $f(P) = I$ for some $f \in \mathcal{F}$. In other words, the EPPM problem is to decide whether or not there is an allowed transformation f such that $f(P) \subset T$. For example, if $\mathcal{F}$ is the set of translations, the problem is simply to decide whether $P + t \subset T$ for some $t \in \mathbb{R}^d$.

Approximate matching is a better model of many situations that arise in practice. Then the quality of the matching between $f(P)$ and I is controlled using a *threshold parameter* $\varepsilon \geq 0$ and a *distance function* δ: $(\mathcal{P}(\mathbb{R}^d), \mathcal{P}(\mathbb{R}^d)) \rightarrow \mathbb{R}$ for measuring distances between point sets. Given $\varepsilon \geq 0$, the *Approximate Point Pattern Matching (APPM)* problem is to determine whether there is a subset $I \subset T$ and a transformation $f \in \mathcal{F}$ such that $\delta(f(P), I) \leq \varepsilon$.

The choice of the distance function δ is another source of diversity in the problem statement. A variant requires that there is a *one-to-one* mapping between $f(P)$ and I, and each point p of $f(P)$ is *ε-close* to its one-to-one counterpart p^* in I, that is, $|p - p^*| \leq \varepsilon$. A commonly studied relaxed version uses matching under a *many-to-one* mapping: it is only required that each point of $f(P)$ has *some* point of I that is *ε-close*; this distance is also known

as the *directed Hausdorff distance*. Still more variants come from the choice of the *norm* $|\cdot|$ to measure the distance between points.

Another form of approximation is obtained by allowing a minimum amount of unmatched points in P: The *Largest Common Point Set (LCP)* problem asks for the largest $I \subset T$ such that $I \subset f(P)$ for some $f \in \mathcal{F}$. In the *Largest Approximately Common Point Set (LACP)* problem each point $p^* \in I$ must occur *ε-close* to a point $p \in f(P)$.

Finally, a problem closely related to point pattern matching is to evaluate for point sets A and B their smallest distance $\min_{f\in\mathcal{F}} \delta(f(A), B)$ under transformations $\mathcal{F}$ or to test if this distance is $\leq \varepsilon$. This problem is called the *distance evaluation problem*.

Key Results

A folk theorem is a voting algorithm to solve EPPM under translations in $O(|P||T| \log(|T||P|))$ time: Collect all translations mapping each point of P to each point of T, sort the set, and report the translation getting most votes. If some translation gets $|P|$ votes, then a subset I such $f(P) = I$ is found. With some care in organizing the sorting, one can achieve $O(|P||T| \log |P|)$ time [13].

The voting algorithm also solves the LCP problem under translations. A faster algorithm specific to EPPM is as follows: Let $p_1, p_2, \cdots p_m$ and $t_1, t_2, \cdots t_n$ be the lists of pattern and target points, respectively, *lexicographicly ordered* according to their d-dimensional coordinate values. Consider the translation $f_{i_1} = t_{i_1} - p_1$, for any $1 \leq i_1 \leq n$. One can scan the target points in the lexicographic order to find a point t_{i_2} such that $p_2 + f_{i_1} = t_{i_2}$. If such is found, one can continue scanning from t_{i_2+1} on to find t_{i_3} such that $p_3 + f_{i_1} = t_{i_3}$. This process is continued until a translated point of P does not occur in T or until a translated occurrence of the entire P is found. Careful implementation of this idea leads to the following result showing that the time bound of the naive string matching algorithm is possible also for the exact point pattern matching under translations.

Theorem 1 (Ukkonen et al. 2003 [13]) *The EPPM problem under translations for point pattern P and target T can be solved in $O(mn)$ time and $O(n)$ space where $m = |P| \leq |T| = n$.*

Quadratic running times are probably the best one can achieve for PPM algorithms:

Theorem 2 (Clifford et al. 2006 [10]) *The LCP problem under translations is 3SUM-hard.*

This means that an $o(|P||T|)$ time algorithm for LCP would yield an $o(n^2)$ algorithm for the `3SUM` problem, where $|T| = n$ and $|P| = \Theta(n)$. The `3SUM` problem asks, given n numbers, whether there are three numbers a, b, and c among them such that $a + b + c = 0$; finding a sub-quadratic algorithm for `3SUM` would be a surprise [5]. For a more in-depth combinatorial characterization of the geometric properties of the EPPM problem, see [7].

For the distance evaluation problems there are plethora of results. An excellent survey of the key results until 1999 is by Alt and Guibas [2]. As an example, consider in the 2-dimensional case how one can decide in $O(n \log n)$ time whether there is a transformation f composed of translation, rotation and scale, such that $f(A) = B$, where $A, B \subset \mathbb{R}^2$ and $n = |A| = |B|$: The idea is to convert A and B into an invariant form such that one can easily check their congruence under the transformations. First, scale is taken into account by scaling A to have the same *diameter* as B (in $O(n \log n)$ time). If A and B are congruent, then they must have the same *centroids* (which can be computed $O(n)$ time). Consider rotating a line from the centroid and listing the angles and distances to other points in the order they are met during the rotation. Having done this (in $O(n \log n)$ time) on both A and B, the lists of angles and distances should be *cyclic shifts* of each other; the list L_A of A occurs as a substring in $L_B L_B$, where L_B is the list of B. This latter step can be done in $O(n)$ time using any linear time exact string matching algorithm. One obtains the following result.

Theorem 3 (Atkinson 1987 [4]) *It is possible to decide in $O(n \log n)$ time whether there is*

a transformation f composed of translation, rotation and scale, such that $f(A) = B$, where $A, B \subset \mathbb{R}^2$ and $|A| = |B| = n$.

Approximate variant of the above problem is much harder. Denote by $f(A) =^{\varepsilon} B$ the *directed approximate congruence* of point sets A and B, meaning that there is a one-to-one mapping from $f(A)$ to B such that for each point in $f(A)$ its image in B is ε-close. The following result demonstrates the added difficulty.

Theorem 4 (Alt et al. 1988 [3]) *It is possible to decide in $O(n^6)$ time whether there is a translation f such that $f(A) =^{\varepsilon} B$, where $A, B \subset \mathbb{R}^2$ and $|A| = |B| = n$. The same algorithm solves the corresponding LACP problem for point pattern P and target T under the one-to-one matching condition in $O((mn)^3)$ time, where $m = |P| \leq |T| = n$.*

To get an idea of the techniques to achieve the $O((mn)^3)$ time algorithm for LACP, consider first the one-dimensional version, i.e., let $P, T \subset \mathbb{R}$. Observe, that if there is a translation f' such that $f'(P) =^{\varepsilon} T$, then there is a translation f such that $f(P) =^{\varepsilon} T$ *and* a point $p \in P$ that is mapped *exactly* at ε-distance of a point $t \in T$. This lets one concentrate on these $2mn$ *representative translations*. Consider these translations sorted from left to right. Denote the left-most translation by f. Create a bipartite graph, whose nodes are the points in P and in T on the different parties. There is an edge between $p \in P$ and $t \in T$ if and only if $f(p)$ is ε-close to t. Finding a maximum matching in this graph tells the size of the largest approximately common point set after applying the translation f. One can repeat this on each representative translation to find the overall largest common point set. When the representative translations are considered from left to right, the bipartite graph instances are such that one can compute the maximum matchings greedily at each translation in time $O(|P|)$ [6]. Hence, the algorithm solves the one-dimensional LACP problem under translations and one-to-one matching condition in time $O(m^2 n)$, where $m = |P| \leq |T| = n$.

In the two-dimensional case, the set of representative translations is more implicitly defined: In short, the mapping of each point $p \in P$ ε-close to each point $t \in T$, gives mn circles. The boundary of each such circle is partitioned into intervals such that the end points of these intervals can be chosen as representative translations. There are $O((mn)^2)$ such representative translations. As in the one-dimensional case, each representative translation defines a bipartite graph. Once the representative translations along a circle are processed e.g., counterclockwise, the bipartite graph changes only by one edge at a time. This allows an $O(mn)$ time update for the maximum matching at each representative translation yielding an overall $O((mn)^3)$ time algorithm [3].

More efficient algorithms for variants of this problem have been developed by Efrat, Itai, and Katz [11], as by-products of more efficient bipartite matching algorithms for points on a plane. Their main result is the following:

Theorem 5 (Efrat et al. 2001 [11]) *It is possible to decide in $O(n^5 \log n)$ time whether there is a translation f such that $f(A) =^{\varepsilon} B$, where $A, B \subset \mathbb{R}^2$ and $|A| = |B| = n$.*

The problem becomes somewhat easier when the one-to-one matching condition is relaxed; one-to-one condition seems to necessitate the use of bipartite matching in one form or another. Without the condition, one can match the points independently of each other. This gives many tools to preprocess and manipulate the point sets during the algorithm using dynamic geometric data structures. Such techniques are exploited e.g., in the following result.

Theorem 6 (Chew and Kedem 1992 [8]) *The LACP problem under translations and using directed Hausdorff distance and the L_1 norm, can be solved in $O(mn \log n)$ time, where $P, T \subset \mathbb{R}^2$ and $m = |P| \leq |T| = n$. The distance evaluation problem for directed Hausdorff distance can be solved in $O(n^2 \log^2 n)$ time.*

Most algorithms revisited here have relatively high running times. To obtain faster algorithms, it seems that randomization and approximation

techniques are necessary. See [9] for a comprehensive summary of the main achievements in that line of development.

Finally, note that the linear transformations considered here are not always enough to model a real-world problem–even when approximate congruence is allowed. Sometimes the proper transformation between two point sets (or between their subsets) is non-linear, without an easily parametrizable representation. Unfortunately, the formulations trying to capture such non-uniformness have been proven NP-hard [1] or even NP-hard to approximate within any constant factor [12].

Applications

Point pattern matching is a fundamental problem that naturally arises in many application domains such as computer vision, pattern recognition, image retrieval, music information retrieval, bioinformatics, dendrochronology, and many others.

Cross-References

- Assignment Problem
- Maximum Cardinality Stable Matchings
- Multidimensional String Matching
- String Matching

Recommended Reading

1. Akutsu T, Kanaya K, Ohyama A, Fujiyama A (2003) Point matching under non-uniform distortions. Discret Appl Math 127:5–21
2. Alt H, Guibas L (1999) Discrete geometric shapes: matching, interpolation, and approximation. In: Sack JR, Urrutia J (eds) Handbook of computational geometry. Elsevier, North-Holland, pp 121–153
3. Alt H, Mehlhorn K, Wagener H, Welzl E (1988) Congruence, similarity and symmetries of geometric objects. Discret Comput Geom 3:237–256
4. Atkinson MD (1997) An optimal algorithm for geometric congruence. J Algorithm 8:159–172
5. Barequet G, Har-Peled S (2001) Polygon containment and translational min-hausdorff-distance between segment sets are 3SUM-hard. Int J Comput Geom Appl 11(4):465–474
6. Böcker S, Mäkinen V (2005) Maximum line-pair stabbing problem and its variations. In: Proceedings of the 21st European workshop on computational geometry (EWCG'05). Technische Universität Eindhoven, pp 183–186
7. Brass P, Pach J (2005) Problems and results on geometric patterns. In: Avis D et al (eds) Graph theory and combinatorial optimization. Springer, New York, pp 17–36
8. Chew LP, Kedem K (1992) Improvements on geometric pattern matching problems. In: Proceedings of the Scandinavian workshop algorithm theory (SWAT). LNCS, vol 621. Springer, Berlin, pp 318–325
9. Choi V, Goyal N (2006) An efficient approximation algorithm for point pattern matching under noise. In: Proceedings of the 7th Latin American symposium on theoretical informatics (LATIN 2006). LNCS, vol 3882. Springer, Berlin, pp 298–310
10. Clifford R, Christodoukalis M, Crawford T, Meredith D, Wiggins G (2006) A fast, randomised, maximum subset matching algorithm for document-level music retrieval. In: Proceedings of the international conference on music information retrieval (ISMIR 2006), University of Victoria
11. Efrat A, Itai A, Katz M (2001) Geometry helps in bottleneck matching and related problems. Algorithmica 31(1):1–28
12. Mäkinen V, Ukkonen E (2002) Local similarity based point-pattern matching. In: Proceedings of the 13th annual symposium on combinatorial pattern matching (CPM 2002). LNCS, vol 2373. Springer, Berlin, pp 115–132
13. Ukkonen E, Lemström K, Mäkinen V (2003) Sweepline the music! In: Klein R, Six HW, Wegner L (eds) Computer science in perspective, essays dedicated to Thomas Ottmann. LNCS, vol 2598. Springer, pp 330–342

Polygon Triangulation

Jan Vahrenhold
Department of Computer Science, Westfälische Wilhelms-Universität Münster, Münster, Germany

Keywords

Computational geometry; Partitioning; Trapezoidation; Triangulation

Years and Authors of Summarized Original Work

1991; Chazelle

Problem Definition

Definition 1 A *simple polygon* is a polygon whose interior is simply connected, i.e., it consists of a single connected component and does not contain holes.

Definition 2 A triangulation of a simple polygon P with N vertices is a partition of the polygon, considered as a full-dimensional subset of the plane, into $N-2$ nonoverlapping triangles such that the set of vertices of these triangles is the set of vertices of P, such that no edge of a triangle lies outside of P, and such that no triangle edges intersect except in their common endpoints.

Key Results

In addition to the regularization-based approach by Garey et al. [7], three other $\mathcal{O}(N \log N)$-time algorithms are milestones on the way toward an optimal linear-time algorithm. In the first of these algorithms [2], Chazelle uses a linear-time "polygon-cutting" approach to partition a simple polygon by a suitably chosen diagonal; the resulting divide-and-conquer scheme yields an $\mathcal{O}(N \log N)$-time algorithm for simple polygons. Hertel and Mehlhorn [8] present an $\mathcal{O}(N \log N)$-time plane-sweep algorithm and refine its analysis to yield an $\mathcal{O}(N + R \log R)$-time upper bound, where R is the number of concave polygon angles. Chazelle and Incerpi [4] present a divide-and-conquer algorithm with $\mathcal{O}(N + S \log S)$ running time; here, S denotes the maximum number of times the boundary of the polygon changes from "spiraling" to "antispiraling."

Polygon Triangulation in $o(N \log N)$ Time

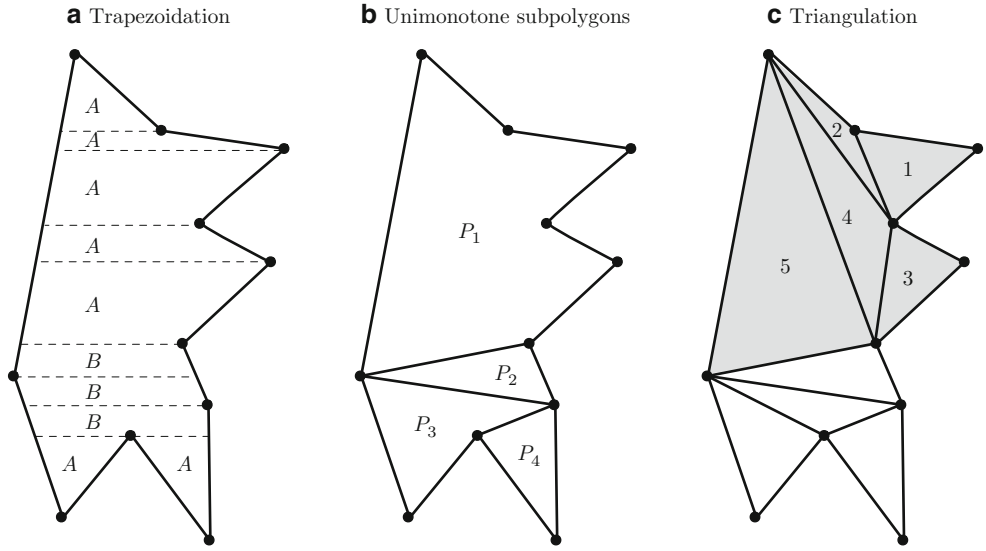
Algorithms with $o(N \log N)$ running time were developed by Tarjan and van Wyk [14] (using Jordan sorting and finger trees) and Kirkpatrick, Klawe, and Tarjan [10] (using efficient point-location structures). Both algorithms can be shown to run in $\mathcal{O}(N \log \log N)$ time; the algorithm by Kirkpatrick et al. can be made to run in $\mathcal{O}(N \log^* N)$ time if the polygon's vertices have bounded integer coordinates. Clarkson, Tarjan, and van Wyk [7] restate the algorithm by Tarjan and van Wyk in a randomized setting using random sampling and develop a Las Vegas algorithm with $\mathcal{O}(N \log^* N)$ expected running time. The same expected running time can be obtained by a considerably simpler randomized incremental construction presented by Seidel [7]; as an added benefit, this algorithm constructs an efficient data structure for vertical ray shooting among a set of line segments.

Polygon Triangulation via Trapezoidation

The key to an efficient polygon triangulation algorithm was that polygon triangulation is linear-time equivalent to polygon trapezoidation. Here, the task is to compute for each vertex v of a simple polygon P the point (if any) of the boundary of P that is visible from v when shooting horizontal rays (*chords*) from v toward $\pm\infty$ through the interior of P. The resulting structure is called the *visibility map* of P.

Theorem 1 ([6]) *Given the trapezoidal decomposition of a simple polygon P, a triangulation of P can be computed in linear time and vice versa.*

The proof builds upon the fact that a trapezoidation for a set of points in general position, i.e., a set in which no two points share the same y-coordinate, consists of trapezoids, which may degenerate into triangles that have exactly two polygon vertices on their boundary. A trapezoid T is said to be a *class-A*-trapezoid if these vertices lie on the same side of T (this includes the case of triangles); otherwise, it is said to be a *class-B*-trapezoid. Fournier and Montuno observed that a polygon can be partitioned into so-called unimonotone polygons by adding diagonals between the vertices of class-B-trapezoids



Polygon Triangulation, Fig. 1 Phases of Fournier and Montuno's algorithm [6]. By connecting the vertices of class-B-trapezoids in (**a**), the polygon is subdivided in unimonotone polygons as shown in (**b**). The numbers in (**c**) indicate the order in which the ear-cutting algorithms construct the triangles in the triangulation of the unimonotone subpolygon P_1 (shown in *gray*)

and that these unimonotone polygons can be triangulated independently in linear time – see Fig. 1.

Polygon Triangulation in Linear Time

The only deterministic linear-time algorithm for triangulating a simple polygon known so far is due to Chazelle [3]. Chazelle's algorithm uses a divide-and-conquer approach to compute the visibility map of a simple polygon. Since the divide-and-conquer approach subdivides the polygon's boundary into polygonal chains, there is no proper notion of the interior of the polygon through which the chords are supposed to pass. Instead, the polygon is embedded into the spherical plane on which the chords can "warp around infinity," and the visibility map is computed by always shooting rays in both directions.

Stated in terms of visibility maps, the algorithm's task now can be reduced to merging two visibility maps. To avoid linear-time merging steps which in turn would lead to $\Theta(N \log N)$ runtime, the algorithm proceeds as follows: in a first, bottom-up phase, the algorithm repeatedly merges the visibility maps of two subchains of the same length that share a common vertex. To ensure a sublinear running time, however, the algorithm does not compute the full visibility map, i.e., the map obtained by shooting rays from each vertex. Instead, for a subchain consisting of $2^{\lambda} + 1$ vertices, the algorithm maintains a $2^{\lceil \beta\lambda \rceil}$-*granular* visibility map.

Definition 3 A visibility map for a polygonal chain P is γ-granular for some $\gamma > 0$ if no part of the boundary of any region consists of more than γ consecutive edges of P and if no two adjacent regions can be merged without violating this property.

The consequence of this definition is that for polygons in general position, i.e., polygons for which no two vertices share the same y-coordinate, a γ-granular visibility map consists of $\mathcal{O}\left(\frac{N}{\gamma} + 1\right)$ regions, each of which is bounded by a constant number of chords and polygonal chains with a total complexity of $\mathcal{O}(\gamma)$. This enables a compact representation of each submap

along with a uniform upper bound on the coarseness of the approximation of the visibility map.

For each subchain π considered throughout the bottom-up phase, the algorithm computes two "oracle" data structures, which are reused in the final phase of the algorithm. The first oracle, the so-called ray shooter, returns in $\mathcal{O}(f(\gamma))$ time for any point in the plane the first point of the γ-granular visibility map of π that is seen when shooting a horizontal ray in either direction. This oracle, whose construction is based upon Lipton and Tarjan's planar separator theorem [11], is used when merging the visibility map of π with the visibility map of a subchain π' that shares a common vertex p with π. Starting with p, the algorithm walks along π and π' and uses the respective ray shooter to update the visibility information for as many vertices as needed to guarantee the desired granularity. Due to Chazelle's polygon-cutting theorem [2], each region in either submap is closed under visibility. This implies that the "ray-shooter" oracle can be defined for each region separately, and only one oracle for a region in π needs to be queried for any vertex in π' (and vice versa) as long as the algorithm keeps track of the region the vertex currently under consideration lies in.

The second oracle, the so-called arc cutter, subdivides π in $\mathcal{O}(g(\gamma))$ time into $g(\gamma)$ subarcs each of which is given along with an $h(\gamma)$-granular visibility map. Using these two oracles, merging a γ_1-granular visibility map for a subchain consisting of N_1 vertices with a γ_2-granular visibility map for a subchain consisting of N_2 vertices, $\gamma_2 \geq \gamma_1$, can be done in

$$\mathcal{O}\left(\left(\frac{N_1}{\gamma_1} + \frac{N_2}{\gamma_2} + 1\right) f(\gamma_2) g(\gamma_2)(h(\gamma_2) + \log(N_1 + N_2))\right)$$

time. Chazelle proves that one can maintain these oracles such that $f(x) \in x^{0.74}$, $g(x) \in \mathcal{O}(\log x)$, and $h(x) \in \mathcal{O}(x^{0.20})$, with $\gamma_2 \in \mathcal{O}(N_2^{0.20})$; this eventually implies a sublinear merging step and hence a linear-time complexity of the bottom-up phase.

The final, top-down phase incrementally refines all regions of the visibility maps produced in the bottom-up phase. Using the "arc-cutter" oracle, each polygonal chain on the boundary of a region is subdivided into an appropriate number of subchains. As a result of the bottom-up phase and by carefully aligning the subchains considered in that phase with the results of the arc cutter, visibility maps, ray shooters, and arc cutters are available for each of these chains. The algorithm then uses the ray shooters to construct new chords and the arc cutters to further refine the visibility maps until the recursion bottoms out, and visibility maps of constant size can be refined by a brute-force algorithm. An inductive proof yields a linear runtime for the refinement of each region in the visibility map that was constructed in the bottom-up phase; hence, the overall running time is linear.

While Chazelle's algorithm uses only reasonably complicated data structures and subroutines, the analysis of both phases strongly suggests large constant factors hidden in the Big-Oh notation. In addition, the algorithm requires rather delicate implementation issues to be solved, in particular regarding the representation of the visibility maps, and thus it is not surprising that Chazelle mentioned developing a simpler, randomized algorithm with expected linear runtime as a major open problem.

Randomized Polygon Triangulation in Expected Linear Time

Over a decade after the publication of Chazelle's deterministic, linear-time algorithm, Amato, Goodrich, and Ramos [1] affirmatively answered Chazelle's question. Their algorithm follows Chazelle's two-phase approach and uses a bottom-up phase to preprocess helper data structures for so-called portal queries in the subchains' visibility maps. The top-down phase also subdivides the polygonal chain into smaller chains and refines the visibility maps. In contrast to Chazelle's algorithm, however, this refinement

step is done on a random sample of the subchains only. As the sampling probability tends to one as the size of the subchain approaches $\mathcal{O}(1)$, both correctness and an expected linear runtime can be shown.

Applications

Being able to efficiently triangulate simple polygons has a variety of applications in computational geometry, computer graphics, and geographic information systems. For some of the problems considered, using a linear-time polygon triangulation algorithm is the key to obtaining an optimal algorithm. One such example among a variety of results is the optimal point-location scheme presented by Kirkpatrick [9] whose preprocessing time is linear assuming the availability of a linear-time triangulation algorithm. Several other applications are covered in O'Rourke's textbook on art gallery problems [12].

Open Problems

The main open problem is to devise an optimal deterministic algorithm that is reasonably efficient in practice.

Experimental Results

Due to its inherent complexity, Chazelle's algorithm has eluded a rigorous experimental evaluation so far. Preliminary results reported by Vahrenhold [15], however, seem to indicate that running even the first nontrivial stage of the bottom-up process takes significantly more time than running, e.g., Seidel's randomized algorithm [13] in full. Similarly, Amato et al. [1] conjecture that their randomized algorithm, despite its expected optimal runtime, is "not likely to be of practical value," either. Hence, for practical purposes, the simplicity of the deterministic algorithm by Hertel and Mehlhorn [8] and the randomized algorithm by Seidel [13] strongly advocates their use.

Cross-References

▸ Geometric Shortest Paths in the Plane
▸ Point Location

Recommended Reading

1. Amato NM, Goodrich MT, Ramos EA (2001) A randomized algorithm for triangulating a simple polygon in linear time. Discret Comput Geom 26:245–265
2. Chazelle BM (1982) A theorem on polygon cutting with applications. In: Proceedings of the twenty-third annual symposium on foundations of computer science, Chicago. IEEE, pp 339–349
3. Chazelle BM (1991) Triangulating a simple polygon in linear time. Discret Comput Geom 6:485–524
4. Chazelle BM, Incerpi JM (1984) Triangulation and shape-complexity. ACM Trans Graph 3(2):135–152
5. Clarkson KL, Tarjan RE, Van Wyk CJ (1989) A fast Las Vegas algorithm for triangulating a simple polygon. Discret Comput Geom 4:423–432
6. Fournier A, Montuno DY (1984) Triangulating simple polygons and equivalent problems. ACM Trans Graph 3(2):153–174
7. Garey MR, Johnson DS, Preparata FP, Tarjan RE (1978) Triangulating a simple polygon. Inf Process Lett 7(4):175–179
8. Hertel S, Mehlhorn K (1983) Fast triangulation of simple polygons. In: Karpinski M (ed) Proceedings of the 4th international conference on fundamentals of computation theory. Lecture notes in computer science, vol 158. Springer, Berlin, pp 207–218
9. Kirkpatrick DG (1983) Optimal search in planar subdivisions. SIAM J Comput 12(1):28–35
10. Kirkpatrick DG, Klawe MM, Tarjan RE (1992) Polygon triangulation in $O(n \log\log n)$ time with simple data structures. Discret Comput Geom 7:329–346
11. Lipton RJ, Tarjan RE (1979) A planar separator theorem for planar graphs. SIAM J Appl Math 36:177–189
12. O'Rourke J (1987) Art gallery theorems and algorithms. International series of monographs on computer science, vol 3. Oxford University Press, New York
13. Seidel R (1991) A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. Comput Geom Theory Appl 1(1):51–64
14. Tarjan RE, Van Wyk CJ (1988) An $O(n \log\log n)$-time algorithm for triangulating a simple polygon. SIAM J Comput 17(1):143–178; erratum in 17(5):1061 (1988)
15. Vahrenhold J (1996) Triangulierung eines einfachen Polygons in linearer Zeit. Master's thesis, Department of Computer Science, University of Münster (in German)

Position Auction

Aries Wei Sun
Department of Computer Science, City University of Hong Kong, Hong Kong, China

Keywords

Adword auction

Years and Authors of Summarized Original Work

2005; Varian

Problem Definition

This problem is concerned with the Nash equilibria of a game based on the ad auction used by Google and Yahoo. This research work [5] is motivated by the huge revenue that the adword auction derives every year. It defines two types of Nash equilibrium in the position auction game, applies economic analysis to the equilibria, and provides some empirical evidence that the Nash equilibria of the position auction describes the basic properties of the prices observed in Google's adword auction reasonably accurately. The problem being studied is closely related to the assignment game studied by [4, 1, 3]. And [2] has independently examined the problem and developed related results.

The Model and Its Notations

Consider the problem of assigning agents $a = 1, 2, \ldots, A$ to slots $s = 1, 2, \ldots, S$ where agent a's valuation for slot s is given by $u_{as} = v_a x_s$. The slots are numbered such that $x_1 > x_2 > \cdots > x_S$. It is assumed that $x_S = 0$ for all $s > S$ and the number of agents is greater than the number of slots. A higher position receives more clicks, so x_s can be interpreted as the click-through rate for slot s. The value $v_a > 0$ can be interpreted as the expected profit per click so $u_{as} = v_a x_s$ indicates the expected profit to advertiser a from appearing in slot s.

The slots are sold via an auction. Each agent bids an amount b_a, with the slot with the best click through rate being assigned to the agent with the highest bid, the second-best slot to the agent with the second highest bid, and so on. Renumbering the agents if necessary, let v_s be the value per click of the agent assigned to slot s. The price agent s faces is the bid of the agent immediately below him, so $p_t = b_{t+1}$. Hence the net profit that agent a can expect to make if he acquires slot s is $(v_a - p_s)\, x_s = (v_a - b_{s+1})\, x_s$.

Definitions

Definition 1 A Nash equilibrium set of prices ***(NE)*** satisfies

$$(v_s - p_s)\, x_s \geq (v_s - p_t)\, x_t, \text{for } t > s$$
$$(v_s - p_s)\, x_s \geq (v_s - p_{t-1})\, x_t, \text{for } t < s$$

where $p_t = b_{t+1}$.

Definition 2 A symmetric Nash equilibrium set of prices ***(SNE)*** satisfies

$$(v_s - p_s)\, x_s \geq (v_s - p_t)\, x_t \text{ for all } t \text{ and } s.$$

Equivalently,

$$v_s\, (x_s - x_t) \geq p_s x_s - p_t x_t \text{ for all } t \text{ and } s.$$

Key Results

Facts of NE and SNE

Fact 1 (Non-negative surplus) In an SNE, $v_s \geq p_s$.

Fact 2 (Monotone values) In an SNE, $v_{s-1} \geq v_s$, for all s.

Fact 3 (Monotone prices) In an SNE, $p_{s-1} x_{s-1} > p_s x_s$ and $p_{s-1} \geq p_s$ for all s. If $v_s > p_s$ then $p_{s-1} > p_s$.

Fact 4 ($NE \supset SNE$) If a set of prices is an SNE then it is an NE.

Fact 5 (One-step solution) If a set of bids satisfies the symmetric Nash equilibria inequalities for $s + 1$ and $s - 1$, then it satisfies these inequalities for all s.

Fact 6 The maximum revenue NE yields the same revenue as the upper recursive solution to the SNE.

A Sufficient and Necessary Condition of the Existence of a Pure Strategy Nash Equilibrium in the Position Auction Game

Theorem 1 *In the position auction described before, a pure strategy Nash equilibrium exists if and only if all the intervals*

$$\left[\frac{p_s x_s - p_{s+1}x_{s+1}}{x_s - x_{s+1}}, \frac{p_{s-1}x_{s-1} - p_s x_s}{x_{s-1} - x_s}\right], \quad \text{for } s = 2, 3, \ldots, S$$

are non-empty.

Applications

The model studied in this paper is a simple and elegant abstraction of the real adword auctions used by search engines such as Google and Yahoo. Different search engines have slightly different rules. For example, Yahoo ranks the advertisers according to their bids, while Google ranks the advertisers not only according to their bids but also according to the likelihood of their links being clicked.

However, similar analysis can be applied to real world situations, as the author has demonstrated above for the Google adword auction case.

Cross-References

▸ Adwords Pricing

Recommended Reading

1. Demange G, Gale D, Sotomayor M (1986) Multi-item auctions. J Polit Econ 94(4):863–872
2. Edelman B, Ostrovsky M, Schwartz M (2005) Internet advertising and the generalized second price auction. NBER Working Paper, 11765, Nov 2005
3. Roth A, Sotomayor M (1990) Two-sided matching. Cambridge University Press, Cambridge
4. Shapely L, Shubik M (1972) The assignment game I: the core. Int J Game Theory 1:111–130
5. Varian HR (2007) Position auctions. Int J Ind Organ 25(6):1163–1178

Power Grid Analysis

Sachin S. Sapatnekar
Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA

Keywords

EDA; Electronic design automation; Noise; On-chip variations; Reliability; Signal integrity

Years and Authors of Summarized Original Work

2002; Zhao, Panda, Sapatnekar, Blaauw
2002; Kozhaya, Nassif, Najm
2005; Qian, Nassif, Sapatnekar

Problem Definition

The power grid of an integrated system is responsible for providing reliable supply and ground voltages to every circuit element in the system. Degradations in the supply voltage levels can result in parametric failures due to increased delays, whereby circuits no longer meet their specifications, as well as catastrophic failures due to incorrect gate switching. Further, power grids are susceptible to reliability faults due to catastrophic failure modes such as electromigration. Therefore, accurate power grid analysis is a vital step in integrated circuit design.

Power grids may be analyzed under DC waveforms that reflect the steady-state currents drawn by the circuit or under transient analysis that captures the response of the grid to specific time-varying current waveforms; inductive effects, particularly in the integrated circuit

package; and decoupling capacitors that are deliberately placed in the circuit to temper the effect of large transients. For both DC and transient analysis, the problem can be abstracted as solving a set of linear algebraic equations of the following form:

$$G\mathbf{V} = \mathbf{E}, \tag{1}$$

where $G \in \Re^{N \times N}$ models the conductances in the system, $\mathbf{V} \in \Re^{N}$ is the vector of unknown node voltages, and $\mathbf{E} \in \Re^{N}$ is the right-hand side (RHS) vector, modeling the current loads. In case of DC analysis, a single such system must be solved, while in case of transient analysis, one such system is solved at each time step. For computational efficiency, a constant time step is often used during transient analysis of power grids in order to ensure that the G matrix, whose entries depend on the time step, remains unchanged through the simulation.

Given a power grid topology with $|E|$ resistors, these equations can be formulated using modified nodal analysis [2] in $O(|E|)$ time. Matrix G is sparse and diagonally dominant ($\sum_{i \neq j} |g_{ij}| \leq g_{ii}, \forall i$), and all off-diagonals of G are less than or equal to zero.

The task of power grid analysis is to determine all voltage levels in the system and verify that the maximum deviation from their ideal values is within a user-specified bound and to ensure that the current density in each wire is within a user-specified limit in order to assure resilience to electromigration failure.

Key Results

Mainstream methods for solving such systems include direct methods such as LU/Cholesky factorization and iterative methods. Due to the favorable structural properties of the power grid, notably sparsity and diagonal dominance, it is possible to solve these systems efficiently. However, the scale of the problem, where power grids may have hundreds of millions or billions of resistors, poses large memory and computation challenges even to the most efficient solvers. As a result, there has been considerable work on developing specialized solvers for power grids. Notable contributions in this direction are described below.

Hierarchy-Based Solvers

In real designs, the power grid is inherently hierarchical since it is created as a part of a hierarchical design process, where individual blocks with locally constructed power are first designed individually and then assembled at the chip level. This structure is exploited in [9] to build a hierarchical solution to the grid.

Based on inherent hierarchy, the power grid has k local partitions, corresponding to blocks, and a global partition that connects the power grids within these blocks. The *global grid* is then defined to include the set of nodes that lie in the global partition and the port nodes, while the grid in a local partition constitutes a *local grid*. The local grid is connected to the global grid through a set of nodes called *ports*, and due to the hierarchical structure, the number of port nodes is a small fraction of all nodes. The technique consists of the following steps:

Macromodeling Each of the k local grids may be modeled as a multi-port linear element represented by a macromodel of the type $\mathbf{I} = A \cdot \mathbf{V} + \mathbf{S}$, where $\mathbf{I}$ and $\mathbf{V}$ are the vectors of port voltages and A and $\mathbf{S}$ are, respectively, a constant matrix and a constant vector. Here, the A matrix is sparsified with bounded and minimal loss of accuracy using an integer knapsack scheme.

Solution of the global grid Once the macromodels for all the local grids are generated, the entire network is abstracted simply as the global grid, with the macromodel elements connected to it at the port nodes. This system is solved to determine the voltages at all ports.

Solution of the local grids Given the port voltages, the local grids are then each separately solved to provide the solution to the entire system.

Multigrid-Based Solvers

Multigrid-based approaches are an effective way of solving large systems of equations and have been customized to solve power grids [3]. The solution proceeds by creating a coarsened form of the network with a reduced number of nodes, which can be solved efficiently, and then by propagating the result of this solution to the full network. The technique consists of four steps:

Grid reduction, in which the large power grid is coarsened by selecting a subset of nodes that are to be maintained, while the other nodes are removed. The number of variables is therefore significantly reduced from from n to m.

Interpolation, in which an $n \times m$ interpolation operator matrix P is defined to map the original grid to the coarsened grid. This interpolation operator relates the voltages on the removed nodes to those on the coarsened grid, thereby allowing the solution of the coarsened grid to accurately reflect that of the original grid.

Solution of the coarsened grid, in which a solution is found for the voltages in the coarsened grid by solving the above linear equations.

Mapping the solution from the coarsened grid to the original grid by applying the interpolation operator concludes the process.

Random Walk-Based Solvers

The diagonal dominance of the power grid enables a special property that creates an exact mapping between the solution of the power grid equations and the use of random walks on a network. This idea has been used in [8] and further sped up in [6]. Unlike other approaches that require all (or most) nodes in a system to be solved together, random walk approaches allow for a single node to be solved alone. This is particularly useful during incremental analysis and optimization [1].

The family of random walk methods has been extended to solve entire systems, in a marriage with iterative linear solvers based on conjugate gradient methods. The intuition is that since random walks provide approximate solutions rapidly, they can be used to build effective preconditioners for iterative solvers [7].

Applications

Power grid analysis is a vital ingredient in the design of every integrated circuit, and there are several commercial offerings of design automation tools that analyze power grids. Aside from the problem of solving the linear system, the issue of determining the worst-case excitation is also a difficult problem. In spite of numerous efforts, automated tools for this purpose have been excessively pessimistic and therefore ineffective. It is generally accepted that user-specified patterns are the most effective way to provide input excitations, particularly in a design world where the power grid must be analyzed at multiple corners and multiple modes (corresponding to different supply voltages that could be applied to the circuit).

Experimental Results

Intelligent solutions for solving linear systems of equations have found extensive use in the analysis of power grids. Solvers that are used include direct solvers as well as iterative solvers based on methods such as preconditioned conjugate gradient-based solvers. Preconditioners based on methods such as support trees have been found to be useful, and random walk preconditioners have also been shown to outperform conventional methods. Due to the computational nature of this task, there has been active work on developing parallel and multithreaded power grid solvers. For example, the 2011 and 2012 editions of the Tau workshop have hosted contests on solving power grid problems [4, 5].

URLs to Code and Data Sets

A set of power grid benchmarks have been made available to the community at http://dropzone.tamu.edu/~pli/PGBench.

Recommended Reading

1. Boghrati B, Sapatnekar SS (2014) Incremental analysis of power grids using backward random walks. ACM Trans Des Autom Electron Syst 19(3), Article 31. doi:10.1145/2611763. http://doi.acm.org/10.1145/2611763
2. Ho CW, Ruehli AE, Brennan PA (1975) The modified nodal approach to network analysis. IEEE Trans Circuits Syst 22(6):505–509
3. Kozhaya J, Nassif SR, Najm FN (2002) A multigrid-like technique for power grid analysis. IEEE Trans Comput-Aided Des Integr Circuits Syst 21(10):1148–1160
4. Li Z, Balasubramanian R, Liu F, Nassif S (2011) 2011 TAU power grid simulation contest: benchmark suite and results. In: Proceedings of the IEEE/ACM international conference on computer-aided design, San Jose, pp 478–481
5. Li Z, Balasubramanian R, Liu F, Nassif S (2012) 2012 TAU power grid simulation contest: benchmark suite and results. In: Proceedings of the IEEE/ACM international conference on computer-aided design, San Jose, pp 643–646
6. Miyakawa T, Yamanaga K, Tsutsui H, Ochi H, Sato T (2011) Acceleration of random-walk-based linear circuit analysis using importance sampling. In: Proceedings of the Great Lakes symposium on VLSI, Lausanne, pp 211–216
7. Qian H, Sapatnekar SS (2008) Stochastic preconditioning for diagonally dominant matrices. SIAM J Sci Comput 30(3):1178–1204
8. Qian H, Nassif SR, Sapatnekar SS (2005) Power grid analysis using random walks. IEEE Trans Comput-Aided Des Integr Circuits Syst 24(8):1204–1224
9. Zhao M, Panda RV, Sapatnekar SS, Blaauw D (2002) Hierarchical analysis of power distribution networks. IEEE Trans Comput-Aided Des Integr Circuits Syst 21(2):159–168

Predecessor Search

Mihai Pătraşcu
Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology (MIT), Cambridge, MA, USA

Keywords

IP lookup; Predecessor problem; Successor problem

Years and Authors of Summarized Original Work

2006; Pătraşcu, Thorup

Problem Definition

Consider an ordered universe U, and a set $T \subset U$ with $|T| = n$. The goal is to preprocess T, such that the following query can be answered efficiently: given $x \in U$, report the predecessor of x, i.e., $\max\{y \in T \mid y < x\}$. One can also consider the dynamic problem, where elements are inserted and deleted into T. Let t_q be the query time, and t_u the update time.

This is a fundamental search problem, with an impressive number of applications. Later, this entry discusses IP lookup (forwarding packets on the Internet), orthogonal range queries and persistent data structures as examples.

The problem was considered in many computational models. In fact, most models below were initially defined to study the predecessor problem.

Comparison model: The problem can be solved through binary search in $\Theta(\lg n)$ comparisons. There is a lot of work on adaptive bounds, which may be sublogarithmic. Such bounds may depend on the finger distance, the working set, entropy etc.

Binary search trees: Predecessor search is one of the fundamental motivations for binary search trees. In this restrictive model, one can hope for an instance optimal (competitive) algorithm. Attempts to achieve this are described in a separate entry. ($O(\log \log n)$-*competitive Binary Search Trees* (2004; Demaine, Harmon, Iacono, Pătraşcu))

Word RAM: Memory is organized as words of b bits, and can be accessed through indirection. Constant-time operations include the standard operations in a language such as C (addition, multiplication, shifts and bitwise operations).

It is standard to assume the universe is $U = \{1, \ldots, 2^\ell\}$, i.e., one deals with ℓ-bit integers.

The floating point representation was designed so that order is preserved when values are interpreted as integers, so any algorithm will also work for ℓ-bit floating point numbers.

The standard *transdichotomous* assumption is that $b = \ell$, so that an input integer is represented in a word. This implies $b \geq \lg n$.

Cell-probe model: This is a nonuniform model stronger than the word RAM, in which the operations are arbitrary functions on the memory words (cells) which have already been probed. Thus, t_q only counts the number of cell probes. This is an ideal model for lower bounds, since it does not depend on the operations implemented by a particular computer.

Communication games: Let Alice have the query x, and Bob have the set T. They are trying to find the predecessor of x through τ rounds of communication, where in each round Alice sends m_A bits, and Bob replies with m_B bits.

This can simulate the cell-probe model when $m_B = b$ and m_A is the logarithm of the memory size. Then $\tau \leq t_q$ and one can use communication complexity to obtain cell-probe lower bounds.

External memory: The unit of access is a page, containing B words of ℓ bits each. B-trees solve the problem with query and update time $O(\log_B n)$, and one can also achieve this oblivious to the value of B (See *Cache-oblivious B-tree* (2005; Bender, Demaine, Farach-Colton).). The cell-probe model with $b = B \cdot \ell$ is stronger than this model.

AC^0 RAM: This is a variant of the word RAM in which allowable operations are functions that have constant depth, unbounded fan-in circuits. This excludes multiplication from the standard set of operations.

RAMBO: this is a variant of the RAM with a nonstandard memory, where words of memory can overlap in their bits. In the static case this is essentially equivalent to a normal RAM. However, in the dynamic case updates can be faster due to the word overlap [5].

The worst-case logarithmic bound for comparison search is not particularly informative when efficiency really matters. In practice, B-trees and variants are standard when dealing with huge data sets. Solutions based on RAM tricks are essential when the data set is not too large, but a fast query time is crucial, such as in software solutions to IP lookup [7].

Key Results

Building on a long line of research, Pătraşcu and Thorup [15, 16] finally obtained matching upper and lower bounds for the static problem in the word RAM, cell-probe, external memory and communication game models.

Let S be the number of words of space available. (In external memory, this is equivalent to S/B pages.) Define $a = \lg S \cdot \ell / n$. Also define $\lg x = \lceil \log_2(x+2) \rceil$, so that $\lg x \geq 1$ even if $x \in [0, 1]$. Then the optimal search time is, up to constant factors:

$$\min \begin{cases} \log_b n = \Theta(\min\{\log_B n, \log_\ell n\}) \\ \lg \frac{\ell - \lg n}{a} \\ \frac{\lg \frac{\ell}{a}}{\lg\left(\frac{a}{\lg n} \cdot \lg \frac{\ell}{a}\right)} \\ \frac{\lg \frac{\ell}{a}}{\lg\left(\frac{\lg \frac{\ell}{a}}{\lg \frac{\lg n}{a}}\right)} \end{cases} \tag{1}$$

The bound is achieved by a deterministic query algorithm. For any space S, the data structure can be constructed in time $O(S)$ by a randomized algorithm, starting with the set T given in sorted order. Updates are supported in expected time $t_q + O(S/n)$. Thus, besides locating the element through one predecessor query, updates change a minimal fraction of the data structure.

Lower bounds hold in the powerful cell-probe model, and hold even for randomized algorithms. When $S \geq n^{1+\varepsilon}$, the optimal trade-off for communication games coincides to (1). Note that the case $S = n^{1+o(1)}$ essentially disappears in the reduction to communication complexity, because Alice's messages only depends on $\lg S$.

Thus, there is no asymptotic difference between $S = O(n)$ and, say, $S = O(n^2)$.

Upper Bounds

The following algorithmic techniques give the optimal result:

- *B-trees* give $O(\log_B n)$ query time with linear space.
- *Fusion trees*, by Fredman and Willard [10], achieve a query time of $O(\log_b n)$. The basis of this is a *fusion node*, a structure which can search among b^ε values in constant time. This is done by recognizing that only a few bits of each value are essential, and packing the relevant information about all values in a single word.
- *Van Emde Boas search* [18] can solve the problem in $O(\lg \ell)$ time by binary searching for the length of the longest common prefix between the query and a value in T. Beginning the search with a table lookup based on the first $\lg n$ bits, and ending when there is enough space to store all answers, the query time is reduced to $O(\lg((\ell - \lg n)/a))$.
- A technique by *Beame and Fich* [4] can perform a multiway search for the longest common prefix, by maintaining a careful balance between ℓ and n. This is relevant when the space is at least $n^{1+\varepsilon}$, and gives the third branch of (1). Pătraşcu and Thorup [15] show how related ideas can be implemented with smaller space, yielding the last branch of (1).

Observe that external memory only features in the optimal trade-off through the $O(\log_B n)$ term coming from B-trees. Thus, it is optimal to either use the standard, comparison-based B-trees, or use the best word RAM strategy which completely ignores external memory.

Lower Bounds

All lower bounds before [15] where shown in the communication game model. Ajtai [1] was the first to prove a superconstant lower bound. His results, with a correction by Miltersen [12], show that for polynomial space, there exists n as a function of ℓ making the query time $\Omega(\sqrt{\lg \ell})$, and likewise there exists ℓ a function of n making the query complexity $\Omega(\sqrt[3]{\lg n})$.

Miltersen et al. [13] revisited Ajtai's proof, extending it to randomized algorithms. More importantly, they captured the essence of the proof in an independent *round elimination lemma*, which is an important tool for proving lower bounds in asymmetric communication.

Beame and Fich [4] improved Ajtai's lower bounds to $\Omega(\lg \ell / \lg \lg \ell)$ and $\Omega(\sqrt{\lg n / \lg \lg n})$ respectively. Sen and Venkatesh [17] later gave an improved round elimination lemma, which can reprove these lower bounds, but also for randomized algorithms.

Finally, using the message compression lemma of [6] (an alternative to round elimination), Pătraşcu and Thorup [15] showed an optimal trade-off for communication games. This is also an optimal lower bound in the other models when $S \geq n^{1+\varepsilon}$, but not for smaller space.

More importantly, [15] developed the first tools for proving lower bounds exceeding communication complexity, when $S = n^{1+o(1)}$. This showed the first separation ever between a data structure or polynomial size, and one of near linear size. This is fundamentally impossible through a direct communication lower bound, since the reduction to communication games only depends on $\lg S$.

The full result of Pătraşcu and Thorup [15] it the trade-off (1). Initially, this was shown only for deterministic query algorithms, but eventually it was extended to a randomized lower bound as well [16]. Among the surprising consequences of this result was that the classic van Emde Boas search is optimal for near-linear space (and thus for dynamic data structures), whereas with quadratic space it can be beaten by the technique of Beame and Fich.

A key technical idea of [15] is to analyze many queries simultaneously. Then, one considers a communication game involving all queries, and proves a direct-sum version of the round elimination lemma. Arguably, the proof is even simpler than for the regular round elimination lemma. This is achieved by considering

a stronger model for the inductive analysis, in which the algorithm is allowed to *reject* a large fraction of the queries before starting to make probes.

Bucketing

The rich recursive structure of the problem can not only be used for fast queries, but also to optimize the space and update time – of course, within the limits of (1). The idea is to place ranges of consecutive values in buckets, and include a single representative of each bucket in the predecessor structure. After performing a query on the predecessor structure (now with fewer elements), one need only search within the relevant bucket.

Because buckets of size $w^{O(1)}$ can be handled in constant time by fusion trees, it follows that factors of w in space are irrelevant. A more extreme application of the idea is given by *exponential trees* [3]. Here buckets have size $\Theta(n^{1-\gamma})$, where γ is a sufficiently small constant. Buckets are handled recursively in the same way, leading to $O(\lg \lg n)$ levels. If the initial query time is at least $t_q \geq \lg^{\varepsilon} n$, the query times at each level decrease geometrically, so overall time only grows by a constant factor. However, any polynomial space is reduced to linear, for an appropriate choice of γ. Also, the exponential tree can be updated in $O(t_q)$ time, even if the original data structure was static.

Applications

Perhaps the most important application of predecessor search is IP lookup. This is the problem solved by routers for each packet on the Internet, when deciding which subnetwork to forward the packet to. Thus, it is probably the most run algorithmic problem in the world. Formally, this is an *interval stabbing* query, which is equivalent to predecessor search in the static case [9]. As this is a problem where efficiency really matters, it is important to note that the fastest deployed software solutions [7] use integer search strategies (not comparison-based), as theoretical results would predict.

In addition, predecessor search is used pervasively in data structures, when reducing problems to *rank space*. Given a set T, one often wants to relabel it to the simpler $\{1, \ldots, n\}$ ("rank space"), while maintaining order relations. If one is presented with new values dynamically, the need for a predecessor query arises. Here are a couple of illustrative examples:

- In orthogonal range queries, one maintains a set of points in U^d, and queries for points in some rectangle $[a_1, b_1] \times \cdots \times [a_d, b_d]$. Though bounds typically grow exponentially with the dimension, the dependence on the universe can be factored out. At query time, one first runs $2d$ predecessor queries transforming the universe to $\{1, \ldots, n\}^d$.
- To make pointer data structures persistent [8], an outgoing link is replaced by a vector of pointers, each valid for some period of time. Deciding which link to follow (given the time being queried) is a predecessor problem.

Finally, it is interesting to note that the lower bounds for predecessor hold, by reductions, for all applications described above. To make these reductions possible, the lower bounds are in fact shown for the weaker *colored predecessor* problem. In this problem, the values in T are colored red or blue, and the query only needs to find the color of the predecessor.

Open Problems

It is known [2] how to implement fusion trees with AC^0 instructions, but not the other query strategies. What is the best query trade-off achievable on the AC^0 RAM? In particular, can van Emde Boas search be implemented with AC^0 instructions?

For the dynamic problem, can the update times be made deterministic? In particular, can van Emde Boas search be implemented with fast deterministic updates? This is a very appealing problem, with applications to deterministic dictionaries [14]. Also, can fusion nodes be updated

deterministically in constant time? Atomic heaps [11] achieve this when searching only among $(\lg n)^{\varepsilon}$ elements, not b^{ε}.

Finally, does an update to the predecessor structure require a query? In other words, can $t_u = o(t_q)$ be obtained, while still maintaining efficient query times?

Cross-References

- ▸ Cache-Oblivious B-Tree
- ▸ $O(\log \log n)$-Competitive Binary Search Tree

Recommended Reading

1. Ajtai M (1988) A lower bound for finding predecessors in Yao's cell probemodel. Combinatorica 8(3):235–247
2. Andersson A, Miltersen PB, Thorup M (1999) Fusion trees can be implemented with AC^0 instructions only. Theor Comput Sci 215(1–2):337–344
3. Andersson A, Thorup M (2002) Dynamic ordered sets with exponential search trees. CoRR cs.DS/0210006. See also FOCS'96, STOC'00.
4. Beame P, Fich FE (2002) Optimal bounds for the predecessor problem and related problems. J Comput Syst Sci 65(1):38–72, See also STOC'99
5. Brodnik A, Carlsson S, Fredman ML, Karlsson J, Munro JI (2005) Worst case constant time priority queue. J Syst Softw 78(3):249–256, See also SODA'01
6. Chakrabarti A, Regev O (2004) An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In: Proceedings of the 45th IEEE symposium on foundations of computer science (FOCS), pp 473–482
7. Degermark M, Brodnik A, Carlsson S, Pink S (1997) Small forwarding tables for fast routing lookups. In: Proceedings of the ACM SIGCOMM, pp 3–14
8. Driscoll JR, Sarnak N, Sleator DD, Tarjan RE (1989) Making data structures persistent. J Comput Syst Sci 38(1):86–124, See also STOC'86
9. Feldmann A, Muthukrishnan S (2000) Tradeoffs for packet classification. In: Proceedings of the IEEE INFOCOM, pp 1193–1202
10. Fredman ML, Willard DE (1993) Surpassing the information theoretic bound with fusion trees. J Comput Syst Sci 47(3):424–436, See also STOC'90
11. Fredman ML, Willard DE (1994) Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J Comput Syst Sci 48(3):533–551, See also FOCS'90
12. Miltersen PB (1994) Lower bounds for union-split-find related problems on random access machines. In: 26th ACM symposium on theory of computing (STOC), pp 625–634
13. Miltersen PB, Nisan N, Safra S, Wigderson A (1998) On data structures and asymmetric communication complexity. J Comput Syst Sci 57(1):37–49, See also STOC'95
14. Pagh R (2000) A trade-off for worst-case efficient dictionaries. Nord J Comput 7:151–163, See also SWAT'00
15. Pătraşcu M, Thorup M (2006) Time-space trade-offs for predecessor search. In: Proceedings of the 38th ACM symposium on theory of computing (STOC), pp 232–240
16. Pătraşcu, M, Thorup M (2007) Randomization does not help searching predecessors. In: Proceedings of the 18th ACM/SIAM symposium on discrete algorithms (SODA)
17. Sen P, Venkatesh S (2003) Lower bounds for predecessor searching in the cell probe model. arXiv:cs.CC/0309033. See also ICALP'01, CCC'03
18. van Emde Boas P, Kaas R, Zijlstra E (1977) Design and implementation of an efficient priority queue. Math Syst Theory 10:99–127, Announced by van Emde Boas alone at FOCS'75

Predecessor Search, String Algorithms and Data Structures

Djamal Belazzougui
Department of Computer Science, Helsinki Institute for Information Technology (HIIT), University of Helsinki, Helsinki, Finland

Keywords

Cell probe model; Integer search; RAM model

Years and Authors of Summarized Original Work

1975; Van Emde Boas
1993; Fredman, Willard
2002; Beame, Fich
2006; Pătraşcu, Thorup
2007; Andersson, Thorup
2014; Pătraşcu, Thorup

Problem Definition

Given a set S of n keys from the set $[1 \dots 2^\ell]$, the goal of *predecessor search* is to return, given a key $y \in [1 \dots 2^\ell]$, the largest key $x \in S$ such that $x \leq y$. We have the following models:

Comparison model Balanced binary search trees [1, 6] can solve the problem in optimal $O(\log n)$ time in the comparison model, in which the key can only be manipulated through comparisons with each other.

External memory model In this model, it is assumed that the data is read and written into blocks of B elements (integers in our case) and the cost of a query or algorithm is the number of read or written blocks. In this model, B-trees [7] can solve the problem in $O(\log_B n)$ time and $O(n)$ space.

RAM model This is the main subject of study. The model assumes that all standard arithmetic and logic operations (including multiplication) on integers of length w take constant time, where w is the computer word size. It is assumed that $w \geq \ell \geq \log n$.

AC^0 RAM model This model is similar to the RAM model except that it only contains instructions that can be implemented with circuits of polynomial size, constant depth and unfounded fan-in. The only affected instruction is multiplication which cannot be implemented with such circuits. While this model has been often considered in the literature in the past, modern computers support multiplications very fast, and the bottleneck is usually the memory access.

Cell probe model This model is used to prove lower bounds. It also has an associated word size w, but the cost of a query or an update is just the number of accessed memory words (computations have zero cost).

Useful Concepts

The two main techniques used for predecessor search are *cardinality reduction* and *length reduction*. At every step of the query, one would wish to either reduce the cardinality of the searched set or reduce the length of the searched keys.

Balanced Binary Search Tree

Cardinality reduction is achieved through the use of *balanced binary search tree*. This allows doing a predecessor search in $O(\log n)$ time in the comparison model. The B-tree is a generalization of the balanced binary search trees, where every node can have B children instead of just two. A static balanced binary search tree allows one to divide the set of searched keys by 2 at every level. A static B-tree allows one to divide the set by B. In the dynamic case, the cardinality can be reduced by a factor less than 2 (less than B for a dynamic B-tree) at every level, but it is guaranteed that the cardinality goes to one after $\Theta(\log n)$ levels ($\Theta(\log_B n)$ for a B-tree).

Trie

A key concept is that of a *trie*. A predecessor search for a key in a trie takes $O(\ell)$ time. A trie built on a set S of n keys from $[1 \dots 2^\ell]$ is a binary tree with ℓ levels numbered top-down. All the leaves of the trie are at level ℓ. Every edge in the tree has a label that is either 0 or 1. Let $x[i]$ denote the bit number i of the integer x, where $i \in [1 \dots \ell]$ and the bits are numbered from the most to the least significant bit ($x[1]$ is the most significant bit). Denote by $x[i \dots j]$ the binary string that consists in the concatenation of the bits $x[i], \dots x[j]$. A node of the trie at depth $d \in [0, \ell]$ will be labeled by a binary string of length d, formed by concatenating the edge labels from the root to the node. There will be a node of depth d labeled by binary string p if and only if there exists at least an element $x \in S$ such that $x[1 \dots d] = p$. A node at depth $d < \ell$ labeled by string p will have as children the nodes at depth $d + 1$ labeled by $p0$ and $p1$ (if they both exist, otherwise it will only have one child). The leaves of the trie are labeled by strings of S and the root is labeled by the empty string. A trie occupies $O(n\ell \log n)$ bits. In order to support predecessor searches, every internal node of the trie stores two elements of S. The two elements stored by a node labeled by binary string p are the

largest element prefixed by p and the predecessor of $p0^{w-|p|}$. A predecessor search on a compacted trie for a key y is then done by traversing the trie top-down, at level $i \in [1 \ldots \ell]$ following the edge labeled by character $y[i+1]$. The node at which the traversal stops is called the *locus* of y. The predecessor of y is then easily determined from its locus. If the locus is a leaf labeled by x, then necessarily $x = y$ and y is returned as the predecessor. Otherwise, the locus is some internal node and the predecessor is one of the two elements of S stored at that internal node. Suppose that the internal node is at level i and is labeled by string p. Then, if $x[i+1] = 1$, the predecessor is the largest element prefixed by p; otherwise, it is the predecessor of $p0^{w-p}$. In a *compacted trie* (*Patricia trie*), only leaves and internal nodes with degree 2 are kept, resulting in a tree of $2n-1$ nodes, n leaves, and $n-1$ internal nodes (see Fig. 1). The trie will then occupy only $O(n(\ell+\log n))$ bits. A predecessor search can be implemented on a compacted trie in a way similar to the non-compacted trie. The main difference is that the locus in a compacted trie is either a leaf or a location in the middle of a compacted edge.

Key Results

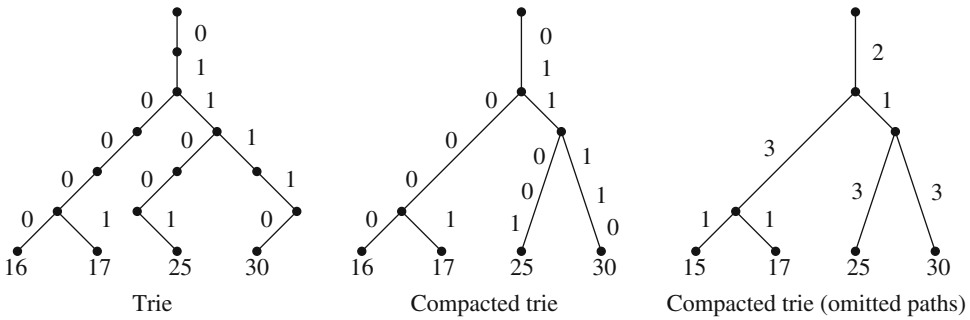
Van Emde Boas

The *Van Emde Boas tree* [17–19] is a compacted trie representation supporting predecessor search by doing a binary search on the trie levels. Since the number of levels is ℓ, the binary search takes time $O(\log \ell)$. The original structure used space $O(2^\ell)$. Later, Willard [20] showed how to use hashing to reduce the space usage to $O(n)$ while keeping the search time bounded by $O(\log \ell)$.

Searching a Compacted Trie (Fusion Trees)

The fusion tree reduces the predecessor search problem to the search on small compacted tries, each encoded in one memory word. The idea of the fusion tree dates back to a paper by Ajtai et al. [2], where it was shown how to implement predecessor search using a compacted trie in which the compacted paths are omitted and only their lengths are stored (the trie on the right in Fig. 1). This allows encoding a trie in $O(n(\log \ell + \log n))$ bits only compared to the $O(n(\ell + \log n))$ bits for the ordinary compacted trie. Then, a predecessor search for a key y is done by first doing a top-down traversal of the compacted trie, by always pretending that a comparison between the compacted paths and the bits of searched key is successful (only bits that label the edges are compared to bits of the searched key). At the end, the search terminates at a leaf that points to an integer $x \in S$ that is one of the elements that share the longest prefix with the searched key. Then, the locus (and hence the predecessor) is determined by filling the traversed compacted paths with bits from x and comparing those bits with the corresponding bits of y.

The main observation is that a predecessor search can be supported in constant time in the cell probe model whenever $n \log \ell \leq w$, since it



Predecessor Search, String Algorithms and Data Structures, Fig. 1 Three trie variants used for storing the integers 16, 17, 25, and 30. On the *left*, an ordinary trie; in the *middle*, a compacted trie; and on the *right*, a compacted trie in which the compacted paths are omitted and replaced by their lengths. Notice that the compacted trie has exactly $2 \cdot 4 - 1 = 7$ nodes

involves reading a constant number of memory locations corresponding to the trie encoded in a constant number of words and to two elements of S (the traversal of the trie is charged zero cost). Later, Fredman and Willard [11] invented the fusion node, where they implemented a predecessor search in constant time in the RAM model, when $n \leq w^{1/c}$ for $c = 6$. This allows searching the predecessors among a set of $w^{1/c}$ keys for $c = 6$ in constant time. By implementing a B-tree with block size $B = w^{1/c}$, one can achieve query time $O(\log_B n) = O(\log n / \log w)$. Finally, Pătraşcu and Thorup [15] have shown how to implement the approach with constant time updates and queries on $B = w^{1/c}$ keys for $c = 4$. This allows searches and dynamic updates in deterministic $O(\log n / \log w)$ time.

Beame and Fich

Beame and Fich [8] use a more advanced search that combines cardinality and length reductions. As a building block, they use a data structure that recursively reduces a search over n keys of length ℓ to a search over a group of n' keys of length ℓ' such that either $\ell' = \ell$ but $n' \leq q$ or $\ell' = \ell/h$ for some parameters h and q. This reduction technique was taken from an algorithm used for integer sorting [5]. Their search time is $O\left(\frac{\log \ell}{\log \log \ell}\right)$ and the space is quadratic. Combining with the fusion tree, this gives query time $O\left(\sqrt{\frac{\log n}{\log \log n}}\right)$ with quadratic space. This is achieved by using the fusion tree when $\log \ell \geq \sqrt{\log n \log \log n}$ and the new data structure when $\log \ell < \sqrt{\log n \log \log n}$. They also prove a matching lower bound, by showing that one cannot achieve space polynomial in n with query time $o\left(\frac{\log \ell}{\log \log \ell}\right)$ for all values of n and w or query time $o\left(\sqrt{\frac{\log n}{\log \log n}}\right)$ for all values of ℓ and w. Later, Sen and Venkatesh show that their lower bound holds even if randomization is allowed [16].

Exponential Search Trees

The *exponential search tree* [4] allows one to transform any predecessor data structure with polynomial space and preprocessing time, and query time $q(n, \ell, w)$, into a data structure with $O(n)$ space and preprocessing time, and query time at most $O(\log \log n \cdot q(n, \ell, w))$. Given some predecessor data structure P that supports queries in time $q(n, \ell, w)$ and uses space and preprocessing time $S(n) = O(n^c)$, the exponential search tree is a recursive data structure built from a sorted set $x_1 < \cdots < x_n$ of n keys as follows:

1. The root has degree $d = n^{1/(c+1)}$.
2. The n keys are partitioned into d blocks of sizes $b = \frac{n}{d}$ each and the data structure P is built on the set which consists in the first element of each of the blocks $2 \ldots d$ (the elements $x_{b+1}, \ldots, x_{(d-1)b+1}$).
3. The root has d children, where every child is itself an exponential search tree built on the $b = n^{c/(c+1)}$ elements of a block.

The recursion stops whenever we have a tree of constant size, in which the predecessor search is trivially supported in constant time. The construction time $C(n)$ follows the equation

$$C(n) = S(n^{1/(c+1)}) + n^{1/(c+1)} \cdot C(n^{c/(c+1)})$$

which gives $C(n) = O(n^{c/(c+1)}) + n^{1/(c+1)} \cdot C(n^{c/(c+1)})$. By iteratively expanding the term $C(n^{c/(c+1)})$, we get $C(n) = O(n)$.

A query is done by traversing the $\log \log n$ levels of the tree and doing predecessor searches at the structure P of each traversed node. The query time follows the equation $Q(n, \ell, w) = q(n^{1/(c+1)}, \ell, w) + Q(n^{c/(c+1)}, \ell, w)$, which solves to $Q(n, \ell, w) = O(\log \log n \cdot q(n, \ell, w))$. In the same paper [4], Andersson and Thorup show how to insert or delete an element in worst-case constant time, once a pointer to its predecessor (or to itself in case of a deletion) has been determined.

Deterministic Dynamic Bounds

When applied to Beame and Fich's solution with time $O\left(\frac{\log \ell}{\log \log \ell}\right)$, the exponential search tree gives $O(n)$ space with time $O\left(\log \log n \cdot \frac{\log \ell}{\log \log \ell}\right)$, and when applied to the

solution with $O\left(\sqrt{\frac{\log n}{\log\log n}}\right)$ time and quadratic space, it keeps the same time bound and achieves $O(n)$ space. By combining the two bounds with the bounds of the fusion tree [15], one gets the following time bounds which are the best known ones for dynamic deterministic predecessor search:

$$O\left(1+\min\left\{\begin{array}{c}\frac{\log n}{\log w}\\ \sqrt{\frac{\log n}{\log\log n}}\\ \log\log n\cdot\frac{\log \ell}{\log\log \ell}\end{array}\right\}\right) \quad (1)$$

The space is $O(n)$ for all the three branches. The bounds refer to the maximum of update and search times.

Optimal Static Bounds

Pătraşcu and Thorup [13] obtained optimal lower and upper bounds for the static case. They obtain optimal trade-offs between the time and the space usage. Define $a = \log\frac{S}{n} + \log w$, where S is the space usage. Then, the optimal time bound is

$$\Theta\left(1+\min\left\{\begin{array}{c}\frac{\log n}{\log w}\\ \log\frac{\ell-\log n}{\log w}\\ \frac{\log\frac{\ell}{a}}{\log\left(\frac{a}{\log n}\cdot\log\frac{\ell}{a}\right)}\\ \frac{\log\frac{\ell}{a}}{\log\left(\log\frac{\ell}{a}\Big/\log\frac{\log n}{a}\right)}\end{array}\right\}\right) \quad (2)$$

They later show that their lower bound holds even if randomization is allowed [14]. The first branch of the upper bound corresponds to the fusion tree. The second branch is obtained by first reducing the length from ℓ to $\ell - \log n$ bits, by dividing the universe into n buckets according to their most significant $\log n$ bits, and then by storing a separate Van Emde Boas structure for every bucket. The query time is then reduced from $O(\log(\ell - \log n))$ to $\log\frac{\ell-\log n}{\log w}$, by stopping the Van Emde Boas recursion when the key length gets to $\log w$ bits. Finally, the last two branches are implemented using a refinement of the technique by Beame and Fich.

Optimal Randomized Dynamic Bounds

When allowing randomization, optimal bounds can be achieved [15]. Again, the bounds are for the maximum of query and update times:

$$\Theta\left(1+\min\left\{\begin{array}{c}\frac{\log n}{\log w}\\ \log\left(\frac{\log(2^\ell-n)}{\log w}\right)\\ \frac{\log\frac{\ell}{\log w}}{\log\left(\log\frac{\ell}{\log w}/\log\frac{\log n}{\log w}\right)}\end{array}\right\}\right) \quad (3)$$

The space usage is linear ($O(n)$) or almost linear $O(nw^{O(1)})$.

The first branch of the upper bound corresponds to the dynamic version of the fusion tree [15] and the third branch to a dynamic version of the fourth branch of the optimal static upper bound. The second branch is similar to the second branch of the optimal static upper bound, with the difference that the term $\ell - \log n$ is replaced by $\log(2^\ell - n)$.

The first and third branches are trivially optimal, since the bounds are the same as the static ones and any lower bound that applies to the latter also applies to the former. The authors show that the second branch is also optimal by proving a corresponding lower bound that is stronger than the static one.

Applications

Range queries are very important in databases. Answering to range queries is an obvious and natural application of predecessor search. A query (in one dimension) asks, given a range $[a, b] \subseteq [1 \ldots 2^\ell]$, to return every element x in the set $S \cap [a, b]$. This can obviously be solved by doing two predecessor queries for a and b and then reporting all elements between the two predecessors (excluding the predecessor of a). In the comparison model and the external memory model, this is the best one can hope for, and the optimal time bounds are $O(\log n + |S \cap [a, b]|)$ and $O(\log_B n +$

$|S \cap [a,b]|/B$) respectively. Surprisingly, in the RAM model, there exists a linear-space static solution with $O(|S \cap [a,b]|)$ query time [3]. An important application of predecessor search is *IP forwarding problem*, which must be solved by every internet router. The router contains a database of subnetworks specified by their IP address prefixes, and each received packet has to be forwarded to the subnetwork with the longest matching prefix. The IP forwarding problem is an instance of the *longest common prefix problem*, which in the static case is equivalent to the predecessor problem.

The lower and upper bounds on predecessor search can be used to prove bounds for other problems through reductions. For example, predecessor search can be reduced to *two-dimensional range search*, allowing one to prove a lower bound of $\Omega(\log \log n)$ time for the two-dimensional range-emptiness problem on sets of n points on a grid of n rows by n columns. Optimal bounds can also be proved for rank queries on sequence representations through reduction to and from predecessor queries [9].

Open Problems

The deterministic complexity of the dynamic predecessor search is still open. Another open problem is whether updates can be supported faster than searches when the search time is optimal or near optimal (of course, one can always support constant update time when the query time is the trivial $\Theta(n)$). For the moment, this is not disallowed by any lower bound and has been achieved for the related *dynamic ranking problem* [10], in which a set $S \subset [1 \ldots 2^\ell]$ is maintained under updates and a query asks, given an integer $y \in S$ to count the number of elements of S smaller than y.

Cross-References

- ▸ Monotone Minimal Perfect Hash Functions
- ▸ Orthogonal Range Searching on Discrete Grids
- ▸ Rank and Select Operations on Sequences

Recommended Reading

1. Georgy Adelson-Velsky G, Landis E M (1962) An algorithm for the organization of information. Proc USSR Acad Sci (in Russian) 146:263–266. English translation by Myron J Ricci in Sov Math Dokl 3:1259–1263 (1962)
2. Ajtai M, Fredman M, Komlós J (1984) Hash functions for priority queues. Inf Control 63(3):217–225
3. Alstrup S, Brodal GS, Rauhe T (2001) Optimal static range reporting in one dimension. In: Proceedings of the thirty-third annual ACM symposium on theory of computing, Heraklion. ACM, pp 476–482
4. Andersson A, Thorup M (2007) Dynamic ordered sets with exponential search trees. J ACM (JACM) 54(3):13
5. Andersson A, Hagerup T, Nilsson S, Raman R (1998) Sorting in linear time? J Comput Syst Sci 57:74–93
6. Bayer R (1972) Symmetric binary b-trees: data structure and maintenance algorithms. Acta Inform 1(4):290–306
7. Bayer R, McCreight EM (1972) Organization and maintenance of large ordered indexes. Acta Inform 1:173–189
8. Beame P, Fich FE (2002) Optimal bounds for the predecessor problem and related problems. J Comput Syst Sci 65(1):38–72
9. Belazzougui D, Navarro G (2012) New lower and upper bounds for representing sequences. In: Algorithms–ESA 2012, Ljubljana. Springer, pp 181–192
10. Chan TM, Pătraşcu M (2010) Counting inversions, offline orthogonal range counting, and related problems. In: Proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms, Austin. Society for Industrial and Applied Mathematics, pp 161–173
11. Fredman ML, Willard DE (1993) Surpassing the information theoretic bound with fusion trees. J Comput Syst Sci 47(3):424–436
12. Mortensen CW, Pagh R, Pătraşcu M (2005) On dynamic range reporting in one dimension. In: Proceedings of the thirty-seventh annual ACM symposium on theory of computing, Baltimore. ACM, pp 104–111
13. Pătraşcu M, Thorup M (2006) Time-space tradeoffs for predecessor search. In: Proceedings of the thirty-eighth annual ACM symposium on theory of computing, San Diego. ACM, pp 232–240
14. Pătraşcu M, Thorup M (2007) Randomization does not help searching predecessors. In: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms, New Orleans. Society for Industrial and Applied Mathematics, pp 555–564
15. Pătraşcu M, Thorup M (2014) Dynamic integer sets with optimal rank, select, and predecessor search. In: Proceedings of of the 55th annual IEEE symposium on foundations of computer science, Philadelphia. arXiv preprint arxiv:1408.3045

16. Sen P, Venkatesh S (2008) Lower bounds for predecessor searching in the cell probe model. J Comput Syst Sci 74:364–385
17. van Emde Boas P (1975) Preserving order in a forest in less than logarithmic time. In: FOCS, Berkeley, pp 75–84
18. van Emde Boas P (1977) Preserving order in a forest in less than logarithmic time and linear space. Inf Process Lett 6(3):80–82
19. van Emde Boas P, Kaas R, Zijlstra E (1976) Design and implementation of an efficient priority queue. Math Syst Theory 10(1):99–127
20. Willard DE (1983) Log-logarithmic worst-case range queries are possible in space $\Theta(n)$. Inf Process Lett 17(2):81–84

Price of Anarchy

George Christodoulou
University of Liverpool, Liverpool, UK

Keywords

Congestion games; Network games; Price of anarchy

Synonyms

Coordination ratio

Years and Authors of Summarized Original Work

2005; Koutsoupias

Problem Definition

The *price of anarchy* captures the lack of coordination in systems where users are selfish and may have conflicted interests. It was first proposed by Koutsoupias and Papadimitriou in [8], where the term *coordination ratio* was used instead, but later Papadimitriou in [12] coined the term price of anarchy that finally prevailed in the literature.

Roughly, the price of anarchy is the system cost (e.g., makespan, average latency) of the worst-case Nash equilibrium over the optimal system cost that would be achieved if the players were forced to coordinate. Although it was originally defined in order to analyze a simple load-balancing game, it was soon applied to numerous variants and to more general games. The family of *(weighted) congestion games* [11, 13] is a nice abstract form to describe most of the alternative settings. (We focus our presentation on cost minimization problems in congestion games. We mention some utility maximization problems where price of anarchy analysis has been used in the Applications section.)

The price of anarchy may vary, depending on the

- Equilibrium solution concept *(e.g., pure, mixed, correlated equilibria)*
- Characteristics of the congestion game
  - Players Set *(e.g., atomic – non-atomic)*
  - Strategy Set *(e.g., symmetric asymmetric, parallel machines-network-general)*
  - Players' cost functions *(e.g., linear, polynomial)*
- Social cost *(e.g., maximum, sum, total latency)*

Notation

Let G be a (finite) game that is determined by the triple $(N, (\mathcal{S}_i)_{i\in N}, (c_i)_{i\in N})$. $N = \{1, \ldots, n\}$ is the set of the players that participate in the game. $\mathcal{S}_i$ is a *pure strategy set* for player i. An element $A_i \in \mathcal{S}_i$ is a *pure strategy* for player $i \in N$. A *pure strategy profile* $A = (A_1, \ldots, A_n)$ is a vector of pure strategies, one for each player. The set of all possible pure strategy profiles is denoted by $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$. The *cost* of a player $i \in N$, for a pure strategy, is determined by a cost function $c_i : \mathcal{S} \mapsto \mathbb{R}$.

A pure strategy profile A is a *pure Nash equilibrium*, if none of the players $i \in N$ can benefit, by unilaterally deviating to another pure strategy $s_i \in \mathcal{S}_i$:

$$c_i(A) \leq c_i(A_{-i}, s_i) \qquad \forall i \in N, \quad \forall s_i \in \mathcal{S}_i,$$

where (A_{-i}, s_i) is the simple strategy profile that results when just the player i deviates from strategy $A_i \in \mathcal{S}_i$ to strategy $s_i \in \mathcal{S}_i$.

Similarly, a *mixed strategy* p_i for a player $i \in N$ is a probability distribution over her pure strategy set $\mathcal{S}_i$. A mixed strategy profile p is the tuple $p = (p_1, \ldots p_n)$, where player i chooses mixed strategy p_i. The expected cost of a player $i \in N$ with respect to the p is

$$c_i(p) = \sum_{A \in \mathcal{S}} p(A) c_i(A),$$

where $p(A) = \prod_{i \in N} p_i(A_i)$ is the probability that pure strategy A occurs, with respect to $(p_i)_{i \in N}$. A mixed strategy profile p is a *Nash equilibrium*, if and only if

$$c_i(p) \leq c_i(p_{-i}, s_i) \qquad \forall i \in N, \quad \forall s_i \in \mathcal{S}_i$$

The *social cost* of a pure strategy profile A, denoted by $\mathrm{SC}(A)$, is the maximum cost of a player $\textsc{Max}(A) = \max_{i \in N} c_i(A)$ or the average cost of a player. For simplicity, the sum of the players' cost is considered (i.e., n times the average cost) $\textsc{Sum}(A) = \sum_{i \in N} c_i(A)$. The same definitions extend naturally for the case of mixed strategies, but with expected costs in this case.

The (mixed) *price of anarchy* [8] for a game is the worst-case ratio, among all the (mixed) Nash equilibria, of the social cost over the optimal cost, $\textsc{Opt} = \min_{P \in \mathcal{S}} \mathrm{SC}(P)$.

$$\mathrm{PA} = \max_{p \text{ is N.E.}} \frac{\mathrm{SC}(p)}{\textsc{Opt}}$$

The price of anarchy for a class of games is the maximum (supremum) price of anarchy among all the games of this class.

Congestion Games Here, a general class of games is described that captures most of the games for which price of anarchy is studied in the literature. A *congestion game [11, 13]*, is defined by the tuple $(N, E, (\mathcal{S}_i)_{i \in N}, (f_e)_{e \in E})$, where $N = \{1, \ldots, n\}$ is a set of players, E is a set of *facilities*, $\mathcal{S}_i \subseteq 2^E$ is the pure strategy set for player i, a pure strategy $A_i \in \mathcal{S}_i$ is a subset of the facility set, and f_e is a *cost (or latency) function* (Unless otherwise stated, linear cost functions are only considered throughout this article. See [14] and references therein for results about more general cost functions, and for additional results see entries 00260, 00251, 00053.) with respect to the facility $e \in E$.

A pure strategy profile $A = (A_1, \ldots, A_n)$ is a vector of pure strategies, one for each player. The cost $c_i(A)$ of player i for the pure strategy profile A is given by

$$c_i(A) = \sum_{e \in A_i} f_e(n_e(A)),$$

where $n_e(A)$ is the number of the players that use facility e in A.

A congestion game is called *symmetric or single commodity*, if all the players have the same strategy set: $\mathcal{S}_i = \mathcal{C}$. The term *asymmetric or multi-commodity* is used to refer to all the games including the symmetric ones. A special class is the class of *network congestion games*. In these games, the facilities are edges of a (multi)graph $G(V, E)$. The pure strategy set for a player $i \in N$ is the simple paths set from a source $s_i \in V$ to a destination $t_i \in V$. In network symmetric congestion games, all the players have the same source and destination.

A natural generalization of congestion games are the *weighted congestion games*, where every player controls an amount of traffic w_i. The cost of each facility $e \in E$ depends on the total load of the facility. In this case, a well-studied social cost function is the weighted sum of players costs, or *total latency*.

In a congestion game with *splittable weights (divisible demands)*, every player $i \in N$, instead of fixing a single pure strategy, is allowed to distribute her demand among her pure strategy set.

Finally, in a *non-atomic congestion game*, there are k different player types $1 \ldots k$. Players are infinitesimal and for each player type i the continuum of the players is denoted by the interval $[0, n_i]$. In general, each player type contributes in a different way to the congestion on the facility $e \in E$, and this contribution is

determined by a positive *rate of consumption* $r_{s,e}$ for a strategy $s \in \mathcal{S}_i$ and a facility $e \in s$. Each player chooses a strategy that results in a *strategy distribution* $x = (x_s)_{s\in\mathcal{S}}$, with $\sum_{s\in S_i} x_s = n_i$.

Key Results

Maximum Social Cost

First, we review results on the price of anarchy w.r.t maximum social cost that was historically the first social cost considered in [8]. Formally, for a pure strategy profile A, the social cost is

$$\mathrm{SC}(A) = \textsc{Max}(A) = \max_{i\in N} c_i(A)$$

The definition naturally extends to mixed strategies.

Theorem 1 ([7–10]) *The price of anarchy for m identical machines is* $\Theta\left(\frac{\log m}{\log\log m}\right)$.

Theorem 2 ([7]) *The price of anarchy for m uniformly related machines with speeds* $s_1 \geq s_2 \geq \cdots \geq s_m$ *is*

$$\Theta\left(\min\left\{\frac{\log m}{\log\log\log m}, \frac{\log m}{\log\left(\frac{\log m}{\log(s_1/s_m)}\right)}\right\}\right).$$

Theorem 3 ([4]) *The price of anarchy for pure equilibria is* $\Theta(\sqrt{n})$ *for asymmetric but at most* $5/2$ *for symmetric congestion games.*

Average Social Cost: Total Latency

Here, we consider as social cost the (weighted) sum (total latency) of the players' cost for (weighted) congestion games, i.e.,

$$\mathrm{SC}(A) = \textsc{Sum}(A) = \sum_{i\in N} c_i(A),$$

$$\mathrm{SC}(A) = \mathrm{C}(A) = \sum_{i\in N} w_i c_i(A).$$

The definition naturally extends for mixed strategies.

Theorem 4 ([2–4]) *The price of anarchy is* $5/2$ *for asymmetric and* $\frac{5n-2}{2n+1}$ *for symmetric congestion games.*

Theorem 5 ([2, 3]) *The price of anarchy for weighted congestion games is* $1+\phi \approx 2.618$.

Theorem 6 ([6]) *The price of anarchy is at most* $3/2$ *for congestion games with splittable weights.*

Theorem 7 ([15, 16]) *The price of anarchy for non-atomic congestion games is* $4/3$.

Key Proof Technique: Smoothness Most of the above results on atomic congestion games have been generalized for polynomial latencies [1–3] and hold for various equilibrium concepts. Roughgarden's *smoothness* framework [14] distills the main ideas in these proofs and provides a general, canonical proof recipe to obtain price of anarchy bounds. He also shows how smoothness provides tight bounds for congestion games with general cost functions.

Applications

The efficiency of large-scale networks, in which selfish users interact, is highly affected due to the users' selfish behavior. The price of anarchy is a quantitative measure of the lack of coordination in such systems. It is a useful theoretical tool for the analysis and design of telecommunication and traffic networks, where selfish users compete on system's resources motivated by their atomic interests and are indifferent to the social welfare.

The price of anarchy has been also studied in utility maximization Problems; see, for example, Valid Utility Games [17]. Finally, a line of work shows that price of anarchy can be used to evaluate the performance of mechanisms; see, for example, [5] for an analysis of simultaneous Second-Price Auctions.

Cross-References

- Best Response Algorithms for Selfish Routing
- Computing Pure Equilibria in the Game of Parallel Links

▸ Price of Anarchy for Machines Models
▸ Selfish Unsplittable Flows: Algorithms for Pure Equilibria

Recommended Reading

1. Aland S, Dumrauf D, Gairing M, Monien B, Schoppmann F (2006) Exact price of anarchy for polynomial congestion games. In: 23rd annual symposium on theoretical aspects of computer science (STACS), Marseille, pp 218–229
2. Awerbuch B, Azar Y, Epstein A (2005) Large the price of routing unsplittable flow. In: Proceedings of the 37th annual ACM symposium on theory of computing (STOC), Baltimore, pp 57–66
3. Christodoulou G, Koutsoupias E (2005) On the price of anarchy and stability of correlated equilibria of linear congestion games. In: Algorithms – ESA 2005, 13th annual European symposium, Palma de Mallorca, pp 59–70
4. Christodoulou G, Koutsoupias E (2005) The price of anarchy of finite congestion games. In: Proceedings of the 37th annual ACM symposium on theory of computing (STOC), Baltimore, pp 67–73
5. Christodoulou G, Kovács A, Schapira M (2008) Bayesian combinatorial auctions. In: ICALP '08: proceedings of the 35th international colloquium on automata, languages and programming, part I, Reykjavik, pp 820–832
6. Cominetti R, Correa JR, Moses NES (2006) Network games with atomic players. In: Automata, languages and programming, 33rd international colloquium (ICALP), Venice, pp 525–536
7. Czumaj A, Vöcking B (2002) Tight bounds for worst-case equilibria. In: Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms (SODA), San Francisco, pp 413–420
8. Koutsoupias E, Papadimitriou CH (1999) Worst-case equilibria. In: Proceedings of the 16th annual symposium on theoretical aspects of computer science (STACS), Trier, pp 404–413
9. Koutsoupias E, Mavronicolas M, Spirakis PG (2003) Approximate equilibria and ball fusion. Theory Comput Syst 36:683–693
10. Mavronicolas M, Spirakis PG (2001) The price of selfish routing. In: Proceedings on 33rd annual ACM symposium on theory of computing (STOC), Hersonissos, pp 510–519
11. Monderer D, Shapley L (1996) Potential games. Games Econ Behav 14:124–143
12. Papadimitriou CH (2001) Algorithms, games, and the internet. In: Proceedings on 33rd annual ACM symposium on theory of computing (STOC), Hersonissos, pp 749–753
13. Rosenthal RW (1973) A class of games possessing pure-strategy Nash equilibria. Int J Game Theory 2:65–67
14. Roughgarden T (2012) Intrinsic robustness of the price of anarchy. Commun ACM 55(7):116–123
15. Roughgarden T, Tardos E (2002) How bad is selfish routing? J ACM 49:236–259
16. Roughgarden T, Tardos E (2004) Bounding the inefficiency of equilibria in nonatomic congestion games. Games Econ Behav 47:389–403
17. Vetta A (2002) Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In: Proceedings of the 43rd annual symposium on foundations of computer science (FOCS), Vancouver, pp 416–425

Price of Anarchy for Machines Models

Artur Czumaj[1] and Berthold Vöcking[2]
[1]Department of Computer Science, Centre for Discrete Mathematics and Its Applications, University of Warwick, Coventry, UK
[2]Department of Computer Science, RWTH Aachen University, Aachen, Germany

Keywords

Algorithmic game theory; Coordination ratio; Nash equilibria; Noncooperative networks; Price of anarchy; Selfish routing; Selfish strategies; Traffic routing

Years and Authors of Summarized Original Work

2002; Czumaj, Vöcking

Problem Definition

This entry considers a selfish routing model formally introduced by Koutsoupias and Papadimitriou [10], in which the goal is to route the traffic on parallel links with linear latency functions. One can describe this model as a scheduling problem with m independent machines with speeds $s_1, \ldots, s_m$ and n independent tasks with weights $w_1, \ldots, w_n$. The goal is to allocate the tasks to the machines to minimize the maximum load of the links in the system.

It is assumed that all tasks are assigned by noncooperative agents. The set of *pure strategies* for task i is the set $\{1,\ldots,m\}$, and a *mixed strategy* is a distribution on this set.

Given a combination $(j_1,\ldots,j_n) \in \{1,\ldots,m\}^n$ of pure strategies, one for each task, the *cost* for task i is $\sum_{j_k=j_i} \frac{w_k}{s_{j_i}}$, which is the time needed for machine j_i chosen by task i to complete all tasks allocated to that machine. Similarly, for a combination of pure strategies $(j_1,\ldots,j_n) \in \{1,\ldots,m\}^n$, the *load* of machine j is defined as $\sum_{j_k=j} \frac{w_k}{s_j}$.

Given n tasks of length $w_1,\ldots,w_n$ and m machines with the speeds $s_1,\ldots,s_m$, let opt denote the *social optimum*, that is, the minimum cost over all combinations of pure strategies:

$$\mathsf{opt} = \min_{(j_1,\ldots,j_n)\in\{1,\ldots,m\}^n} \max_{1\le j\le m} \sum_{i:j_i=j} \frac{w_i}{s_j}.$$

For example, if all machines have the same unit speed ($s_j = 1$ for every j, $1 \le j \le m$) and all tasks have the same unit weight ($w_i = 1$ for every i, $1 \le i \le n$), then the social optimum is $\lceil \frac{n}{m} \rceil$.

It is also easy to see that in any system

$$\mathsf{opt} \ge \frac{\max_i w_i}{\max_j s_j}.$$

It is known that computing the social optimum is $\mathcal{NP}$-hard even for identical speeds (see [10]).

For mixed strategies, let p_i^j denote the probability that an agent i sends the entire traffic w_i to a machine j. Let ℓ_j denote the *expected load* on a machine j, that is,

$$\ell_j = \frac{1}{s_j} \cdot \sum_{i=1}^{n} w_i\, p_i^j.$$

For a task i, the *expected cost of task i on machine j* is equal to

$$c_i^j = \frac{w_i}{s_j} + \sum_{t\ne i} \frac{w_t\, p_t^j}{s_j} = \ell_j + (1 - p_i^j)\,\frac{w_i}{s_j}.$$

The expected cost c_i^j corresponds to the expected finish time of task i on machine j under the processor sharing scheduling policy. This is an appropriate cost model with respect to the underlying traffic routing application.

Definition 1 (Nash equilibrium) The probabilities $(p_i^j)_{1\le i\le n, 1\le j\le m}$ define a *Nash equilibrium* if and only if any task i will assign nonzero probabilities only to machines that minimize c_i^j, that is, $p_i^j > 0$ implies $c_i^j \le c_i^q$, for every q, $1 \le q \le m$.

As an example, in the system considered above in which all machines have the same unit speed and all weights are the same, the uniform probabilities $p_i^j = \frac{1}{m}$ for all $1 \le j \le m$ and $1 \le i \le n$ define a system in a Nash equilibrium.

The existence of a Nash equilibrium over mixed strategies for noncooperative games was shown by Nash [13]. In fact, the routing game considered here admits an equilibrium even if all players are restricted to pure strategies, what has been shown by Fotakis et al. [7].

Fix an arbitrary Nash equilibrium, that is, fix the probabilities $(p_i^j)_{1\le i\le n, 1\le j\le m}$ that define a Nash equilibrium. Consider the randomized allocation strategies in which each task i is allocated to a single machine chosen independently at random according to the probabilities p_i^j, that is, task i is allocated to machine j with probability p_i^j. Let C_j, $1 \le j \le m$, be the random variable indicating the *load of machine j* in our random experiment. Observe that C_j is the weighted sum of independent 0–1 random variables J_i^j, $\mathbf{Pr}[J_i^j = 1] = p_i^j$, such that

$$C_j = \frac{1}{s_j} \sum_{i=1}^{n} w_i \cdot J_i^j.$$

Let $\mathfrak{c}$ denote the *maximum expected load* over all machines, that is,

$$\mathfrak{c} = \max_{1\le j\le m} \ell_j.$$

Notice that $\mathbf{E}[C_j] = \ell_j$, and therefore, $\mathfrak{c} = \max_{1\le j\le m} \mathbf{E}[C_j]$.

Finally, let the *social cost* C be defined as the expected maximum load (instead of maximum expected load), that is,

$$\mathsf{C} = \mathbf{E}[\max_{1 \le j \le m} C_j].$$

Observe that $c \le C$ and possibly $c \ll C$. The goal is to estimate the *price of anarchy* (also called the *worst-case coordination ratio*) which is the worst-case ratio

$$R = \max \frac{\mathsf{C}}{\mathsf{opt}},$$

where the maximum is over all Nash equilibria.

Key Results

Early Work

The study of the price of anarchy has been initiated by Koutsoupias and Papadimitriou [10], who showed also some very basic results for this model. For example, they proved that for two identical machines, the price of anarchy is exactly $\frac{3}{2}$, and for two machines (with possibly different speeds), the price of anarchy is at least $\phi = \frac{1+\sqrt{5}}{2}$. Koutsoupias and Papadimitriou showed also that for m identical machines, the price of anarchy is $\Omega(\frac{\log m}{\log \log m})$ and it is at most $\mathcal{O}(\sqrt{m \ln m})$, and for m arbitrary machines, the price of anarchy is $\mathcal{O}(\sqrt{\frac{s_1}{s_m} \sum_{j=1}^{m} \frac{s_j}{s_m}} \sqrt{\log m})$, where $s_1 \ge s_2 \ge \cdots \ge s_m$ [10].

Koutsoupias and Papadimitriou [10] conjectured also that the price of anarchy for *m identical machines* is $\Theta(\frac{\log m}{\log \log m})$. In the quest to resolve this conjecture, Mavronicolas and Spirakis [12] considered the problem in the so-called fully mixed model, which is a special class of Nash equilibria in which all p_i^j are strictly positive. In this model, Mavronicolas and Spirakis [12] showed that for m identical machines in the fully mixed Nash equilibrium, the price of anarchy is $\Theta(\frac{\log m}{\log \log m})$. Similarly, they proved also that for m (not necessarily identical) machines and n identical weights in the fully mixed Nash equilibrium, if $m \le n$, then the price of anarchy is $\Theta(\frac{\log n}{\log \log n})$.

The motivation behind studying fully mixed equilibria is the so-called fully mixed Nash equilibrium conjecture stating that these equilibria maximize the price of anarchy because they maximize the randomization. The conjecture seems to be quite appealing as a fully mixed equilibrium can be computed in polynomial time, which led to numerous studies of this kind of equilibria with the hope to obtain efficient algorithms for computing or approximating the price of anarchy with respect to mixed equilibria. However, Fischer and Vöcking [6] disproved the fully mixed Nash equilibrium conjecture and showed that there is a mixed Nash equilibrium whose expected cost is larger than the expected cost of the fully mixed Nash equilibrium by a factor of $\Omega(\frac{\log m}{\log \log m})$. Furthermore, they presented polynomial time algorithms for approximating the price of anarchy for mixed equilibria on identical machines up to a constant factor.

Tight Bounds for the Price of Anarchy

Czumaj and Vöcking [4] entirely resolved the conjecture of Koutsoupias and Papadimitriou [10] and gave an exact description of the price of anarchy as a function of the number of machines and the ratio of the speed of the fastest machine over the speed of the slowest machine. (To simplify the notation, for any real $x \ge 0$, let $\log x$ denote $\log x = \max\{\log_2 x, 1\}$. Also, following standard convention, $\Gamma(N)$ is used to denote the *gamma (factorial) function*, which for any natural N is defined by $\Gamma(N+1) = N!$ and for an arbitrary real $x > 0$ is $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt$. For the inverse of the gamma function, $\Gamma^{(-1)}(N)$, it is known that $\Gamma^{(-1)}(N) = x$ such that $\lfloor x \rfloor ! \le N - 1 \le \lceil x \rceil !$. It is well known that $\Gamma^{(-1)}(N) = \frac{\log N}{\log \log N}(1 + o(1))$.)

Theorem 1 ([4] Upper bound)

The price of anarchy for m machines is bounded from above by

$$\mathcal{O}\left(\min\left\{\frac{\log m}{\log\log\log m}, \frac{\log m}{\log\left(\frac{\log m}{\log(s_1/s_m)}\right)}\right\}\right),$$

where it is assumed that the speeds satisfy $s_1 \ge \cdots \ge s_m$.

In particular, the price of anarchy for m machines is $\mathcal{O}\left(\frac{\log m}{\log\log\log m}\right)$.

The theorem follows directly from the following two results [4]: that the maximum expected load $\mathfrak{c}$ satisfies $\mathfrak{c} = \mathfrak{opt} \cdot \Gamma^{(-1)}(m) = \mathfrak{opt} \cdot \mathcal{O}\left(\min\left\{\frac{\log m}{\log\log m}, \log\left(\frac{s_1}{s_m}\right)\right\}\right)$ and that the social cost C satisfies $\mathsf{C} = \mathfrak{opt} \cdot \mathcal{O}\left(\frac{\log m}{\log\left(\frac{\mathfrak{opt}\cdot\log m}{c}\right)} + 1\right)$.

If one applied these results to systems in which all agents follow only *pure strategies*, then since then $\ell_j = C_j$ for every j, it holds that $\mathsf{C} = \mathfrak{c}$. This leads to the following result.

Corollary 1 ([4]) *For pure strategies the price of anarchy for m machines is upper bounded by*

$$\mathcal{O}\left(\min\left\{\frac{\log m}{\log\log m}, \log\left(\frac{s_1}{s_m}\right)\right\}\right),$$

where it is assumed that the speeds satisfy $s_1 \geq \cdots \geq s_m$.

Theorem 3 below proves that this corollary gives an asymptotically tight bound for the price of anarchy for pure strategies.

By Theorem 1, in the special case when all *machines are identical*, the price of anarchy is $\mathcal{O}\left(\frac{\log m}{\log\log m}\right)$; this result has been also obtained independently by Koutsoupias et al. [11]. However, in this special case one can get a stronger bound that is tight up to an additive constant.

Theorem 2 ([4]) *For m identical machines the price of anarchy is at most*

$$\Gamma^{(-1)}(m) + \Theta(1) = \frac{\log m}{\log\log m} \cdot (1 + o(1)).$$

One can obtain a lower bound for the price of anarchy for *m identical machines* by considering the system in which $p_i^j = \frac{1}{m}$ for every i, j. Gonnet [9] proved that then the price of anarchy is $\Gamma^{(-1)}(m) - \frac{3}{2} + o(1)$, which implies that Theorem 2 is tight up to an additive constant.

The next theorem shows that the upper bound in Theorem 1 is asymptotically tight.

Theorem 3 ([4] Lower bound) *The price of anarchy for m machines is lower bounded by*

$$\Omega\left(\min\left\{\frac{\log m}{\log\log\log m}, \frac{\log m}{\log\left(\frac{\log m}{\log(s_1/s_m)}\right)}\right\}\right).$$

In particular, the price of anarchy for m machines is $\Omega\left(\frac{\log m}{\log\log\log m}\right)$.

In fact, it can be shown [4] (analogously to the upper bound) that for every positive integer m, positive real r, and $S \geq 1$, there exists a set of m machines with $\frac{s_1}{s_m} = S$ being in a Nash equilibrium and satisfying $\mathfrak{opt} = r$, $\mathfrak{c} = \mathfrak{opt} \cdot \Omega\left(\min\left\{\frac{\log m}{\log\log m}, \log\left(\frac{s_1}{s_m}\right)\right\}\right)$, and $C = \mathfrak{opt} \cdot \Omega\left(\frac{\log m}{\log\left(\frac{\mathfrak{opt}\cdot\log m}{c}\right)}\right)$.

Applications

The model discussed here has been extended in the literature in numerous ways, in particular in [1,5,8]; see also survey presentations in [3,14].

Open Problems

An interesting attempt that adds an algorithmic or constructive element to the analysis of the price of anarchy is made in [2]. The idea behind "coordination mechanisms" is not to study the price of anarchy for a fixed system, but to design the system in such a way that the increase in cost or the loss in performance due to selfish behavior is as small as possible. This is a promising direction of research that might result in practical guidelines of how to build a distributed system that does not suffer from selfish behavior but might even exploit the selfishness of the agents.

Experimental Results

None is reported.

URLs to Code and Data Sets

None is reported.

Cross-References

- Computing Pure Equilibria in the Game of Parallel Links
- Price of Anarchy

Recommended Reading

1. Awerbuch B, Azar Y, Richter Y, Tsur D (2006) Tradeoffs in worst-case equilibria. Theor Comput Sci 361(2–3):200–209
2. Christodoulou G, Koutsoupias E, Nanavati A (2009) Coordination mechanisms. Theor Comput Sci 410(36):3327–3336
3. Czumaj A (2004) Selfish routing on the Internet. In: Leung J (ed) Handbook of scheduling: algorithms, models, and performance analysis. CRC, Boca Raton
4. Czumaj A, Vöcking B (2007) Tight bounds for worst-case equilibria. ACM Trans Algorithms 3(1):Article 4
5. Czumaj A, Krysta P, Vöcking B (2010) Selfish traffic allocation for server farms. SIAM J Comput 39(5):1957–1987
6. Fischer S, Vöcking B (2007) On the structure and complexity of worst-case equilibria. Theor Comput Sci 378(2):165–174
7. Fotakis D, Kontogiannis S, Koutsoupias E, Mavronicolas M, Spirakis P (2009) The structure and complexity of Nash equilibria for a selfish routing game. Theor Comput Sci 410(36):3305–3326
8. Gairing M, Lücking T, Mavronicolas M, Monien B (2006) The price of anarchy for polynomial social cost. Theor Comput Sci 369(1–3):116–135
9. Gonnet G (1981) Expected length of the longest probe sequence in hash code searching. J Assoc Comput Mach 28(2):289–304
10. Koutsoupias E, Papadimitriou CH (1999) Worst-case equilibria. In: Proceeding of the 16th annual symposium on theoretical aspects of computer science (STACS), Trier, pp 404–413
11. Koutsoupias E, Mavronicolas M, Spirakis P (2003) Approximate equilibria and ball fusion. Theory Comput Syst 36(6):683–693
12. Mavronicolas M, Spirakis P (2001) The price of selfish routing. In: Proceeding of the 33rd annual ACM symposium on theory of computing (STOC), Heraklion, pp 510–519
13. Nash JF Jr (1951) Non-cooperative games. Ann Math 54(2):286–295
14. Vöcking B (2007) Selfish load balancing. In: Nisan N, Roughgarden T, Tardos É, Vazirani V (eds) Algorithmic game theory. Cambridge University Press, Cambridge

Privacy Preserving Auction

Zhiyi Huang
Department of Computer Science, The University of Hong Kong, Hong Kong, Hong Kong

Keywords

Differential privacy; Mechanism design

Years and Authors of Summarized Original Work

2012; Huang, Kannan
2014; Hsu, Huang, Roth, Roughgarden, Wu

Problem Definition

Let there be n agents and a set of feasible outcomes Ω. For concreteness, readers may think of Ω as the set of allocations of m items to n agents. Each agent has a private value function $v_i : \Omega \mapsto [0, 1]$ over feasible outcomes. We focus on direct revelation mechanisms, which first let each agent i report a value function $\bar{v}_i$, then choose a feasible outcome $\omega \in \Omega$ and a payment p_i for each agent i according to the reported value functions. Let $\omega(\bar{v})$ and $p(\bar{v})$ denote the outcome and payment vector chosen by the mechanism. Note that both $\omega(\bar{v})$ and $p(\bar{v})$ may be random variables.

We hope to achieve the following three objectives:

Objective 1: Maximizing Social Welfare

The social welfare of a feasible outcome $\omega \in \Omega$ is the sum of the agents' values for the outcome, namely, $\sum_{i=1}^{n} v_i(\omega)$. We hope to approximately maximize the expected social welfare of the chosen outcome over the randomness of the mechanism, a widely considered objective in mechanism design.

Approximately maximizing social welfare given the true value functions is a well-studied algorithmic problem (e.g., [4, 16]). In our mechanism design setting, agents may choose

not to report the true value if it fits their interests. So the mechanism has an additional challenge of motivating the agents to report truthfully.

Objective 2: Incentive Compatibility

We adopt the standard assumption that each agent i aims to maximize the expectation of his quasi-linear utility, which equals his value for the chosen outcome less his payment. A mechanism is *incentive compatible* if truth telling maximizes an agent's expected utility regardless of the reported values of other agents, that is, for any agent i, any true value v_i, reported value $\bar{v}_i$, and any reported values of other agents $\bar{v}_{-i}$, we have $\mathbb{E}[v_i(\omega(v_i, \bar{v}_{-i})) - p_i(v_i, \bar{v}_{-i})] \geq \mathbb{E}[v_i(\omega(\bar{v}_i, \bar{v}_{-i})) - p_i(\bar{v}_i, \bar{v}_{-i})]$. We also consider a relaxed notion called α-*incentive compatibility*, where an agent's expected utility of truth telling can be worst off by at most an α additive factor comparing to his utility of reporting any alternative value.

There is a vast literature on designing incentive compatible mechanisms with approximately optimal social welfare (see, e.g., [11] for a comprehensive survey). We remark the Vickrey-Clarke-Groves (VCG) mechanism [2, 5, 15], which chooses an outcome that maximizes the social welfare and uses payments to align the interests of the agents and the mechanism designer. When computational efficiency is not of concern, the VCG mechanism gives optimal social welfare and is incentive compatible for arbitrary problems. However, it does not achieve the next objective.

Objective 3: Protecting Agents' Privacy

Our last objective is to protect the agents' privacy by ensuring that the chosen outcome and payments do not reveal too much information about any individual agent' private value function. Agents may care about their privacy for both exogenous and endogenous reasons. On the one hand, privacy is a basic desideratum. On the other hand, violating an agent's privacy could explicitly hurt the agent's utility in the future, e.g., companies may post higher reserve prices based on an agent's past values if such information is revealed by previous mechanisms.

Definition 1 ([3]) A mechanism is (ϵ, δ)-differentially private if for any agent i, any value v_i, alternative value v_i', any values of other agents v_{-i}, and any subset of feasible outcome $S \subseteq \Omega$,

$$\Pr[\omega(v_i, v_{-i}) \in S] \leq e^{\epsilon} \cdot \Pr[\omega(v_i', v_{-i}) \in S] + \delta \ .$$

We remark that the payments may violate the agents' privacy as well. Nevertheless, we can make the prices privacy preserving without changing the agents' expected utilities by adding any scale of noise with expectation zero to the payments. (Having arbitrarily large variance in the payment and, thus, in the utility of an agent is an undesirable property. In some settings, it is possible to privately compute prices without having large variance. Readers are referred to Hsu et al. [6] for details, which we will omit due to space constraint.) So we focus on the privacy property of the outcome in the above definition.

We provide two informal interpretations of differential privacy (for sufficiently small δ). Information theoretically, a mechanism being (ϵ, δ)-differentially private implies that the outcome reveals at most $O(\epsilon^2)$ bits of information about any individual agent's private value. Game theoretically, it implies that truth telling may decrease an agent's future utility by at most a factor of $e^{-\epsilon} \approx 1 - \epsilon$.

For some mechanism design problems, such as auctioning m items to n agents, no (ϵ, δ)-differentially private mechanisms can approximately maximize social welfare. For such resource allocation problems, let ω_{-i} denote the allocation to all agents except agent i. We consider the following relaxed notion of privacy:

Definition 2 ([8]) A mechanism is (ϵ, δ)-jointly differentially private if for any agent i, any value v_i, alternative value v_i', any values of other agents v_{-i}, and any subset of feasible outcome $S \subseteq \Omega$,

$$\Pr[\omega_{-i}(v_i, v_{-i}) \in S] \leq e^{\epsilon} \cdot \Pr[\omega_{-i}(v_i', v_{-i}) \in S] + \delta \ .$$

In settings where each agent can see only his own allocation, a mechanism being (ϵ, δ)-jointly

differentially private also implies that it reveals at most $O(\epsilon^2)$ bits of information of an agent's value and that truth telling decreases an agent's future utility by at most $e^{-\epsilon}$, even if the adversary colludes with all other agents.

Related Work

The problem we consider in this article falls into the growing literature on the interface of game theory and differential privacy (see, e.g., Pai and Roth [14] for a survey). McSherry and Talwar [10] proposed using differentially private mechanisms to design auctions by pointing out that differential privacy implies approximate incentive compatibility and resilience to collusion. They also proposed the exponential mechanism, which is an important building block in one of the results we will discuss. Nissim et al. [13] showed how to convert differentially private mechanisms into exactly incentive compatible mechanisms in some settings, but the final mechanisms no longer protect agents' privacy. Xiao [17] proposed mechanisms that are both incentive compatible and differentially private in some special cases. Unfortunately, it does not seem possible to extend the results of Nissim et al. [13] and Xiao [17] to more general problems. Finally, Xiao [17], Chen et al. [1], and Nissim et al. [12] considered modeling the agents' concern for privacy in the utility functions and introduced incentive compatible mechanisms for some special cases in this model. In sum, most previous techniques apply only to special cases. In this article, we summarize two recent techniques for designing privacy-preserving auctions for a large family of mechanism design problems.

Key Results

Almost all mechanism design problems can be classified into two families: social choice problems and resource allocation problems. In a social choice problem, the set of feasible outcome is independent of the number of agents n. In particular, the number of feasible outcome is independent of n. For example, leader elections and choosing a subset of public projects subject a budget constraint fall into this family. In a resource allocation problem, such as allocating m items to n agents, the set of feasible outcome depends on the number of agents. In particular, the number of feasible outcome grows exponentially with n. Below we discuss two techniques by Huang and Kannan [7] and Hsu et al. [6] for designing privacy-preserving auctions for these two families of problems, respectively.

Social Choice Problems

Huang and Kannan [7] proposed a technique for designing incentive compatible and ϵ-differentially private mechanisms for arbitrary mechanism design problems. For social choice problems, in particular, this technique also gives nearly optimal social welfare.

Theorem 1 ([7]) *For any mechanism design problem, there is an incentive compatible and ϵ-differentially private mechanism that gives at least* $\mathrm{OPT} - \frac{2}{\epsilon}(\ln|\Omega| + \ln\frac{1}{\beta})$ *social welfare with probability at least* $1-\beta$.

This mechanism is based on the exponential mechanism by McSherry and Talwar [10], a general differentially private mechanism that can be applied to a large family of problems. The social welfare guarantee and ϵ-differential privacy in Theorem 1 follow directly from properties of the exponential mechanism. However, the exponential mechanism is not incentive compatible in general. Huang and Kannan [7] noticed that the exponential mechanism can be viewed as maximizing a linear combination of the social welfare and the Shannon entropy of the outcome distribution. Therefore, its allocation rule is equivalent to that of the VCG mechanism in a virtual market where the set of feasible outcomes are distributions over the original outcomes, the set of agents are the original n agents plus an additional agent whose value equals the entropy of the chosen distribution. As a result, using the payments in the virtual market along with the exponential mechanism achieves incentive compatibility.

In social choice problems, $\ln|\Omega|$ is a constant independent of n. So the loss in social welfare is a constant independent of n. On the other hand,

the social welfare is the sum of n agents' values, each of which is between 0 and 1. Hence, in a large market with many agents, it is reasonable to expect the optimal social welfare (if not of scale $\Theta(n)$) to be much larger than the additive loss in Theorem 1 in practical instances.

In resource allocation problems, however, $|\Omega|$ grows exponentially in n and, thus, $\ln|\Omega|$ is of scale $\Omega(n)$. For instance, consider matching $m = n$ items to n agents. Then, $|\Omega| = n!$ and $\ln|\Omega| = \Omega(n \ln n)$. Even if the optimal social welfare is of scale $\Theta(n)$, we would need ϵ to be at least $\Omega(\ln n)$ to have nontrivial social welfare guarantee in Theorem 1. This means that the mechanism would reveal $\Omega(\ln^2 n)$ bits of information of an agent's private value, and truth telling may decrease an agent's future utility by a poly(n) factor. Further, this is not only a limitation of the current technique. Huang and Kannan [7] showed that no ϵ-differentially private can give nontrivial social welfare guarantee for $\epsilon = o(\ln n)$, even without incentive compatibility.

Resource Allocation Problems

Given the obstacles for applying differential privacy to resource allocation problems, Hsu et al. [6] looked into a relaxed notion of privacy, namely, joint differential privacy. In particular, they considered matching m items to n agents where each item has a supply of at least s copies and then generalized the results to combinatorial auctions with gross substitute value functions (e.g., [9]). Their first result is a jointly differentially private (yet not incentive compatible) mechanism with nearly optimal social welfare when the supply s is poly-logarithmic in n and m. Their main technique is a noisy variant of the deferred-acceptance algorithm by Kelso and Crawford [9].

Theorem 2 ([6]) *For combinatorial auctions with gross substitute valuations, there is an ϵ-jointly differentially private algorithm that gives at least $OPT - \alpha n$ social welfare with probability at least $1-\beta$ if $s = \Omega(\frac{1}{\epsilon\alpha^3}\text{polylog}(n, m, \frac{1}{\alpha}, \frac{1}{\beta}))$.*

They also showed a supply of $\omega(1)$ is needed for a jointly differentially private mechanism to achieve $o(n)$ additive loss in social welfare. More precisely, they showed:

Theorem 3 ([6]) *No jointly differentially private algorithm can compute matchings with social welfare at least $OPT - \alpha n$ if $s \le O(\frac{1}{\sqrt{\alpha}})$.*

Their approach can also be used to design approximately incentive compatible and jointly differentially private mechanisms, but the supply needs to be polynomially large.

Theorem 4 (implicit in [6]) *For combinatorial auctions with gross substitute valuations, there is an α-incentive compatible and ϵ-jointly differentially private algorithm that gives at least $OPT - \alpha n$ social welfare with probability at least $1-\beta$ if $s = \Omega(m)$, where the constant depends on ϵ, δ, α, and β.*

Open Problems

The results of Huang and Kannan [7] and Hsu et al. [6] provided a preliminary step towards designing auctions that protect agents' privacy. There are still many open problems in this topic, some of which we sketch below.

First, the techniques of Hsu et al. [6] fundamentally rely on properties of gross substitute value functions and, thus, cannot be extended to more general value functions. Further, there are many important families of value functions beyond gross substitute, e.g., sub-modular functions, sub-additive functions, etc. So it is natural to seek for techniques that work for more general families of value functions.

Problem 1 Are there jointly differentially private mechanisms that achieve nearly optimal social welfare for arbitrary value functions?

Even if we restrict our attention to gross substitute value functions or even to matching markets, Theorems 2 and 3 leave a large gap between the upper and lower bounds on the supply needed by jointly differentially private mechanisms to get nearly optimal social welfare. Closing this gap would advance our understanding on joint differential privacy.

Problem 2 What is the minimal supply needed so that a jointly differentially private mechanism can achieve nearly optimal social welfare in combinatorial auctions? In particular, is the logarithmic dependency on n and m in Theorem 2 necessary?

Finally, the current technique for achieving both approximate incentive compatibility and joint differential privacy requires a polynomially large supply of items, much larger than the supply needed for achieving joint differential privacy alone. Does approximate incentive compatibility make the problem fundamentally harder? Or is it just a limitation of the current technique?

Problem 3 What is the minimal supply needed so that an approximately incentive compatible and jointly differentially private mechanism can achieve nearly optimal social welfare in combinatorial auctions? In particular, is the polynomial dependency on m in Theorem 4 necessary?

Cross-References

▸ Mechanism Design and Differential Privacy

Recommended Reading

1. Chen Y, Chong S, Kash IA, Moran T, Vadhan S (2013) Truthful mechanisms for agents that value privacy. In: 14th conference on electronic commerce. ACM, New York, pp 215–232
2. Clarke EH (1971) Multipart pricing of public goods. Public Choice 11(1):17–33
3. Dwork C, McSherry F, Nissim K, Smith A (2006) Calibrating noise to sensitivity in private data analysis. In: Theory of cryptography. Springer, Berlin/Heidelberg, pp 265–284
4. Feige U (2009) On maximizing welfare when utility functions are subadditive. SIAM J Comput 39(1):122–142
5. Groves T (1973) Incentives in teams. Econom J Econom Soc 41:617–631
6. Hsu J, Huang Z, Roth A, Roughgarden T, Wu SZ (2014) Private matchings and allocations. In: 46th annual symposium on theory of computing (STOC). ACM, New York
7. Huang Z, Kannan S (2012) The exponential mechanism for social welfare: private, truthful, and nearly optimal. In: 53rd annual symposium on foundations of computer science (FOCS). IEEE, Washington, DC, pp 140–149
8. Kearns M, Pai M, Roth A, Ullman J (2014) Mechanism design in large games: incentives and privacy. In: 5th conference on innovations in theoretical computer science. ACM, New York, pp 403–410
9. Kelso AS, Crawford VP (1982) Job matching, coalition formation, and gross substitutes. Econom J Econom Soc 50:1483–1504
10. McSherry F, Talwar K (2007) Mechanism design via differential privacy. In: 48th annual symposium on foundations of computer science (FOCS). IEEE, pp 94–103
11. Nisan N, Roughgarden T, Tardos E, Vazirani VV (2007) Algorithmic game theory. Cambridge University Press, Cambridge/New York
12. Nissim K, Orlandi C, Smorodinsky R (2012) Privacy-aware mechanism design. In: 13th conference on electronic commerce. ACM, New York, pp 774–789
13. Nissim K, Smorodinsky R, Tennenholtz M (2012) Approximately optimal mechanism design via differential privacy. In: 3rd conference on innovations in theoretical computer science. ACM, New York, pp 203–213
14. Pai MM, Roth A (2013) Privacy and mechanism design. ACM SIGecom Exch 12(1):8–29
15. Vickrey W (1961) Counterspeculation, auctions, and competitive sealed tenders. J Financ 16(1):8–37
16. Vondrák J (2008) Optimal approximation for the submodular welfare problem in the value oracle model. In: 40th annual symposium on theory of computing. ACM, New York, pp 67–74
17. Xiao D (2013) Is privacy compatible with truthfulness? In: 4th conference on innovations in theoretical computer science. ACM, New York, pp 67–86

Private Spectral Analysis

Moritz Hardt
IBM Research – Almaden, San Jose, CA, USA

Keywords

Differential privacy; Power method; Singular value decomposition; Spectral analysis

Years and Authors of Summarized Original Work

2014; Dwork, Talwar, Thakurta, Zhang
2014; Hardt, Price

Problem Definition

Spectral analysis refers to a family of popular and effective methods that analyze an input matrix by exploiting information about its eigenvectors or singular vectors. Applications include principal component analysis, low-rank approximation, and spectral clustering. Many of these applications are commonly performed on data sets that feature sensitive information such as patient records in a medical study. In such cases privacy is a major concern. Differential privacy is a powerful general-purpose privacy definition. This entry explains how differential privacy may be applied to task of approximately computing the top singular vectors of a matrix.

Generally speaking, the input is a real-valued matrix $A \in \mathbb{R}^{m \times n}$ and a parameter $k \in \mathbb{N}$. We think of the input matrix as specifying n attributes for m individuals. The goal of the algorithm is to approximately compute the first $k \leq \min\{m, n\}$ singular vectors of A while achieving differential privacy. There are several notions of approximation as well as several variants of differential privacy that make sense in this context.

Approximation Guarantee

Let $A = U\Sigma V^\top$ denote the singular value decomposition of A with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\max\{m,n\}} \geq 0$. Further, let U_k and V_k represent the first k columns of U and V, respectively. In other words, U_k consists of the first k left singular vectors of A and V_k consists of the first k right singular vectors.

Principal Angle

Principal angles are a useful tool for comparing the distance between subspaces. The sine of the largest principal angle between subspaces X, Y of equal dimension represented by orthonormal matrices is defined as $\sin\theta(X, Y) = \left\|(I - XX^\top)Y\right\|_2$, where the norm refers to the spectral norm (or ℓ_2-operator norm).

A natural objective is to require that the algorithm $\mathcal{M}$ outputs an orthonormal matrix $X \in \mathbb{R}^{m \times k}$ so as to minimize $\left\|(I - XX^\top)V_k\right\|$. We call this the *principal angle objective*. The angle is of course 0 when $X = V_k$. We will also be interested in the case where the rank of X is larger than the rank of V_k. Note that our objective is still well defined.

Expressed Variance

Another natural objective is to output an orthonormal matrix $X \in \mathbb{R}^{m \times k}$ so as to maximize the variance captured by the subspace spanned by the columns of X. A convenient way to express this objective is to maximize the quantity $\|AX\|_F^2$, where the norm refers to the Frobenius norm. It is not difficult to show that this objective is maximized for $X = V_k$. Again, the objective is still well defined when the rank of X is larger than that of X.

Privacy Guarantee

Differential privacy requires the definition of a *neighborhood relation* on matrices, denoted $A \sim A'$. Pairs of matrices in this relation are called *neighboring*. Differential privacy requires that the algorithm maps neighboring databases to nearly indistinguishable output distributions.

Definition 1 Given a neighborhood relation $\sim$, we say that a randomized algorithm $\mathcal{M}$ satisfies (ϵ, δ)-differential privacy if for all neighboring matrices $A \sim A'$ and for every measurable set S in the output space of the algorithm, we have that

$$\Pr\{\mathcal{M}(A) \in S\} \leq \exp(\epsilon)\Pr\left\{\mathcal{M}(A') \in S\right\} + \delta\,. \tag{1}$$

Neighborhood Relations. Typically in differential privacy, the neighborhood relation is chosen to be the set of all pairs of matrices that differ in at most one row. Unfortunately, this definition is unattainable in the spectral setting as the privacy definition is sensitive to the scale of the row vector that's being changed. Indeed, if we replace a single row of the matrix by a vector Δu of norm Δ, then as we let Δ tend to infinity, the top right singular vector of the matrix will tend to the vector u (say, in angle).

To circumvent this problem, we will generally specify a norm bound in each neighbor relation. It is important to note that the strength of privacy

definition then depends on the scaling of the matrix.

- We say that A, A' are *entry neighbors* if they differ by at most 1 in absolute value.
- We say that A, A' are *row neighbors* if they differ in at most one row by a vector whose Euclidean norm is bounded by 1.

All entry neighbors are of course also row neighbors so that the privacy definition based on row neighbors is stronger than that of entry neighbors. It is sometimes natural to scale the matrix such that either all entries have magnitude at most 1 or all rows have Euclidean norm at most 1. While this may strengthen the privacy guarantee, it also leads to a corresponding deterioration in the utility guarantee of the algorithm as the signal-to-noise ratio decreases. It is tempting to nonuniformly scale each row by a different factor. However, this can dramatically change the singular vector decomposition and does not in general lead to an easily interpretable guarantee.

Key Results

We describe two simple and effective methods that lead to nearly optimal approximation guarantees in various settings we introduced above. The first algorithm is based on the well-known power method. The other uses a simple Gaussian noise addition step (Fig. 1).

Noisy Power Method

For simplicity we describe the algorithm in the case where A is a symmetric $n \times n$ matrix. The algorithm extends straightforwardly to rectangular and asymmetric matrices as explained in [5]. We first state a general-purpose analysis of PPM.

Theorem 1 ([3]) *Let $k \leq p$. Then, the private power method satisfies (ϵ, δ)-differential privacy under the entry neighbor relation, and after $L = O(\frac{\sigma_k}{\sigma_k - \sigma_{k+1}} \log(n))$ iterations, we have with probability $9/10$ that*

$$\left\| (I - X_L X_L^\top) V_k \right\| \leq O\Big(\frac{\sigma \max_{\ell=0}^{L} \|X_\ell\|_\infty \sqrt{n \log L}}{\sigma_k - \sigma_{k+1}} \cdot \frac{\sqrt{p}}{\sqrt{p} - \sqrt{k-1}} \Big).$$

When $p = k + \Omega(k)$, the trailing factor becomes a constant. If $p = k$, it creates a factor k overhead. In the worst case we can always bound $\|X_\ell\|_\infty$ by 1 since X_ℓ is an orthonormal basis. However, in principle, we could hope that a much better bound holds provided that the target basis V_k has small coordinates. Hardt and Roth [4, 5] suggested a way to accomplish a stronger bound by considering a notion of *coherence* of A, denoted as $\mu(A)$. The coherence parameter varies between 1 and n but is often sublinear in n. Intuitively, the coherence measures the correlation between the singular vectors of the matrix with the standard basis. Low coherence means that the singular vectors have small coordinates in the standard basis. Many results on matrix completion and robust PCA crucially rely on such an assumption though the exact notion is somewhat different here. Specifically, if $A = V \Sigma V^\top$ is a singular vector decomposition of A, we define $\mu(A) \stackrel{\text{def}}{=} n \max_{i,j \in [n} |V_{ij}|^2$.

Theorem 2 ([3]) *Under the assumptions of Theorem 3, we have the conclusion*

Input: Symmetric $A \in \mathbb{R}^{n \times n}$, L, p, privacy parameters $\varepsilon, \delta > 0$
1. Let X_0 be a random orthonormal basis and put $\sigma = \varepsilon^{-1} \sqrt{4pL \log(1/\delta)}$
2. For $\ell = 1$ to L:
   a) $Y_\ell \leftarrow A X_{\ell-1} + G_\ell$ where $G_\ell \sim \mathrm{N}(0, \|X_{\ell-1}\|_\infty^2 \sigma^2)^{n \times p}$.
   b) Compute the QR-factorization $Y_\ell = X_\ell R_\ell$

Output: Matrix X_L

Private Spectral Analysis, Fig. 1 Private power method. Here $\|X\|_\infty = \max_{ij} |X_{ij}|$

Input: Matrix $A \in \mathbb{R}^{m \times n}$, privacy parameters $\varepsilon, \delta > 0$, parameter $p \in \mathbb{N}$
1. Let E be a symmetric matrix where the upper triangle (including the diagonal) is sampled i.i.d. from $N(0, \sigma^2)$ where $\sigma = \sqrt{2\ln(1.25/\delta)}/\varepsilon$
2. $C \leftarrow A^\top A + E$

Output: Top p singular vectors $X \in \mathbb{R}^{n \times p}$ of $\widetilde{C}$

Private Spectral Analysis, Fig. 2 Gaussian mechanism

$$\left\|(I - X_L X_L^\top) V_k\right\| \leq O\left(\frac{\sigma\sqrt{\mu(A)\log n \log L}}{\sigma_k - \sigma_{k+1}} \cdot \frac{\sqrt{p}}{\sqrt{p} - \sqrt{k-1}}\right).$$

Gaussian Mechanism

The Gaussian mechanism first appeared in [1] and was recently revisited [2]. The algorithm simply computes the covariance matrix of the data set and adds suitably scaled (symmetric) Gaussian noise to the covariance matrix. The result is differentially private, and the top singular vectors of the perturbed covariance matrix serve as an approximation of the true singular vectors (Fig. 2).

Theorem 3 ([2]) *Let $k \leq p$. .Then, the Gaussian mechanism satisfies (ϵ, δ)-differential privacy under the row neighbor relation, and with probability $1 - o(1)$, we have $\|AX\|_F^2 \geq \|AV_k\|_F^2 - O\left(\sigma k\sqrt{n}\right)$. Moreover, with probability $1 - o(1)$,*

$$\|AX\|_F^2 \geq \|AV_k\|_F^2 - O\left(\frac{\sigma^2 pn}{\sigma_k^2 - \sigma_{p+1}^2}\right).$$

Applications

Principal Component Analysis

In principal component analysis, the goal is to compute the top k singular vectors of the $n \times n$ matrix $A^\top A$. Recall that we identified data points with row vectors in A. The singular vectors of $A^\top A$ are identical to the right singular vectors of A. Hence, both algorithms we previously discussed immediately solve this problem.

Low-Rank Approximation

In low-rank approximation, the goal is to output a matrix B of rank k such that $\|A - B\|_\nu$ is small, where $\nu \in \{2, F\}$. For either norm, the optimal solution is given by the truncated singular value decomposition $B = U_k \Sigma_k V_k^\top$. In the context of privacy-preserving spectral analysis, a good approximation $\tilde{V}_k$ to V_k typically leads to a good low-rank approximation by performing a privacy-preserving multiplication step $\tilde{\Sigma}_k \tilde{U}_k = A\tilde{V}_k + N$, where N is suitably chosen noise matrix. See, for example, [5].

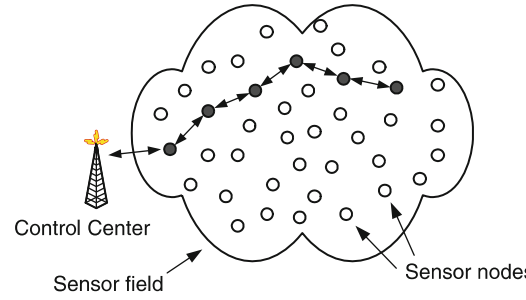
Open Problems

1. Is it possible to obtain an expressed variance guarantee for the noisy power method? For instance, can we match the bounds achieved by Gaussian noise addition via the power method? The problem with the Gaussian mechanism is that it computes the matrix $A^\top A$ which is impractical when n is large but A may be sparse. In this case, the power method is computationally far more efficient.
2. Can we weaken the incoherence assumption in Theorem 2?
3. Theorem 3 depends on the separation between σ_k and σ_{k+1} even when $p > k$. Is it possible to replace the dependence on $\sigma_k - \sigma_{k+1}$ with a dependence on $\sigma_k - \sigma_{p+1}$?

Recommended Reading

1. Blum A, Dwork C, McSherry F, Nissim K (2005) Practical privacy: the SuLQ framework. In: Proceedings of the 24th PODS. ACM, pp 128–138
2. Dwork C, Talwar K, Thakurta A, Zhang L (2014) Analyze gauss: optimal bounds for privacy-preserving

principal component analysis. In: Proceedings of the 46th symposium on theory of computing (STOC). ACM, pp 11–20
3. Hardt M, Price E (2014) The noisy power method: A meta algorithm with applications. CoRR abs/1311.2495v2. http://arxiv.org/abs/1311.2495
4. Hardt M, Roth A (2012) Beating randomized response on incoherent matrices. In: Proceedings of the 44th symposium on theory of computing (STOC). ACM, pp 1255–1268
5. Hardt M, Roth A (2013) Beyond worst-case analysis in private singular vector computation. In: Proceedings of the 45th Symposium on Theory of Computing (STOC). ACM



Probabilistic Data Forwarding in Wireless Sensor Networks, Fig. 1 A sensor network

Probabilistic Data Forwarding in Wireless Sensor Networks

Sotiris Nikoletseas
Computer Engineering and Informatics, University of Patras, Patras, Greece
Computer Technology Institute and Press "Diophantus", Patras, Greece

Keywords

Data propagation; Routing

Years and Authors of Summarized Original Work

2004; Chatzigiannakis, Dimitriou, Nikoletseas, Spirakis

Problem Definition

An important problem in wireless sensor networks is that of *local detection and propagation*, i.e., the local sensing of a crucial event and the energy and time efficient propagation of data reporting its realization to a control center (for a graphical presentation, see Fig. 1). This center (called the *"sink"*) could be some human authorities responsible of taking action upon the realization of the crucial event. More formally:

Definition 1 Assume that a single sensor, E, senses the realization of a *local event* $\mathcal{E}$. Then the *propagation problem* is the following: "How can sensor P, via cooperation with the rest of the sensors in the network, efficiently propagate information reporting the realization of the event to the sink S?"

Note that this problem is in fact closely related to the more general problem of data propagation in sensor networks.

Wireless Sensor Networks

Recent dramatic developments in micro-electro-mechanical systems (MEMS), wireless communications and digital electronics have led to the development of small in size, low-power, low-cost sensor devices. Such extremely small (soon in the cubic millimetre scale) devices integrate sensing, data processing and wireless communication capabilities. Examining each such device individually might appear to have small utility, however the effective *distributed self-organization* of large numbers of such devices into an ad-hoc network may lead to the efficient accomplishment of large sensing tasks. Their wide range of applications is based on the use of various sensor types (i.e., thermal, visual, seismic, acoustic, radar, magnetic, etc.) to monitor a wide variety of conditions (e.g., temperature, object presence and movement, humidity, pressure, noise levels etc.). For a survey on wireless sensor networks see [1] and also [6, 9].

A Simple Model

Sensor networks are comprised of a vast number of ultra-small homogeneous sensors, which are called *"grain" particles*. Each grain particle is a fully-autonomous computing and communication device, characterized mainly by its available power supply (battery) and the energy cost of computation and transmission of data. Such particles (in the model here) do not move. Each particle is equipped with a set of monitors (sensors) for light, pressure, humidity, temperature etc. and has a *broadcast* (digital radio) *beacon mode*.

It is assumed that grain particles are *randomly deployed* in a given area of interest. Such a placement may occur e.g., when throwing sensors from an airplane over an area. A special case is considered, when the network being a lattice (or grid) deployment of sensors. This grid placement of grain particles is motivated by certain applications, where it is possible to have a pre-deployed sensor network, where sensors are put (possibly by a human or a robot) in a way that they form a *2-dimensional lattice*.

It is assumed that each particle has the following abilities: (i) It can estimate the direction of a received transmission (e.g., via the technology of direction-sensing antennae). (ii) It can estimate the distance from a nearby particle that did the transmission (e.g., via estimation of the attenuation of the received signal). (iii) It knows the direction towards the sink S. This can be implemented during a set-up phase, where the (powerful) sink broadcasts the information about itself to all particles. (iv) All particles have a common co-ordinates system. Notice that GPS information is not assumed. Also, there is no need to know the global structure of the network.

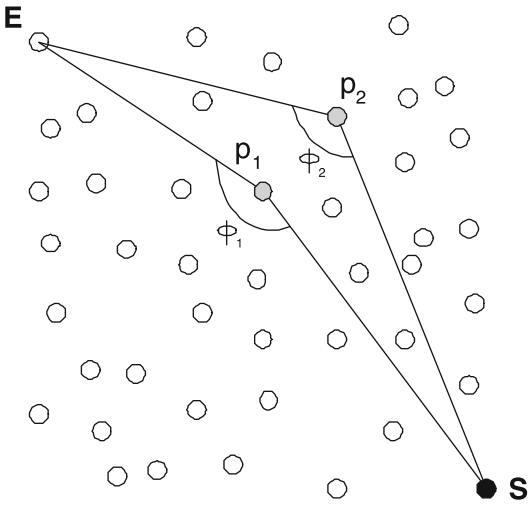
Key Results

The Basic Idea

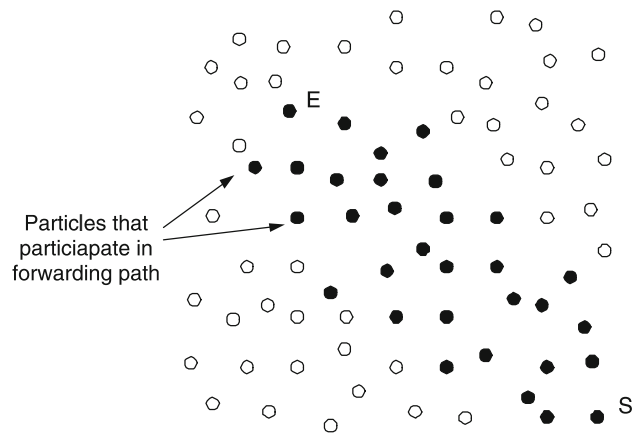
For the above problem [3] proposes a protocol which tries to minimize energy consumption by *probabilistically favoring certain paths of local data transmissions towards the sink*. Thus this protocol is called PFR (Probabilistic Forwarding Protocol). Its basic idea is to *avoid flooding* by favoring (in a probabilistic manner) data propagation along sensors which lie "close" to the (optimal) transmission line, ES, that connects the sensor node detecting the event, E, and the sink, S. This is implemented by locally calculating the angle $\phi = (\widehat{EPS})$, whose corner point P is the sensor currently running the local protocol, having received a transmission from a nearby sensor, previously possessing the event information (see Fig. 2). If φ is equal or greater to a predetermined threshold, then p will transmit (and thus propagate the information further). Else, it decides whether to transmit with probability equal to $\frac{\phi}{\pi}$. Because of the probabilistic nature of data propagation decisions and to prevent the propagation process from early failing, the protocol initially uses (for a short time period which is evaluated) a flooding mechanism that leads to a sufficiently large *"front"* of sensors possessing the data under propagation. When such a "front" is created, probabilistic Forwarding is performed.

The PFR Protocol

The protocol evolves in two phases:



Probabilistic Data Forwarding in Wireless Sensor Networks, Fig. 2 Angle φ and proximity to the optimal line



Probabilistic Data Forwarding in Wireless Sensor Networks, Fig. 3 Thin zone of particles

Phase 1: The "Front" Creation Phase

Initially the protocol builds (by using a limited, in terms of rounds, flooding) a sufficiently large "front" of particles, to guarantee the survivability of the data propagation process. During this phase, each particle having received the data to be propagated, deterministically forwards them towards the sink.

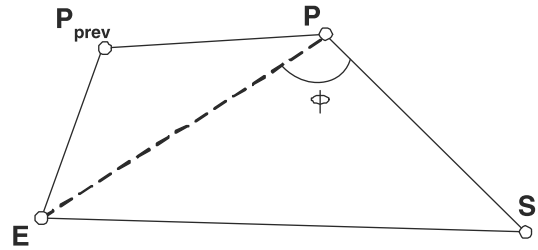
Phase 2: The Probabilistic Forwarding Phase

Each particle P possessing the information under propagation (called $info(\mathcal{E})$ hereafter), calculates an angle φ by calling the subprotocol "φ-calculation" (see description below) and broadcasts $info(\mathcal{E})$ to all its neighbors with probability $\mathbb{P}_{fwd}$ (or it does not propagate any data with probability $1 - \mathbb{P}_{fwd}$) as follows:

$$\mathbb{P}_{fwd} = \begin{cases} 1 & \text{if } \phi \geq \phi_{\text{threshold}} \\ \frac{\phi}{\pi} & \text{otherwise} \end{cases}$$

where φ is the $(\widehat{EPS})$ angle and $\phi_{\text{threshold}} = 134°$ (the selection reasons of this value are discussed in [3]).

If the density of particles is appropriately large, then for a line ES there is (with high probability) a sequence of points "closely surrounding ES" whose angles φ are larger than $\phi_{\text{threshold}}$ and so that successive points are within transmission range. All such points broadcast and thus essentially they follow the line ES (see Fig. 3).



Probabilistic Data Forwarding in Wireless Sensor Networks, Fig. 4 Angle φ calculation example

The φ-calculation Subprotocol (see Fig. 4)

Let P_{prev} the particle that transmitted $info(E)$ to P.

1. When P_{prev} broadcasts $info(E)$, it also attaches the info $|EP_{\text{prev}}|$ and the direction $\overrightarrow{P_{\text{prev}}E}$.
2. P estimates the direction and length of line segment $P_{\text{prev}}P$, as described in the model.
3. P now computes angle $(E\widehat{P_{\text{prev}}}P)$, and computes $|EP|$ and the direction of $\overrightarrow{PE}$ (this will be used in further transmission from P).
4. P also computes angle $(P_{\text{prev}}\widehat{P}E)$ and by subtracting it from $(P_{\text{prev}}\widehat{P}S)$ it finds φ.

Performance Properties of PFR

Any protocol Π solving the data propagation problem must satisfy the following three properties: **(a) Correctness**.Π must guarantee that data arrives to the position S, given that the whole network exists and is operational. **(b) Robustness**.

Π must guarantee that data arrives at enough points in a small interval around S, in cases where part of the network has become inoperative. **(c) Efficiency**. If Π activates k particles during its operation then Π should have a small ratio of the number of activated over the total number of particles $r = \frac{k}{N}$. Thus r is an energy efficiency measure of Π. It is shown that this is indeed the case for PFR.

Consider a partition of the network area into small squares of a fictitious grid G (see Fig. 5). When particle density is high enough, occupancy arguments guarantee that with very high probability (tending to 1) all squares get particles. All the analysis is conditioned on this event, call it F, of at least one particle in each square. Below only sketches of proofs are provided (full proofs can be found in [3]).

The Correctness of PFR

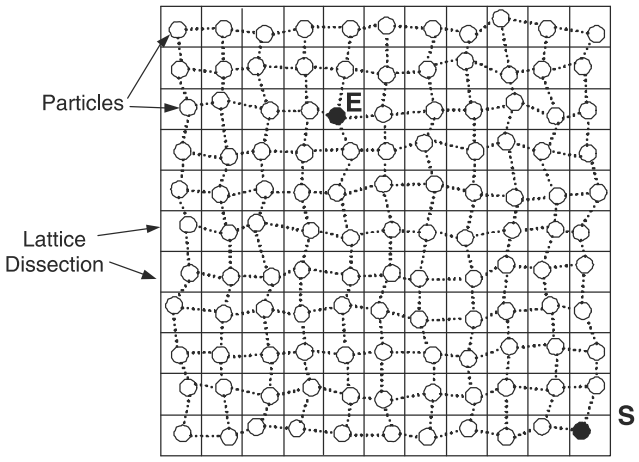
Consider any square Σ intersecting the ES line. By the occupancy argument above, there is w.h.p. a particle in this square. Clearly, the worst case is when the particle is located in one of the corners of Σ (since the two corners located most far away from the ES line have the smallest φ-angle among all positions in Σ). By geometric calculations, [3] proves that the angle φ of this particle is $\phi > 134°$. But the initial square (i.e., that containing E) always broadcasts and any intermediate intersecting square will be notified(by induction) and thus broadcast because of the argument above. Thus the sink will be reached if the whole network is operational:

Lemma 1 ([3]) *PFR succeeds with probability 1 given the event F.*

The Energy Efficiency of PFR

Consider a "lattice-shaped" network like the one in Fig. 5 (all results will hold for any random deployment "in the limit"). The analysis of the energy efficiency considers particles that are active but are as far as possible from ES. [3] estimates an upper bound on the number of particles in an $n \times n$ (i.e., $N = n \times n$) lattice. If k is this number then $r = \frac{k}{n^2}$ $(0 < r \leq 1)$ is the "energy efficiency ratio" of PFR. More specifically, in [3] the authors prove the (very satisfactory) result below. They consider the area around the ES line, whose particles participate in the propagation process. The number of active particles is thus, roughly speaking, captured by the size of this area, which in turn is equal to $|ES|$ times the maximum distance from $|ES|$. This maximum distance is clearly a random variable. To calculate the expectation and variance of this variable, the authors in [3] basically "upper bound" the stochastic process of the distance from ES by a random walk on the line, and subsequently "upper bound" this random walk by a well-known stochastic process (i.e., the "discouraged arrivals"

Probabilistic Data Forwarding in Wireless Sensor Networks, Fig. 5
A lattice dissection G



birth and death Markovian process. Thus they prove:

Theorem 2 ([3]) *The energy efficiency of the PFR protocol is* $\Theta\left(\left(\frac{n_0}{n}\right)^2\right)$ *where* $n_0 = |ES|$ *and* $n = \sqrt{N}$*, where N is the number of particles in the network. For* $n_0 = |ES| = o(n)$*, this is o(1).*

The Robustness of PFR

Consider particles "very near" to the *ES* line. Clearly, such particles have large φ-angles (i.e., $\phi > 134°$). Thus, even in the case that some of these particles are not operating, the probability that none of those operating transmits (during phase 2) is very small. Thus:

Lemma 3 ([3]) PFR *manages to propagate the crucial data across lines parallel to ES, and of constant distance, with fixed nonzero probability (not depending on n,* $|ES|$*).*

Applications

Sensor networks can be used for continuous sensing, event detection, location sensing as well as micro-sensing. Hence, sensor networks have several important applications, including (a) security (like biological and chemical attack detection), (b) environmental applications (such as fire detection, flood detection, precision agriculture), (c) health applications (like telemonitoring of human physiological data) and (d) home applications (e.g., smart environments and home automation). Also, sensor networks can be combined with other wireless networks (like mobile) or fixed topology infrastructures (like the Internet) to provide transparent wireless extensions in global computing scenaria.

Open Problems

It would be interesting to come up with formal models for sensor networks, especially with respect to energy aspects; in this respect, [10] models energy dissipation using stochastic methods. Also, it is important to investigate fundamental trade-offs, such as those between energy and time. Furthermore, the presence of mobility and/or multiple sinks (highly motivated by applications) creates new challenges (see e.g., [2, 11]). Finally, heterogeneity aspects (e.g., having sensors of various types and/or combinations of sensor networks with other types of networks like p2p, mobile and the Internet) are very important; in this respect see e.g., [5, 13].

Experimental Results

An implementation of the PFR protocol along with a detailed comparative evaluation (using simulation) with greedy forwarding protocols can be found in [4]; with clustering protocols (like LEACH, [7]) in [12]; with tree maintenance approaches (like Directed Diffusion, [8]) in [5]. Several performance measures are evaluated, like the success rate, the latency and the energy dissipation. The simulations mainly suggest that PFR behaves best in sparse networks of high dynamics.

Cross-References

- Communication in Ad Hoc Mobile Networks Using Random Walks
- Obstacle Avoidance Algorithms in Wireless Sensor Networks
- Randomized Energy Balance Algorithms in Sensor Networks

Recommended Reading

1. Akyildiz IF, Su W, Sankarasubramaniam Y, Cayirci E (2002) Wireless sensor networks: a survey. J Comput Netw 38:393–422
2. Chatzigiannakis I, Kinalis A, Nikoletseas S (2006) Sink mobility protocols for data collection in wireless sensor networks. In: Proceedings of the 4th ACM/IEEE international workshop on mobility management and wireless access protocols (MobiWac). ACM, pp 52–59
3. Chatzigiannakis I, Dimitriou T, Nikoletseas S, Spirakis P (2004) A probabilistic algorithm for efficient

and robust data propagation in smart dust networks. In: Proceedings of the 5th European wireless conference on mobile and wireless systems (EW 2004), pp 344–350. Also in: Ad-Hoc Netw J 4(5):621–635 (2006)
4. Chatzigiannakis I, Dimitriou T, Mavronicolas M, Nikoletseas S, Spirakis P (2003) A comparative study of protocols for efficient data propagation in smart dust networks. In: Proceedings of the 9th European symposium on parallel processing (EuroPar), distinguished paper. Lecture notes in computer science, vol 2790. Springer, pp 1003–1016. Also in the Parall Process Lett (PPL) J 13(4):615–627 (2003)
5. Chatzigiannakis I, Kinalis A, Nikoletseas S (2005) An adaptive power conservation scheme for heterogeneous wireless sensors. In: Proceedings of the 17th annual ACM symposium on parallelism in algorithms and architectures (SPAA 2005). ACM, pp 96–105. Also in: Theory Comput Syst (TOCS) J 42(1):42–72 (2008)
6. Estrin D, Govindan R, Heidemann J, Kumar S (1999) Next century challenges: scalable coordination in sensor networks. In: Proceedings of the 5th ACM/IEEE international conference on mobile computing (MOBICOM)
7. Heinzelman WR, Chandrakasan A, Balakrishnan H (2000) Energy-efficient communication protocol for wireless microsensor networks. In: Proceedings of the 33rd Hawaii international conference on system sciences (HICSS)
8. Intanagonwiwat C, Govindan R, Estrin D (2000) Directed diffusion: a scalable and robust communication paradigm for sensor networks. In: Proceedings of the 6th ACM/IEEE international conference on mobile computing (MOBICOM)
9. Kahn JM, Katz RH, Pister KSJ (1999) Next century challenges: mobile networking for smart dust. In: Proceedings of the 5th ACM/IEEE international conference on mobile computing, pp 271–278
10. Leone P, Rolim J, Nikoletseas S (2005) An adaptive blind algorithm for energy balanced data propagation in wireless sensor networks. In: Proceedings of the IEEE international conference on distributed computing in sensor networks (DCOSS). Lecture notes in computer science (LNCS), vol 3267. Springer, pp 35–48
11. Luo J, Hubaux J-P (2005) Joint mobility and routing for lifetime elongation in wireless networks. In: Proceedings of the 24th INFOCOM
12. Nikoletseas S, Chatzigiannakis I, Antoniou A, Efthymiou C, Kinalis A, Mylonas G (2004) Energy efficient protocols for sensing multiple events in smart dust networks. In: Proceedings of the 37th annual ACM/IEEE simulation symposium (ANSS'04). IEEE Computer Society Press, pp 15–24
13. Triantafillou P, Ntarmos N, Nikoletseas S, Spirakis P (2003) NanoPeer networks and P2P worlds. In: Proceedings of the 3rd IEEE international conference on peer-to-peer computing (P2P 2003), pp 40–46

Probe Selection

Wen Xu[4], Weili Wu[1,2,4], and Ding-Zhu Du[3,4]
[1]College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi Province, China
[2]Department of Computer Science, California State University, Los Angeles, CA, USA
[3]Computer Science, University of Minnesota, Minneapolis, MN, USA
[4]Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Keywords

Approximation algorithm; Group testing; Probe selection; Virus detection

Years and Authors of Summarized Original Work

2006; Du, Hwang
2007; Wang, Du, Jia, Deng, Wu, MacCallum

Problem Definition

The virus identification is an important research topic in molecular biology. One method is using probes. A probe is a short oligonucleotide of size 8–25, which plays a role of ID when identify a virus in a biological sample through hybridization. If each probe hybridizes to a unique virus, then identification of virus is straightforward. However, unique probes are very hard to be obtained, especially for virus subtypes which are closely related. Therefore, how to identify virus with the minimum number of nonunique probes becomes an interesting problem.

Given a biological sample and a set of possibly nonunique probes, how to select a minimum subset of probes to identify viruses in the biological sample. This problem is called the *nonunique probe selection*.

Key Results

Suppose the biological sample contains only one virus. The problem is to determine what is this virus. To do so, it is sufficient to select probes satisfying the condition that different viruses hybridize different subsets of probes. This condition enables us to find the virus easily from a test outcome.

In general, suppose the biological sample contains at most d viruses. Then selected probes should satisfy the condition that different sets of at most d viruses should hybridize different subsets of selected probes. Schilep, Torney, and Rahman [9] first pointed out that this is actually a nongroup testing group testing problem [3].

Consider each virus as an item and each probe as a pool consisting of all viruses hybridized by the probe. A nonadaptive group testing with n items and t pools can be represented, and $t \times n$ binary matrix with rows labeled by pools and columns labeled by items and cell (i, j) contains 1-entry if and only if the ith pool contains item j. This binary matrix is called the incidence matrix of the nonadaptive group testing. In theory of nonadaptive group testing, the above condition means that the incidence matrix is $\bar{d}$-separable. Actually, a binary matrix is $\bar{d}$-separable if all Boolean sums of at most d columns are distinct. Here, by Boolean sum, we mean the following: If each column is seen as a set of rows corresponding to 1-entries in the column, then the Boolean sum can be seen as a union of columns. The Boolean sum is a classic statement in the study of group testing. With a $\bar{d}$-separable matrix, the test outcome can identify up to d viruses in biological sample.

In nonadaptive group testing, each test is on a pool. Thus, each probe can also be seen as a test. The test outcome is *positive* if the probe is hybridized by some virus in a biological sample and *negative* otherwise. Test outcomes for all selected probes can be written as a column vector which is exactly the union of columns corresponding viruses contained in the biological sample, where 1-entry denotes a positive outcome and 0-entry denotes a negative outcome. Therefore, the definition of $\bar{d}$-separable matrix means that different sets of at most d viruses receive different test-outcome t-dimensional vectors.

The nonunique probe selection problem can also be formulated as follows:

> MIN-$\bar{d}$-SS (Minimum $\bar{d}$-Separable Submatrix). Given a binary matrix $\boldsymbol{M}$, find the minimum of rows to form a $\bar{d}$-separable submatrix.

For any fixed d, MIN-$\bar{d}$-SS is NP-hard [3]. Moreover, from the test outcome obtained from $\bar{d}$-separable, it may take time $O(n^t)$ to find all existing viruses. This means that it is hard to decode the test outcome from a $\bar{d}$-separable matrix [3]. Therefore, Thai et al. [10] considered to use a d-disjunct matrix instead of $\bar{d}$-separable matrix. A binary matrix is *d-disjunct* if any union of d columns cannot contain the $(d + 1)$th column. Decoding test outcome from a d-disjunct matrix is very easy [3]. This introduces another minimization problem:

> MIN-d-DS (Minimum d-Disjunct Submatrix). Given a d-disjunct binary matrix $\boldsymbol{M}$, find a minimum subset of rows to form d-disjunct submatrix.

Theoretically, there is another similar problem as follows:

> MIN-d-SS (Minimum d-Separable Submatrix). Given a d-separable binary matrix $\boldsymbol{M}$, find a minimum subset of rows to form d-separable submatrix where a binary matrix is d-separable if all Boolean sums of exactly d columns are distinct.

For $d = 1$, MIN-d-SS is exactly the minimum test cover problem [5], also called the minimum test set problem [2] or the minimum test collection [6], which has a greedy approximation with performance $1 + 2 \ln n$ where n is the number of items [2]. This fact makes a suggestion that design greedy approximations for MIN-d-SS, MIN-$\bar{d}$-SS, and MIN-d-DS.

In fact, it is easy to construct greedy approximations with performance ratio $1 + 2d \ln n$ for MIN-d-SS, $1 + (d + 1) \ln n$ for MIN-d-DS, and $1 + 2d \ln(n + 1)$ for MIN-$\bar{d}$-SS. For example, let us study MIN-d-DS. Consider the collection $\mathcal{S}$ of all possible pairs (C, D) of one column

C and a submatrix D with d columns. Clearly $|\mathcal{S}| < n^{d+1}$. A row is said to *cover* such a pair (C, D) if and only if at this row, the entry of column C is 1 and all entries of columns in D are 0. Now, MIN-d-DS is equivalent to the problem of finding the minimum number of rows covering all such pairs. This is a special case of the set cover problem. It is well known that there is a greedy algorithm for the set cover problem with performance ratio $1 + \ln|\mathcal{S}| < 1 + (d+1)\ln n$.

This greedy algorithm works well only for small d because its running time is $O(n^{d+1})$. When d is large, it runs too slow. Therefore, we must look for other smart ways. Schilep, Torney, and Rahman [9] proposed an algorithm which adds probe one by one until the incidence matrix with considered viruses forms a $\bar{d}$-separable matrix. This does not work for large d, neither. In fact, if d is not bounded, then testing whether a binary matrix is d-separable, or $\bar{d}$-separable, or d-disjunct is co-NP-complete [3]. There exist more methods [8] in the literature, which work well for small d. However, no efficient method has been found to produce good solutions for larger d.

<?pag ?>In some applications, the pool size cannot be too big due to the sensitivity of tests. For example, UNH suggested in ADS testing, each pool should not contain more than five blood samples. When the pool size is bounded, the problem becomes easier. For instance, let us consider the case that every pool has size at most 2 so that all pools of size 2 together with items form a graph G where pools are edges and item are vertices. Halldórsson et al. [6] and De Bontridder et al. [2] proved that in this case, MIN-1-SS is still APX-hard, which means that there is no polynomial-time approximation scheme for it unless NP=P. They also showed that MIN-1-SS in this case has a polynomial-time approximation with performance ratio $7/6 + \varepsilon$ for any fixed $\varepsilon > 0$.

A surprising result was showed by Wang et al. [11] that a subgraph H of G represents a d-disjunct matrix if and only if every vertex in H has degree at least $d + 1$, and hence, finding such an H with minimum number of edges is polynomial-time solvable. What about the case that all pools have size 3 Wang et al. proved that in this case MIN-d-DS is still NP-hard. However, there exist polynomial-time approximations with better performance.

Applications

In practice, we may select nonunique probes in the following steps [9]:

Step 1. Estimate an upper bound d for the number of viruses existing in a given biological sample. Collect a large set of nonunique probes to form a $\bar{d}$-separable matrix.

Step 2. From this large set of probes, find a subset of probes to identify up to d viruses by computing an approximation solution for MIN-d-DS or MIN-$\bar{d}$-SS.

Step 3. Decode the presence or absence of viruses in the given biological sample from test outcome.

Open Problems

When d is not fixed, MIN-d-DS belongs to Σ_2^p and is conjectured to be Σ_2^p-complete [3].

Recommended Reading

1. Berman P, Dasgupta B, Kao M-Y (2005) Tight approximability results for test set problems in bioinformatics. J Comput Syst Sci 71:145–162
2. De Bontridder KMJ, Halldórsson BV, Halldórsson MM, Hurkens CAJ, Lenstra JK, Ravi R, Stougie L (2003) Approximation algorithms for the test cover problem. Math Program 98:477–491
3. Du D-Z, Hwang FK (2006) Pooling designs and nonadaptive group testing. World Scientific, New Jersey
4. Du D-Z, Ko K-I (2000) Theory of computational complexity. Wiley, New York
5. Garey MR, Johnson DS (1979) Computers and intractability. W.H. Freeman, San Francisco
6. Halldórsson BV, Halldórsson MM, Ravi R (2001) On the approximability of the minimum test collection problem. Lect Notes Comput Sci 2161:158–169
7. Karp RM, Stougton R, Yeung KY (1999) Algorithms for choosing differential gene expression experiments. In: Proceedings of the third annual international conference on computational molecular biology, Lyon, pp 208–217

8. Klau G, Rahmann S, Schliep A, Vingron M, Reinert K (2004) Optimal robust non-unique probe selection using integer linear programming. Bioinformatics 20:I186–I193
9. Schliep A, Torney DC, Rahmann S (2003) Group testing with DNA chips: generating designs and decoding experiments. In: Proceedings of the 2nd IEEE computer society bioinformatics conference, Stanford
10. Thai M, Deng P, Wu W, Znati T (2006) Efficient algorithms for non-unique probes selection using d-disjunct matrix. Muscript
11. Wang F, Du H, Jia X, Deng P, Wu W, MacCallum D (2007) Non-unique probe selection and group testing. Theor Comput Sci 381:29–32<?pag ?>

Prophet Inequality and Online Auctions

Mohammad Taghi Hajiaghayi and Vahid Liaghat
Department of Computer Science, University of Maryland, College Park, MD, USA

Keywords

Auction design; Online algorithms; Online auctions; Prophet inequality

Years and Authors of Summarized Original Work

2007; Hajiaghayi, Kleinberg, Sandholm
2012; Kleinberg, Weinberg
2012; Alaei, Hajiaghayi, Liaghat
2015; Esfandiari, Hajiaghayi, Liaghat, Monemizadeh

Problem Definition

The topic of prophet inequality has been studied in optimal stopping theory since the 1970s [7, 9, 10] and more recently in computer science [1, 3, 6, 8]. In the prophet inequality setting, given (not necessary identical) independent distributions $D_1, \ldots, D_n$, a sequence of random variables $x_1, \ldots, x_n$ where x_i is drawn from D_i, a collection M of *feasible* subsets of $\{1, \ldots, n\}$, an onlooker has to choose from the succession of these values, where x_i is revealed to us at time step i. The onlooker starts with an empty set $S = \phi$. Upon the arrival of a value x_i, the onlooker can choose to either add x_i to the set S or discard it permanently. After the arrival of all values, the indices of values in S should form a feasible set in M. The revenue of the onlooker is the total value of variables in S. The onlooker's goal is to maximize his/her (expected) revenue compared to the hindsight expected revenue of a prophet who knows the drawn values in advance. The optimal offline solution (the prophet's revenue) is defined as $OPT = \mathrm{E}\left[\max_{I \in M} \sum_{i \in I} x_i\right]$. The *competitive ratio* of an algorithm for the onlooker is defined as the worst-case ratio of the expected revenue of the onlooker over OPT. This inequality ratio has been interpreted as meaning that a prophet with complete foresight has only a bounded advantage over an onlooker who observes the variables one by one, and this explains the name prophet inequality.

Different Variants

The *basic prophet inequality* discovered by Krengel, Sucheston, and Garling in the 1970s concerns the case in which the onlooker can only choose one value [9], i.e., M is the set of singletons. Decades later in 2007, Hajiaghayi, Kleinberg, and Sandholm [6] considered the *k-choice prophet secretary* variant in which sets with at most k elements are feasible. Later in 2012, Kleinberg and Weinberg [8] considered the more general *matroid prophet inequality*. In this variant the collection M contains the independent sets of a matroid.

Other prophet inequality settings (dependent D_i's, restricted prophets, etc.) have been considered in the literature as well. For an overview of these models, we refer the reader to [6, 8] and references therein.

Key Results

Krengel, Sucheston, and Garling [9] were first to consider basic prophet inequality. Using a very simple example, they showed no online algorithm can have a competitive ratio better than $\frac{1}{2}$: let

$q = \frac{1}{\epsilon}$. The first value, i.e., x_1 is always 1. The second value is either q with probability ϵ or 0 with probability $1 - \epsilon$. Observe that the expected revenue of any (randomized) online algorithm is at most $\max\left\{1, \epsilon\left(\frac{1}{\epsilon}\right)\right\} = 1$. However the prophet, i.e., the optimum offline solution, would choose x_2 if $x_2 = q$; otherwise he would choose the first value. Thus the optimum offline revenue is $(1-\epsilon) \times 1 + \epsilon\left(\frac{1}{\epsilon}\right) \approx 2$. We note that without considering stochastic assumptions, we cannot hope to get any constant competitive ratio.

An algorithm for the basic prophet inequality problem can be described by setting a threshold for every step: we stop at the first step that the arriving value is higher than the threshold of that step. The classical prophet inequality result [9] states that by choosing the same threshold $OPT/2$ for every step, one achieves the tight competitive ratio of $1/2$.

For the k-choice variant, Hajiaghayi et al. [6] show an algorithm with the competitive ratio $1 - O\left(\frac{\sqrt{\ln k}}{\sqrt{k}}\right)$. Later Alaei [1] improved this bound to $1 - \frac{1}{\sqrt{k+3}}$ using an involved randomized approach (gamma-conservative magician). Alaei, Hajiaghayi, and Liaghat simplified and generalized these results to the matching prophet inequality [2, 3]. Later they generalized their result to the *online stochastic generalized assignment problem* [4] (GAP) with slightly worse competitive ratio of $1 - \frac{1}{\sqrt{k}}$. In GAP, we have a set of items to be placed in a set of bins. The bins are known in advance, but the sequence of items arrives online; each item has a value and a size; upon arrival, an item can be placed in one of the bins or can be discarded permanently; the objective is to maximize the total value of the placement. Both value and size of an item may depend on the bin in which the item is placed; the size of an item is revealed only after it has been placed in a bin; distribution information is available about the value and size of each item in advance (not necessarily i.i.d.); however, items arrive in adversarial order (nonadaptive adversary). Alaei et al. [4] show an algorithm with the competitive ratio of $1 - \frac{1}{\sqrt{k}}$ where in this setting k is interpreted as the minimum number of items that can fill up the capacity of a bin.

Kleinberg and Weinberg [8] considered the matroid prophet inequality. They show an elegant algorithm that still achieves the competitive ratio of $1/2$. Generalizing their result still further, they show that under an intersection of p matroid constraints, the prophet's revenue exceeds the onlooker's by a factor of at most $O(p)$, and this factor is also tight. Kleinberg and Weinberg design the following algorithm for the matroid prophet inequality. The algorithm pretends that the online selection process is Phase 1 of a two-phase game; after each x_i has been revealed in Phase 1 and the algorithm has accepted some set A_1, Phase 2 begins. In Phase 2, a new weight will be sampled for every matroid element, independently of the Phase 1 weights, and the algorithm will play the role of the prophet on the Phase 2 weights, choosing the max-weight subset A_2 such that $A_1 \cup A_2$ is independent. However, the revenue for choosing an element in Phase 2 is only half of its value. When observing element i and deciding whether to select it, our algorithm can be interpreted as making the choice that would maximize its expected revenue if Phase 1 were to end immediately after making this decision and Phase 2 were to begin. Of course, Phase 2 is purely fictional: it never actually takes place, but it plays a key role in both the design and the analysis of the algorithm. The analysis of the algorithm is involved and relies on a careful analysis of the expected revenue at each step. For further intuitions about the analysis, we refer the reader to [8].

Applications

Beyond their interest as theorems about pure online algorithms or optimal stopping rules, prophet inequalities also have applications to mechanism design. Mechanism design has traditionally focused on the offline setting where all agents are present up front. However, many electronic commerce applications do not fit that model because the agents can arrive and depart dynamically. This is characteristic, for example, of online ticket auctions, search keyword auctions, Internet auctions, and scheduling computing jobs on a

cloud. The online aspect is characteristic of some important traditional applications as well, such as the sale of a house, where the buyers arrive and depart dynamically.

The pioneer work of Hajiaghayi, Kleinberg, and Sandholm [6] initiated the research on the relationship between algorithmic mechanism design and prophet inequalities. They observed that algorithms used in the derivation of prophet inequalities, owing to their monotonicity properties, could be interpreted as (temporarily) truthful online auction mechanisms and that the prophet inequality in turn could be interpreted as the mechanism's approximation guarantee. Indeed, Bayesian optimal mechanism design problems provide a compelling application of prophet inequalities in economics. In such a Bayesian market, we have a set of n agents with private types sampled from (not necessary identical) known distributions. Upon receiving the reported types, a seller has to allocate resources and charge prices to the agents. The goal is to maximize the seller's revenue in equilibrium. Chawla et al. [5] pioneered the study of the approximability of a special class of such mechanisms, *sequential posted pricing* (SPM): the seller makes a sequence of take-it-or-leave-it offers to agents, offering an item for a specific price. They show although simple, SPMs approximate the optimal revenue in many different settings. Therefore prophet inequalities directly translate to approximation factors for the seller's revenue in these settings through standard machineries. Indeed one can analyze the so-called *virtual values* of winning bids introduced by Roger Myerson [11], to prove via prophet inequalities that the expected virtual value obtained by the SPM mechanism approximates an offline optimum that is with respect to the exact types. Chawla et al. [5] provide a type of prophet inequality in which one can choose the ordering of agents. As mentioned before, Kleinberg and Weinberg [8] later improved their result by giving an algorithm with the tight competitive ratio of 0.5 for an adversarial ordering.

Cross-Reference

▸ Algorithmic Mechanism Design

Recommended Reading

1. Alaei S (2011) Bayesian combinatorial auctions: expanding single buyer mechanisms to many buyers. In: FOCS, Palm Springs
2. Alaei S, Hajiaghayi MT, Liaghat V, Pei D, Saha B (2011) Adcell: ad allocation in cellular networks. In: ESA, Saarbrücken
3. Alaei S, Hajiaghayi M, Liaghat V (2012) Online prophet-inequality matching with applications to ad allocation. In: EC, Valencia, pp 18–35
4. Alaei S, Hajiaghayi M, Liaghat V (2013) The online stochastic generalized assignment problem. In: APPROX, Berkeley
5. Chawla S, Hartline JD, Malec DL, Sivan B (2010) Multi-parameter mechanism design and sequential posted pricing. In: STOC, Cambridge
6. Hajiaghayi MT, Kleinberg RD, Sandholm T (2007) Automated online mechanism design and prophet inequalities. In: AAAI, Vancouver
7. Kennedy DP (1987) Prophet-type inequalities for multi-choice optimal stopping. Stoch Proc Appl 24:77–88
8. Kleinberg R, Weinberg SM (2012) Matroid prophet inequalities. In: STOC, New York, pp 123–136
9. Krengel U, Sucheston L (1977) Semiamarts and finite values. Bull Am Math Soc 83:745–747
10. Krengel U, Sucheston L (1978) On semiamarts, amarts, and processes with finite value. In Kuelbs J (ed) Probability on banach spaces. M.L. Dekker, New York
11. Myerson RB (1981) Optimal auction design. Math Oper Res 6:58–73

Q

Quadtrees and Morton Indexing

Herman Haverkort[1] and Laura Toma[2]
[1]Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
[2]Department of Computer Science, Bowdoin College, Brunswick, ME, USA

Keywords

IO-efficient algorithms; Segment intersection; Space decomposition; Space-filling curve

Years and Authors of Summarized Original Work

2002; Hjaltason, Samet
2006; Agarwal, Arge, Danner
2010; de Berg, Haverkort, Thite, Toma
2013; McGranaghan, Haverkort, Toma

Problem Definition

The quadtree describes a class of data structures for geometric objects. A quadtree partitions space hierarchically using a stopping rule that decides when a region is small enough so that it does not need to be subdivided further. If the space is d dimensional, a quadtree recursively divides a d-dimensional hypercube containing the input data into 2^d hypercubes until each region satisfies the given stopping rule. In 2D, the hypercubes are squares. Three-dimensional quadtrees are also known as *octrees*. Quadtrees have been used for many types of data, such as points, line segments, polygons, rectangles, curves, and images, and for many types of applications. For a detailed presentation, we refer to the book by Samet [10]. While their worst-case behavior is good only in some simple cases, quadtrees perform well empirically in many applications.

A quadtree can be stored as a tree that corresponds to the hierarchical subdivision of the input region. A region that is subdivided further is then represented by a node with four children, one for each quadrant; the cells that are not subdivided further constitute the leaves of the tree and represent a subdivision of the input region. A quadtree with m leaves has exactly $(m-1)/3$ internal nodes and $4m/3 - 1/3$ nodes in total. Hence it can be described by a sequence of $4m/3 - 1/3$ bits representing the nodes of the tree in preorder, where each internal node is represented by a 1 (meaning that the next bit encodes its first child) and each leaf is represented by a 0. However, for efficient navigation one would typically use a pointer-based data structure. Alternatively, one may store only the leaves of the tree, ordered along a space-filling curve. This variant of the quadtree is called the *linear quadtree* and was introduced by Gargantini [3]. The linear quadtree has smaller memory requirements as it does not store the tree structure but only the data in the

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

leaves. This makes it particularly useful when dealing with large data.

In this entry we focus on quadtree construction algorithms that are efficient on very large data. To analyze these algorithms, we use the ▸ I/O-Model and the ▸ Cache-Oblivious Model. We'll use the terms linear quadtree and quadtree subdivision interchangeably. We define the size of a subdivision as the number of cells it contains and the size of a cell is the size of the data (points and edges) it contains/intersects.

The Complexity of Quadtrees for Points in the Plane

Let P be a set of n points in the plane and assume, for simplicity, that the points lie in the unit square. A quadtree for P corresponds to a recursive subdivision of the unit square into four equal regions, called canonical squares, quadrants, or cells, until each square contains at most one point. Following customary terminology in the computational geometry literature (and in deviation from Samet [10]), we refer to this generically as a *point quadtree*.

In the worst case, the size of a quadtree subdivision on P cannot be bounded by a function of n. If ϵ is the distance between the two closest points in P, the worst-case complexity is $\Theta(n \lg \frac{1}{\epsilon})$, and the corresponding tree may have a large number of empty nodes. A *compressed* quadtree is a quadtree where paths of nodes that each have three empty children are merged into a single node along with their empty children; the region corresponding to the merged node is called a *donut* and represents the difference between two canonical squares. A compressed quadtree for a set of n points in the plane such that each cell contains at most one point has size $\Theta(n)$ and height $\Theta(n)$ in the worst case.

The Complexity of Quadtrees for Line Segments in the Plane

Let $\mathcal{E}$ be a set of n non-intersecting line segments in the plane – for example, the edges of a planar subdivision – and assume, as above, that the edges lie in the unit square. We refer to a quadtree for $\mathcal{E}$ generically as an *edge quadtree* and assume that each edge is stored in all the cells that it intersects. The simplest way to define an edge quadtree may be to take a point quadtree on the endpoints of the edges and then store each edge with the leaves that correspond to the quadtree cells intersected by the edge. We denote by l the number of intersections between $\mathcal{E}$ and the cells in the subdivision. Even if we use a compressed quadtree, in the worst case, there can be $\Theta(n)$ cells that each intersects $\Theta(n)$ edges, so $l = \Theta(n^2)$, and the quadtree will have size $\Theta(n + l) = \Theta(n^2)$. Other edge quadtrees can be defined by formulating stopping criteria that allow subdividing cells further in order to limit the number of edges that intersect each cell; this will result in a subdivision with more cells but smaller number of edges per cell. However, obtaining a good trade-off between the size of a cell (number of points and edges inside or intersecting it) and the number of cells in the subdivision is not possible in the worst case. Note that an edge quadtree that splits a region until it intersects a single edge will result in a subdivision of unbounded size since the distance between two edges can be arbitrarily small.

Quadtrees and Morton Indexing

Quadtrees are often used in conjunction with a z-order space-filling curve. A z-order, or Morton order, can be understood as a mapping from two-dimensional (in general multidimensional) data to one dimension. We use a z-order curve that visits the four quadrants of the initial square, recursively, in the order top left, top right, bottom left, and bottom right. This order gives a well-defined ordering between any two canonical squares in the subdivision. If we define canonical squares to be closed on the top and left side and open on the bottom and right side, the z-order also gives a well-defined ordering between any two points in the input region. Let $p = (p_x, p_y)$ be a point in the unit square $[0, 1)^2$, with the x-axis oriented from left to right and the y-axis oriented from top to bottom. We define the z-index $Z(p)$ of p to be the value in the range $[0, 1)$ obtained by interleaving the bits in the fractional parts of p_x and p_y, starting with a bit from p_y. The value $Z(p)$ is sometimes called the *Morton block index* of p. The z-order of two points in the unit square

is the order of their z-indices. A crucial property is that the z-indices of all points in a canonical square σ form an interval $[z_1, z_2)$ of $[0, 1)$, where z_1 is the z-index of the top left corner of σ. A donut cell is the difference between two canonical squares $[z_1, z_2)$ and $[z_3, z_4)$, and thus, it is the union of two intervals $[z_1, z_3)$ and $[z_4, z_2)$.

With this notation, a (compressed) quadtree subdivision corresponds to a subdivision $\mathcal{Q}$ of the z-order curve and can be viewed as a set of consecutive, adjacent, nonoverlapping intervals, covering $[0, 1)$, in z-order: $\mathcal{Q} = \{[z_1 = 0, z_2), [z_2, z_3), \ldots\}$. Each interval $[z_i, z_{i+1})$ corresponds to a cell σ_i, which is either a canonical square or a part of a donut. We note that this representation does not make any assumptions on the stopping criterion used to generate the quadtree subdivision and thus works on any quadtree subdivision, no matter how many points are in a region and whether it is compressed or not. A linear edge quadtree can therefore be represented as a set of key-edge pairs, where each intersection of an edge e with a quadtree cell σ corresponding to an interval $[z_1, z_2)$ is represented by storing edge e with key z_1; thus each cell stores all edges that intersect it [2, 4].

Key Results

Point Quadtrees

Agarwal et al. [1] described an algorithm for constructing a quadtree on a set of n points in the plane such that each cell contains $O(k)$ points; the algorithm runs in $O(\frac{n}{B}\frac{h}{\log M/B})$ I/O's, where h is the height of the quadtree. Effectively, this is $O(\text{sort}(n))$ I/O's only when $h = O(\log n)$, which is true when the points are nicely distributed. A bound on the size of the quadtree is not given, and the quadtree is not compressed, which means the quadtree size can be unbounded in the worst case. The algorithms were implemented and tested as part of an application to interpolate LIDAR datasets, which are nicely distributed and unlikely to cause worst-case behavior.

De Berg et al. [2] described an algorithm to construct a compressed quadtree subdivision with at most one point per cell in $O(\text{sort}(n))$ I/O's, as a step in the construction of their Guard-quadtree for edges which is discussed below. Haverkort et al. [4] describe a simple generalization of this algorithm which constructs a compressed quadtree subdivision of $O(n/k)$ cells with at most k points per cell in the same I/O-bound. Thus, compared to the algorithm by Agarwal et al., a stronger bound on the I/O-complexity is obtained, along with an upper bound on the number of cells in the subdivision.

PM Quadtrees

A variety of edge quadtrees were described by Samet and various co-authors [5–7, 9, 11, 12]. All of these solutions are aimed at subdividing the cells that intersect too many edges, while also limiting the total size of the quadtree and being able to construct it I/O-efficiently.

The *PM quadtree* [11] allows a region to contain more than one edge if the edges meet at a vertex inside the region; otherwise it keeps subdividing it. Variants of PM quadtrees differ in how to handle regions that contain no vertices (only edges). The *segment quadtree* [12] is a linear quadtree in which a leaf cell is either empty, contains one edge and no vertices, or contains precisely one vertex and its incident edges. The most versatile structure within the PM family is the *PMR quadtree* [9], a linear quadtree where each region may have a variable number of segments and regions are split if they contain more than a predetermined threshold of edges. The tree is built incrementally, by inserting each segment into all the regions that it intersects. When a region contains more segments than a predetermined splitting threshold, the region is split, once, into four quadrants. Improved algorithms for the construction (or *bulk loading*) of the PMR quadtree were described in [5–7]. These algorithms are developed and optimized with massive data in mind and use I/O-efficient sorting as one of the steps. It is reported that in many cases (although not in the worst case), the I/O-cost of the bulk-loading algorithm is the same as that of external sorting [5]. The algorithms are reported to perform well in practice, but there are several disadvantages: the the resulting quadtree depends on the insertion order; complexity is analyzed

in terms of various parameters that depend on the data; and the performance is not worst-case optimal. On the plus side, the algorithms can handle insertions and work in situations where the data is dynamic.

Star-Quadtrees

The Star-quadtree by De Berg et al. [2] is designed for fat triangulations (a triangulation is fat if every angle of every triangle is larger than some fixed positive constant δ). A Star-quadtree is a linear, uncompressed edge quadtree that splits a region until all edges intersecting a region are incident on one common endpoint (similar to the PM quadtree by Samet and Webber [11]). The Star-quadtree can be built on any set of edges in the plane, but, when the input is a fat triangulation, it can be shown that this stopping rule creates (1) a quadtree of $\Theta(n)$ size and (2) each leaf cell in the quadtree (each cell in the subdivision) intersecting $\Theta(1)$ edges. The height of the quadtree can still be $\Theta(n)$, which makes a top-down construction, such as that used by Agarwal et al. [1], height dependent and not optimal. The authors of the Star-quadtree describe a completely different algorithm for its construction that crucially exploits the stopping criterion and runs in $O(\text{sort}(n))$ I/O's if the input is a fat triangulation.

Guard-Quadtrees

The Guard-quadtree by De Berg et al. [2] is designed for sets of non-intersecting edges of low density – a set of edges has density λ if any disk D is intersected by at most λ edges whose length is at least the diameter of D. For a given set of n edges, the authors define a set of at most $4n$ *guards*, namely, the vertices of the minimum axis-parallel bounding rectangles of the individual edges. The Guard-quadtree is a linear, compressed edge quadtree that splits a region until it contains at most one *guard*. As the set of guards is a superset of the endpoints of the edges, this leads to a subdivision that is more refined than a quadtree built only on the endpoints of edges. The stopping rule, together with compression, leads to a quadtree subdivision that has $O(n)$ cells and each cell intersects $O(1)$ edges, provided the set of edges to be stored has low density. Furthermore, the quadtree can be constructed in $O(\text{sort}(n))$ I/O's in this case.

K-Quadtrees

Combining ideas from De Berg et al. [2] with packing more vertices in a region, Haverkort et al. [4] described an I/O-efficient edge quadtree referred to as a K-quadtree. For any $k \geq 1$, the K-quadtree is a compressed, linear quadtree built on the endpoints of the edges, with $O(n/k)$ cells in total and such that each cell contains $O(k)$ vertices (and such that each edge is stored in all cells that they intersect). Each cell in the subdivision can intersect $O(n)$ edges in the worst case. For $k = 1$, a K-quadtree is a linear, compressed edge quadtree with $O(n)$ cells and at most one vertex per cell. Larger values of k can be chosen to trade off between the number of cells $O(n/k)$ and the number of vertices in a cell $O(k)$.

The algorithm for building a K-quadtree has two steps: First it builds, in $O(\text{sort}(n))$ I/O's, a linear, compressed quadtree subdivision on the endpoints of $\mathcal{E}$ with $O(n/k)$ cells in total and such that each cell contains $O(k)$ vertices. This step is a simple generalization of the algorithm for building Guard-quadtrees from [2]. In the second step, the K-quadtree construction algorithm computes the intersections between the edges and the subdivision in $O(\text{sort}(n + l))$ I/O's, where $l = O(n^2/k)$ is the total number of intersections.

The main idea of the second step of the algorithm is to split the set of edges into edges of positive slope $\mathcal{E}_+$ and edges of negative slope $\mathcal{E}_-$ and compute the intersections of each set separately. The intersections of $\mathcal{E}_+$ with the subdivision are computed by *time-forward processing*, as follows. The cells of the quadtree subdivision are scanned in z-order. At any point during this scan, there is a *frontier*: an xy-monotone curve that constitutes the boundary between the cells that have already been scanned and the cells that are still to be scanned. The algorithm relies on the property that an edge of positive slope intersects the cells in the subdivision in z-order (a similar property holds for the edges of negative slope and a reflected version of the z-order). During the scan, each edge of $\mathcal{E}_+$ is passed on, from

each intersected quadtree cell to the next, through a supporting data structure that stores the edges intersecting the frontier.

Unlike standard instantiations of time-forward processing, the supporting data structure is not a priority queue, but it is a list, implemented as two stacks, containing the edges that intersect the frontier, in order along the frontier. At each point in time, the list starts at the bottom of one stack and goes up to the top and then down the other stack. The cutting point between the two stacks corresponds to the current scanning position in the list; scanning backward or forward in the list for lookups and updates is implemented by moving elements from one stack to another. The key to I/O-efficiency is that the total amount of scanning that is needed to maintain the supporting data structure is linear in the output size, incurring only $O(\text{scan}(l))$ I/Os.

As the algorithm relies only on the basic building blocks of I/O-efficient sorting, scanning, and stacks, it is also easy to implement cache obliviously.

Compared to a quadtree that employs a stopping criterion that aims to bound the number of edges intersecting a cell (like PMR, Star- and Guard-quadtrees), the simpler K-quadtree has a couple of advantages: (1) the resulting subdivision size is smaller; (2) the total size of the quadtree (the number of intersections between edges and the subdivisions) is also smaller since the size of the subdivision is smaller; and (3) the quadtree can be built in $O(\text{sort}(n + l))$ I/O's, without making any assumptions about the input.

Datasets

Common test datasets for 2D quadtrees are triangulated terrains and USA TIGER data. They represent relatively simple classes of inputs; however they arise frequently in practice and have been used extensively as test beds for spatial index structures. The TIGER dataset consists of 50 datasets, one for each state, containing roads, railways, boundaries, and hydrography in the state. The size of a dataset ranges from 115,626 edges (Delaware) to 40.4 million edges (Texas). The TIGER datasets can be downloaded from http://www.census.gov/cgi-bin/geo/shapefiles2013/main.

Experimental Results

Since many of the quadtree algorithms perform much better in practice than their theoretical worst-case bounds, experimental analysis is an important way to assess their merits. Some of the early experimental analysis of quadtrees performance on massive data was by Hjaltason and Samet [6]. They describe ample results concerning practical performance of PMR quadtrees in terms of construction time, insertions and bulk insertions, comparison with R-tree bulk-loading, and, as an application, performance of spatial join using quadtrees to store the datasets. Their test data consists of TIGER datasets ranging from 40K lines to approximately 260K edges on a machine with 64MB RAM.

Agarwal et al. [1] implemented and tested their I/O-efficient point quadtree part of an application to interpolate LIDAR datasets, where it was used specifically for batched neighbor finding (finding the points in all neighbor leaves for each quadtree leaf). The algorithms are scalable up to at least 500 million points (20GB raw data) (their platform was an Intel 3.4GHz machine with 1GB RAM running Linux).

Haverkort et al. [4] described an experimental analysis of K-quadtrees reporting on the construction time and size of the quadtree (number of cells and number of edge-cell intersections) for various values of k, as well as computing a spatial join using K-quadtrees. The K-quadtree construction algorithm is efficient and scalable, with the running time getting faster as more points are packed into a leaf. Even though the number of edges intersecting a cell may be large, the average size of a cell stays low and the total size of the quadtree is linear. Their tests use TIGER data with the largest bundle, corresponding to the entire USA, having approximately 427 million edges, on a machine with 512MB RAM. A comparison with the PMR quadtree results of

Hjaltason and Samet [6] is difficult because of the difference in platforms.

Extensions

A series of recent results have shown that compressed quadtrees and Delaunay triangulations are equivalent structures, in the sense that a compressed quadtree of a set of points P in the plane can be computed in linear time given the Delaunay triangulation $DT(P)$; and the other way around, the Delaunay triangulation $DT(P)$ can be computed in linear time given a compressed quadtree of P; see, for example, Löffler and Mulzer [8]. In the I/O-model, both problems can be solved in $O(\text{sort}(n))$ I/O's. This naturally brings the question of whether one can be computed from the other in $O(\text{scan}(n))$ I/O's.

Recommended Reading

1. Agarwal PK, Arge L, Danner A (2006) From point cloud to grid DEM: a scalable approach. In: Proceedings of the 12th symposium on spatial data handling, Vienna, pp 771–788
2. de Berg M, Haverkort H, Thite S, Toma L (2010) Star-quadtrees and guard-quadtrees: I/O-efficient indexes for fat triangulations and low-density planar subdivisions. Comput Geom 43(5):493–513
3. Gargantini I (1982) An effective way to represent quadtrees. Commun ACM 25(12):905–910
4. Haverkort H, Toma L, Wei BP (2013) An edge quadtree for external memory. In: Proceedings of the 12th international symposium on experimental algorithms, Rome, pp 115–126
5. Hjaltason G, Samet H (1999) Improved bulk-loading algorithms for quadtrees. In: Proceedings of the ACM international symposium on advances in GIS, Kansas City, pp 110–115
6. Hjaltason GR, Samet H (2002) Speeding up construction of PMR quadtree-based spatial indexes. VLDB J 11:190–137
7. Hjaltason G, Samet H, Sussmann Y (1997) Speeding up bulk-loading of quadtrees. In: Proceedings of the ACM international symposium on advances in GIS, Las Vegas
8. Löffler M, Mulzer W (2011) Triangulating the square and squaring the triangle: quadtrees and delaunay triangulations are equivalent. In: Proceedings of the 22nd ACM-SIAM symposium on discrete algorithms (SODA), San Francisco, pp 1759–1777
9. Nelson R, Samet H (1987) A population analysis for hierarchical data structures. In: Proceeding of the SIGMOD, San Francisco, pp 270–277
10. Samet H (2006) Foundations of multidimensional and metric data structures. Morgan-Kaufmann, San Francisco
11. Samet H, Webber R (1985) Storing a collection of polygons using quadtrees. ACM Trans Graph 4(3):182–222
12. Samet H, Shaffer C, Webber R (1986) The segment quadtree: a linear quadtree-based representation for linear features. In: Data structures for raster graphics pp 91–123

Quantification of Regulation in Networks with Positive and Negative Interaction Weights

Colin Campbell[1] and Réka Albert[2]
[1]Department of Physics, Pennsylvania State University, University Park, PA, USA
[2]Department of Biology and Department of Physics, Pennsylvania State University, University Park, PA, USA

Keywords

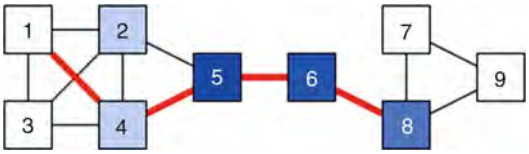
Complex systems; Edge weights; Graph theory; Interaction strength; Network communicability; Regulation; Signal transduction; Systems biology

Years and Authors of Summarized Original Work

2011; Campbell, Thakar, Albert

Problem Definition

A network representation of a complex system comprises nodes, which represent system elements, and edges, which represent interactions between the elements. Networks may be described in terms of their topology; for instance, some nodes may be connected to an atypically large number of other nodes, and some may act as bridge nodes that participate in paths between



Quantification of Regulation in Networks with Positive and Negative Interaction Weights, Fig. 1 Common network measures applied to a sample 9-node network with symmetric interactions. Darker nodes have higher betweenness centrality (i.e., they tend to act as a bridge between other pairs of nodes); note that even nodes with low degree (i.e., few connections) may have high betweenness centrality. Highlighted edges show a shortest path (length 4) between nodes 1 and 8

many other pairs of nodes (Fig. 1). For a review of topological network measures, see [1–3].

In some contexts, this topological structure serves as a basis for a dynamical description, where nodes are characterized by a dynamic variable that is regulated by the node's interactions. For instance, in the Boolean framework, nodes are either ON or OFF (1 or 0, respectively) [4]. In biological regulatory networks, where interactions between system elements can represent both upregulation and downregulation, one common dynamic scheme is summative [5]:

$$x_i\,(t+\tau) = \mathrm{sgn}\left(\sum_j E_{j,i}\,x_j\,(t)\right),$$

where $E_{j,i}$ is the weight of the interaction from node j to node i and absent interactions have a weight of 0 by definition. In such a framework, the state change of a node can propagate to the node(s) it directly regulates, then to the node(s) they regulate, and so on. This information flow across a network is sometimes referred to as network communicability [6]. A topological analysis of the network should, in principle, give insight into its dynamical structure and address questions such as, "Which nodes yield a strong influence on many other nodes?," "Which nodes are regulated in a complex way by many other nodes?," and "Which nodes seem to have a peripheral impact on the dynamics of the network?"

However, while networks have been used to explicate the structure and function of a large and diverse array of complex systems, most network measures consider the most general properties of networks and are therefore ill-suited for application to specialized networks. The positive and negative edge weights typically used in biological regulatory networks are one such specialization: standard network measures do not consider edge weights of opposite sign and are therefore ill-equipped to fully capture the dynamical implications of their topology.

Here, we address this shortcoming by developing a suite of topological measures that address the regulatory relationship between nodes that are connected by edges with both positive and negative edge weights. We first consider node-node interactions and then summarize those measures to quantify both the regulatory impact of a node on the entire network and of the entire network on a node.

Key Results

To first consider node-node relationships, we introduce two complementary measures. The **weighted node-node path count** from node i to node j considers both the number of paths from node i to node j and their length:

$$\omega_{ij} \equiv \sum_{l=1}^{l_{\max}} \frac{p_{lij}^{+} + p_{lij}^{-}}{l}$$

Here, p_{lij}^{+} and p_{lij}^{-} respectively indicate the number of positive and negative paths from node i to node j of length l. While we here consider a path to be positive if it contains 0 or an even number of negative edges, note that this measure effectively ignores the sign of the paths. To take this into consideration, we introduce the **node-node path influence**:

$$\pi_{ij} \equiv \sum_{l=1}^{l_{\max}} \frac{p_{lij}^{+} - p_{lij}^{-}}{l}$$

π_{ij} is therefore bounded by the range $[-\omega_{ij}, \omega_{ij}]$. ω_{ij} indicates the regulatory strength insofar as it is large when there are many short paths between the nodes and decreases when the paths are few and/or long; π_{ij} indicates the overall regulatory nature of those interactions. Values close to

0 relative to $|\omega_{ij}|$ indicate mixed (complex) regulation, while values close to $|\omega_{ij}|$ indicate overall positive or negative regulation.

Node-network relationships may be assessed by cumulating these measures with a fixed source or target node. The **node path influence**, ι_i, and **node path susceptibility**, σ_j, take this into account for a fixed source and target node, respectively:

$$\iota_i \equiv \sum_j \pi_{ij}\omega_{ij}$$

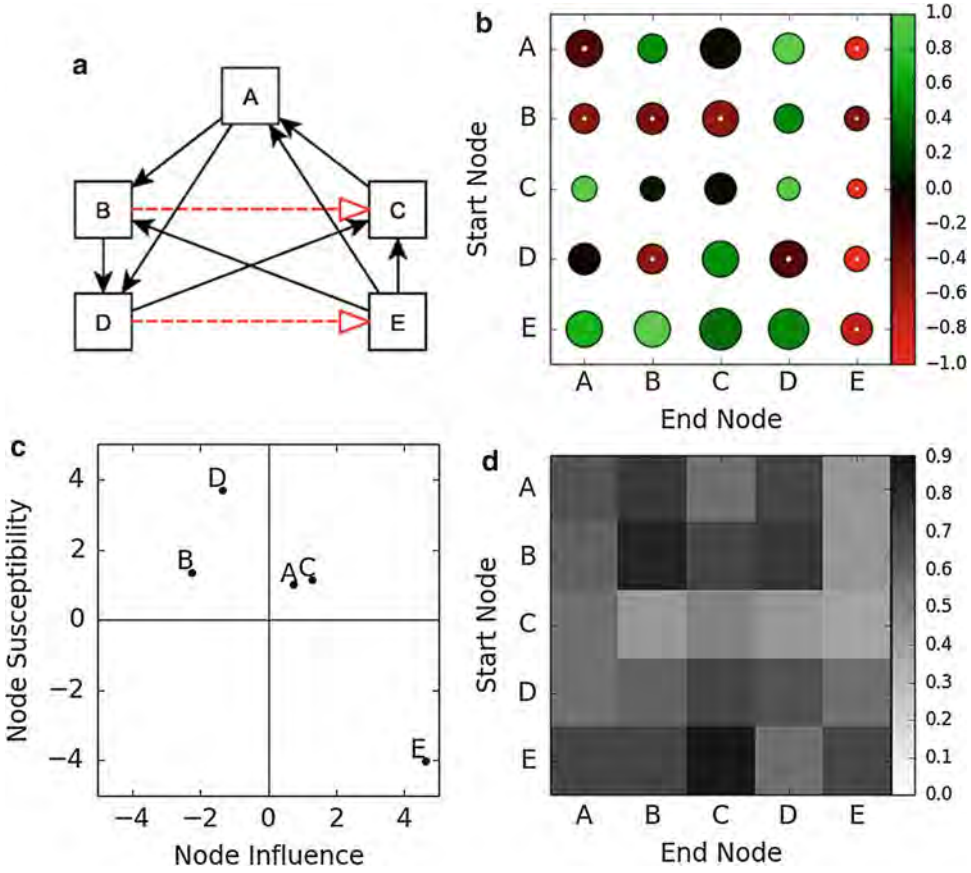
$$\sigma_j \equiv \sum_i \pi_{ij}\omega_{ij}.$$

The summative product results in large absolute values for these measures only when the regulation is both strong and consistent in sign. Nodes receiving low values are regulated weakly and/or in a complex way.

In cases where edge weights take on values other than ± 1, the p_{ijl} values may readily be modified to, for instance, the sum of the mean interaction weights of the pertinent paths. This modification reduces to the above definition when edge weights are restricted to ± 1. In both cases, however, the above measures are characterized by the parameter $l_{\max}$, which represents the longest path considered by the algorithm. Counting all paths of arbitrary length for all but the simplest networks is computationally intractable, and so in practice $l_{\max}$ must generally be a low number. We therefore introduce a complementary measure, **strength of connection**, which considers paths of arbitrary length through network erosion. The measure is determined for any two nodes i and j via a procedure that assigns every node a characteristic value. In the below pseudo-algorithm, these values are stored in a dictionary d.

```
N = number of nodes in graph
d = {}
for node in graph: d[node] = infinity

if i != j:
  while a path exists between i and j:
    SP = [nodes on the shortest path between i and j]
    SPL = length of shortest path between i and j
    if SPL == 1:
      delete edge between i and j
      d[i] = d[j] = SPL
    else:
      for node in SP:
        if d[node] == infinity: d[node] = SPL
        if (node != i and node != j): remove node from graph
  return sum(1/(values in d))/(N/2)
else:
    while a cycle containing i exists:
      SP = [nodes on the shortest cycle containing node i]
      SPL = length of shortest cycle containing node i
      if SPL == 1:
        delete self-edge
        d[i] = SPL
      else:
        for node in SP:
          if d[node] == infinity: d[node] = SPL
          if node != i: remove node from graph
  return sum(1/(values in d))/((N+2)/2)
```



Quantification of Regulation in Networks with Positive and Negative Interaction Weights, Fig. 2 Adapted from Figs. 1 and 2 of [7]. (**a**) A fully connected 5-node network. *Solid black arrows* indicate positive regulation, while *dashed*, *red arrows* indicate negative regulation. (**b**) A circle at position i, j has a size proportional to $\omega_{ij}\left(\max\left(\omega_{ij}\right) = 2.75\right)$ and color determined by π_{ij} / ω_{ij}, with positive, neutral, and negative sign corresponding to *green, black*, or *red* coloring, respectively. Circles are additionally identified with a small *white concentric circle* if $\pi_{ij} / \omega_{ij} \leq -0.2$. (**c**) A scatter plot of node path influence, ι, and node path susceptibility, σ. (**d**) The strength of connection measure indicates which node pairs remain well connected under network erosion; the values vary significantly despite each node having equal degree and the network being strongly connected

The normalization factors force the returned value to be bounded by 1 [7]. While this algorithm does not consider the sign of paths, it is straightforward to modify it to, e.g., include only those paths that are of the specified sign. Such a modification would then yield both a positive strength of connection and a negative strength of connection. We demonstrate the above-defined measures for a simple network in Fig. 2.

Applications

The analytical measures introduced above may be applied to any network with both positive and negative edge weights. Biological regulatory networks are a prime example of complex systems that are often modeled in this way. For example, the measures have been applied to explicate the regulatory cross talk of a network of the immune response responding

to both respiratory bacteria and allergen [7]. The measures stand to inform the dynamical regulation between nodes from a strictly topological perspective and thereby (1) provide insight into systems where the dynamic behavior is poorly understood and (2) complement dynamic analysis in systems where the regulatory behavior is understood.

Open Problems

The methodology discussed here considers the topology of a network with weighted positive and negative interactions. However, network analysis often involves an investigation of network dynamics, where the details of the interactions encoded in the network topology play a pivotal role. The role of network topology in constraining network dynamics is an active area of study (see, e.g., [8]).

Recommended Reading

1. Albert R, Barabási A-L (2002) Statistical mechanics of complex networks. Rev Mod Phys 74:47–97. doi:10.1103/RevModPhys.74.47
2. Newman MEJ (2010) Networks: an introduction. Oxford University Press, Oxford/New York
3. Newman MEJ (2012) Communities, modules and large-scale structure in networks. Nat Phys 8:25–31. doi:10.1038/nphys2162
4. Wang R-S, Saadatpour A, Albert R (2012) Boolean modeling in systems biology: an overview of methodology and applications. Phys Biol 9:055001. doi:10.1088/1478-3975/9/5/055001
5. Campbell C, Yang S, Albert R, Shea K (2011) A network model for plant–pollinator community assembly. Proc Natl Acad Sci 108:197–202. doi:10.1073/pnas.1008204108
6. Estrada E, Hatano N, Benzi M (2012) The physics of communicability in complex networks. Phys Rep 514:89–119. doi:10.1016/j.physrep.2012.01.006
7. Campbell C, Thakar J, Albert R (2011) Network analysis reveals cross-links of the immune pathways activated by bacteria and allergen. Phys Rev E 84:031929. doi:10.1103/PhysRevE.84.031929
8. Zañudo JGT, Albert R (2013) An effective network reduction approach to find the dynamical repertoire of discrete dynamic networks. Chaos Interdiscip J Nonlinear Sci 23:025111. doi:10.1063/1.4809777

Quantum Algorithm for Element Distinctness

Andris Ambainis
Faculty of Computing, University of Latvia, Riga, Latvia

Keywords

Element distinctness; Quantum algorithms; Quantum search; Quantum walks

Years and Authors of Summarized Original Work

2004; Ambainis

Problem Definition

In the *element distinctness* problem, one is given a list of N elements $x_1, \ldots, x_N \in \{1, \ldots, m\}$ and one must determine if the list contains two equal elements. Access to the list is granted by submitting queries to a black box, and there are two possible types of query.

Value Queries. In this type of query, the input to the black box is an index i. The black box outputs x_i as the answer. In the quantum version of this model, the input is a quantum state that may be entangled with the workspace of the algorithm. The joint state of the query, the answer register, and the workspace may be represented as $\sum_{i,y,z} a_{i,y,z}|i, y, z\rangle$, with y being an extra register which will contain the answer to the query and z being the workspace of the algorithm. The black box transforms this state into $\sum_{i,y,z} a_{i,y,z}|i, (y + x_i) \mod m, z\rangle$. The simplest particular case is if the input to the black box is of the form $\sum_i a_i|i, 0\rangle$. Then, the black box outputs $\sum_i a_i|i, x_i\rangle$. That is, a quantum state consisting of the index i is transformed into a quantum state, each component of which contains x_i together with the corresponding index i.

Comparison Queries. In this type of query, the input to the black box consists of two indices i, j. The black box gives one of the three possible answers: "$x_i > x_j$", "$x_i < x_j$," or "$x_i = x_j$." In the quantum version, the input is a quantum state consisting of basis states $|i, y, z\rangle$, with i, j being two indices and z being algorithm's workspace.

There are several reasons why the element distinctness problem is interesting to study. First of all, it is related to sorting. Being able to sort $x_1, \ldots, x_N$ enables one to solve the element distinctness by first sorting $x_1, \ldots, x_N$ in increasing order. If there are two equal elements $x_i = x_j$, then they will be the next one to another in the sorted list. Therefore, after one has sorted $x_1, \ldots, x_N$, one must only check the sorted list to see if each element is different from the next one. Because of this relation, the element distinctness problem captures some of the same difficulty as sorting. This has led to a long line of research on classical lower bounds for the element distinctness problem (cf. [5, 11, 21] and many other papers).

Second, the central concept of the algorithms for the element distinctness problem is the notion of a collision. This notion can be generalized in different ways, and its generalizations are useful for building quantum algorithms for various graph-theoretic problems (e.g., triangle finding [18]) and matrix problems (e.g., checking matrix identities [12]).

A generalization of element distinctness is element k-distinctness [3], in which one must determine if there exist k different indices $i_1, \ldots, i_k \in \{1, \ldots, N\}$ such that $x_{i1} = x_{i2} = \ldots = x_{ik}$. A further generalization is the k-subset finding problem [14], in which one is given a function $f(y_1, \ldots, y_k)$ and must determine whether there exist $i_1, \ldots, i_k \in \{1, \ldots, N\}$ such that $f(x_{i1}, x_{i2}, \ldots, x_{ik}) = 1$ (where $x_1, \ldots, x_N$ are the input data).

Key Results

Element Distinctness: Summary of Results

In the classical (non-quantum) context, the natural solution to the element distinctness problem is done by sorting, as described in the previous section. This uses $O(N)$ value queries (or $O(N \log N)$ comparison queries) and $O(N \log N)$ time. Any classical algorithm requires $\Omega(N)$ value or $\Omega(N \log N)$ comparison queries. If the algorithm is restricted to $o(N)$ space, stronger lower bounds are known [21].

In the quantum context, Buhrman et al. [13] gave the first nontrivial quantum algorithm, using $O(N^{3/4})$ queries. Ambainis [3] then designed a new algorithm, based on a novel idea using quantum walks. Ambainis' algorithm uses $O(N^{2/3})$ queries and is known to be optimal: Aaronson and Shi [1, 2, 15] have shown that any quantum algorithm for element distinctness must use $\Omega(N^{2/3})$ queries.

For quantum algorithms that are restricted to storing r values x_i (where $r < N^{2/3}$), the best algorithm runs in $O(N/\sqrt{r})$ time.

All of these results are for value queries. They can be adapted to the comparison query model, with a $\log N$ factor increase in the complexity. The time complexity is within a polylogarithmic $O(\log^c N)$ factor of the query complexity, as long as the computational model is sufficiently general [3]. (Random access quantum memory is necessary for implementing any of the known quantum algorithms.)

Using the quantum walk methods, one can also solve the k-distinctness problem [3]. This gives a quantum algorithm for k-distinctness (and k-subset finding) that uses $O(N^{k/(k+1)})$ value queries and $O(N^{k/(k+1)})$ memory. For the case when the memory is restricted to $r < N^{k/(k+1)}$ values of x_i, it suffices to use $O(r + (N^{k/2})/(r^{(k-1)/2}))$ value queries. The results generalize to comparison queries and time complexity, with a polylogarithmic factor increase in the time complexity (similarly to the element distinctness problem). For the k-subset finding problem, Belovs and Rosmanis [8] have shown that there is a function $f(y_1, \ldots, y_k)$ for which $\Omega(N^{k/(k+1)})$ queries are also necessary.

For the k-distinctness problem, a better quantum algorithm has been recently developed by Belovs [6], using the learning graph approach. It solves 3-distinctness using $O(N^{5/7})$ value

queries and k-distinctness using $O\left(N^{1-\frac{2^k-2}{2^k-1}}\right)$ value queries. The algorithm for 3-distinctness can be implemented so that it runs in time $O(N^{5/7}\log^c N)$ [9]. It is an open problem to construct a time-efficient implementation for $k > 3$.

Element Distinctness: The Methods

Ambainis' algorithm has the following structure. Its state space is spanned by basic states $|T\rangle$, for all sets of indices $T \subseteq \{1,\ldots,N\}$ with $|T| = r$. The algorithm starts in a uniform superposition of all $|T\rangle$ and repeatedly applies a sequence of two transformations:

1. Conditional phase flip: $|T\rangle \to -|T\rangle$ for all T such that T contains i, j with $x_i = x_j$, and $|T\rangle \to |T\rangle$ for all other T;
2. Quantum walk: perform $O(\sqrt{r})$ steps of quantum walk, as defined in [3]. Each step is a transformation that maps each $|T\rangle$ to a combination of basis states $|T'\rangle$ for T' that differ from T in one element.

The algorithm maintains another quantum register, which stores all the values of $x_i, i \in T$. This register is updated with every step of the quantum walk.

If there are two elements i, j such that $x_i = x_j$, repeating these two transformations $O(N/r)$ times increases the amplitudes of $|T\rangle$ containing i, j. Measuring the state of the algorithm at that point with high probability produces a set T containing i, j. Then, from the set T, we can find i and j.

The basic structure of [3] is similar to Grover's quantum search, but with one substantial difference. In Grover's algorithm, instead of using a quantum walk, one would use Grover's *diffusion transformation*. Implementing Grover's diffusion requires $\Omega(r)$ updates to the register that stores $x_i, i \in T$. In contrast to Grover's diffusion, each step of quantum walk changes T by one element, requiring just one update to the list of $x_i, i \in T$. Thus, $O(\sqrt{r})$ steps of quantum walk can be performed with $O(\sqrt{r})$ updates, quadratically better than Grover's diffusion. And, as shown in [3], the quantum walk provides a sufficiently good approximation of diffusion for the algorithm to work correctly.

This was one of the first uses of quantum walks to construct quantum algorithms. Ambainis, Kempe, and Rivosh [4] then generalized it to handle searching on grids (described in another entry of this encyclopedia). Their algorithm is based on the same mathematical ideas, but has a slightly different structure. Instead of alternating quantum walk steps with phase flips, it performs a quantum walk with two different walk rules – the normal walk rule and the "perturbed" one. (The normal rule corresponds to a walk without a phase flip and the "perturbed" rule corresponds to a combination of the walk with a phase flip.)

Generalization to Arbitrary Markov Chains

Szegedy [20] and Magniez et al. [19] have generalized the algorithms of [4] and [3], respectively, to speed up the search of an arbitrary Markov chain. The main result of [19] is as follows.

Let P be an irreducible Markov chain with state space X. Assume that some states in the state space of P are *marked*. Our goal is to find a marked state. This can be done by a classical algorithm that runs the Markov chain P until it reaches a marked state (Algorithm 1).

There are three costs that contribute to the complexity of Algorithm 1:

1. **Setup cost** S: the cost to sample the initial state x from the initial distribution.
2. **Update cost** U: the cost to simulate one step of a random walk.
3. **Checking cost** C: the cost to check if the current state x is marked.

The overall complexity of the classical algorithm is then $S + t_2(t_1 U + C)$. The required t_1 and t_2 can be calculated from the characteristics of the Markov chain P. Namely,

Proposition 1 ([19]) *Let P be an ergodic, yet symmetric Markov chain. Let $\delta > 0$ be the eigenvalue gap of P, and assume that whenever*

Algorithm 1: Search by a classical random walk

1. Initialize x to a state sampled from some initial distribution over the states of P.
2. t_2 times repeat:
   (a) If the current stage y is marked, output y and stop;
   (b) Simulate t_1 steps of random walk, starting with the current state y.
3. If the algorithm has not terminated, output "no marked state."

the set of marked states M is nonempty, we have $|M|/|X| \geq \epsilon$. Then there are $t_1 = O(1/\delta)$ and $t_2 = O(1/\epsilon)$ such that Algorithm 1 finds a marked element with high probability.

Thus, the cost of finding a marked element classically is $O(S + 1/\epsilon(1/\delta U + C))$. Magniez et al. [19] construct a quantum algorithm that finds a marked element in $O(S' + 1/\epsilon(1/\sqrt{\delta}U' + C'))$ steps, with S', U', and C' being quantum versions of the setup, update, and checking costs (in most of applications, these are of the same order as S, U, and C). This achieves a quadratic improvement in the dependence on both ε and δ.

The element distinctness problem is solved by a particular case of this algorithm: a search on the Johnson graph. The Johnson graph is the graph whose vertices v_T correspond to subsets $T \subseteq \{1, \ldots, N\}$ of size $|T| = r$. A vertex v_T is connected to a vertex v'_T, if the subsets T and T' differ in exactly one element. A vertex v_T is *marked* if T contains indices i, j with $x_i = x_j$.

Consider the following Markov chain on the Johnson graph. The starting probability distribution s is the uniform distribution over the vertices of the Johnson graph. In each step, the Markov chain chooses the next vertex v'_T from all vertices that are adjacent to the current vertex v_T, uniformly at random. While running the Markov chain, one maintains a list of all $x_i, i \in T$. This means that the costs of the classical Markov chain are as follows:

- Setup cost of $S = r$ queries (to query all $x_i, i \in T$ where v_T is the starting state).
- Update cost of $U = 1$ query (to query the value $x_i, i \in T' - T$, where v_T is the vertex before the step and v'_T is the new vertex).
- Checking cost of $C = 0$ queries (the values $x_i, i \in T$ are already known to the algorithm, and no further queries are needed).

The quantum costs S', U', and C' are of the same order as S, U, and C.

For this Markov chain, it can be shown that the eigenvalue gap is $\delta = O(1/r)$ and the fraction of marked states is $\epsilon = O((r^2)/(N^2))$. Thus, the quantum algorithm runs in time

$$O\left(S' + \frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}U' + C'\right)\right) = O\left(S' + \sqrt{r}\left(\frac{N}{r}U' + C'\right)\right) = O\left(r + \frac{N}{\sqrt{r}}\right).$$

Learning Graphs

Another framework that generalizes the element distinctness is the learning graphs by Belovs [6]. A learning graph is a structure that describes algorithm's information about the input data. Using this approach, many quantum algorithms can be described as sequences of high-level instructions (which can be compiled into a standard quantum query algorithm). For example, the element distinctness algorithm corresponds to a sequence of three operations:

1. Load $O(N^{2/3})$ values x_i for randomly chosen $i \in \{1, 2, \ldots, N\}$.
2. Load one of the two equal elements x_i.
3. Load the other equal element x_j.

Belovs [6] describes rules for determining the complexity of each step. In the algorithm above, the complexities are $O(N^{2/3})$, $O(\sqrt{N})$, and $O(N^{2/3})$, respectively. This results in the same overall complexity of $O(N^{2/3})$.

The learning graph approach has been used to construct new quantum algorithms for k-distinctness [7], triangle-finding [6], and other tasks.

Q

Applications

Magniez et al. [19] showed how to use the ideas from the element distinctness algorithm as a subroutine to solve the *triangle problem*. In the triangle problem, one is given a graph G on n vertices, accessible by queries to an oracle, and they must determine whether the graph contains a triangle (three vertices v_1, v_2, v_3 with $v_1 v_2$, $v_1 v_3$, and $v_2 v_3$ all being edges). This problem requires $\Omega(n^2)$ queries classically. Magniez et al. [19] showed that it can be solved using $O(n^{1.3} \log^c n)$ quantum queries, with a modification of the element distinctness algorithm as a subroutine. This has been improved by several authors. Currently, the best quantum algorithm for triangle finding is by Le Gall [17] which uses $O(n^{1.25} \log^c n)$ queries. It is also based on the quantum walks but uses them in a much more complex way.

The methods of Szegedy [20] and Magniez et al. [19] can be used as subroutines for quantum algorithms for checking matrix identities [12,18].

Bernstein et al. [10] have used the element distinctness algorithm to design a quantum algorithm for the subset sum problem, by combining the element distinctness algorithm with ideas from classical algorithms for subset sum. The resulting algorithm solves the subset sum problem for n numbers in $2^{(0.241+o(1))n}$ time steps, under some heuristic assumptions that are similar to the ones that are assumed for classical subset sum algorithms. The best classical algorithm uses $2^{(0.291+o(1))n}$ time steps.

Open Problems

1. How many queries are necessary to solve the element distinctness problem if the memory accessible to the algorithm is limited to r items, $r < N^{2/3}$? The algorithm of [3] gives $O(N/\sqrt{r})$ queries, and the best lower bound is $\Omega(N^{2/3})$ queries.
2. Consider the following problem:

   Graph collision [18]. The problem is specified by a graph G (which is arbitrary but known in advance) and variables $x_1, \ldots, x_N \in \{0, 1\}$, accessible by queries to an oracle. The task is to determine if G contains an edge uv such that $x_u = x_v = 1$. How many queries are necessary to solve this problem?

   The element distinctness algorithm can be adapted to solve this problem with $O(N^{2/3})$ queries [18], but there is no matching lower bound. Is there a better algorithm? A better algorithm for the graph collision problem would immediately imply a better algorithm for the triangle problem.

Cross-References

- Quantum Analogues of Markov Chains
- Quantum Algorithm for Finding Triangles
- Quantum Algorithm for Search on Grids
- Quantum Algorithms for Matrix Multiplication and Product Verification
- Quantum Search

Recommended Reading

1. Aaronson S, Shi Y (2004) Quantum lower bounds for the collision and the element distinctness problems. J ACM 51(4):595–605
2. Ambainis A (2005) Polynomial degree and lower bounds in quantum complexity: collision and element distinctness with small range. Theor Comput 1:37–46
3. Ambainis A (2007) Quantum walk algorithm for element distinctness. SIAM J Comput 37(1):210–239
4. Ambainis A, Kempe J, Rivosh A (2006) Coins make quantum walks faster. In: Proceedings of the ACM/SIAM symposium on discrete algorithms (SODA'06), Miami, pp 1099–1108
5. Beame P, Saks M, Sun X, Vee E (2003) Time-space trade-off lower bounds for randomized computation of decision problems. J ACM 50(2):154–195
6. Belovs A (2012) Span programs for functions with constant-sized 1-certificates: extended abstract. In: Proceedings of ACM symposium on theory of computing (STOC'12), New York, pp 77–84
7. Belovs A (2012) Learning-graph-based quantum algorithm for k-distinctness. In: Proceedings of IEEE conference on foundations of computer science (FOCS'12), New Brunswick, pp 207–216
8. Belovs A, Rosmanis A (2013) On the power of non-adaptive learning graphs. In: IEEE conference on computational complexity, Palo Alto, pp 44–55
9. Belovs A, Childs A, Jeffery S, Kothari R, Magniez F (2013) Time-efficient quantum walks for 3-distinctness. In: Proceedings of international colloquium on automata, languages and programming (ICALP'13), Riga, vol 1, pp 105–122

10. Bernstein D, Jeffery S, Lange T, Meurer A (2013) Quantum algorithms for the subset-sum problem. In: Proceedings of international workshop on post-quantum cryptography (PQCrypto'13), Limoges, pp 16–33
11. Borodin A, Fischer M, Kirkpatrick D, Lynch N (1981) A time-space tradeoff for sorting on non-oblivious machines. J Comput Syst Sci 22:351–364
12. Buhrman H, Spalek R (2006) Quantum verification of matrix products. In: Proceedings of the ACM/SIAM symposium on discrete algorithms (SODA'06), Miami, pp 880–889
13. Buhrman H, Durr C, Heiligman M, Høyer P, Magniez F, Santha M, de Wolf R (2005) Quantum algorithms for element distinctness. SIAM J Comput 34(6):1324–1330
14. Childs AM, Eisenberg JM (2005) Quantum algorithms for subset finding. Quantum Inf Comput 5:593
15. Kutin S (2005) Quantum lower bound for the collision problem with small range. Theor Comput 1:29–36
16. Le Gall F (2014, to appear) Improved quantum algorithm for triangle finding via combinatorial arguments. In: Proceedings of the IEEE symposium on foundations of computer science (FOCS 2014), Philadelphia
17. Magniez F, Nayak A (2005) Quantum complexity of testing group commutativity. In: Proceedings of the international colloquium automata, languages and programming (ICALP'05), Lisbon, pp 1312–1324
18. Magniez F, Santha M, Szegedy M (2007) Quantum algorithms for the triangle problem. SIAM J Comput 37(2):413–424
19. Magniez F, Nayak A, Roland J, Santha M (2007) Search by quantum walk. In: Proceedings of the ACM symposium on the theory of computing (STOC'07), San Diego, pp 575–584
20. Szegedy M (2004) Quantum speed-up of Markov Chain based algorithms. In: Proceedings of the IEEE conference on foundations of computer science (FOCS'04), Rome, pp 32–41
21. Yao A (1994) Near-optimal time-space tradeoff for element distinctness. SIAM J Comput 23(5):966–975

Quantum Algorithm for Factoring

Sean Hallgren
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, State College, PA, USA

Years and Authors of Summarized Original Work

1994; Shor

Problem Definition

Every positive integer n has a unique decomposition as a product of primes $n = p_1^{e_1} \cdots p_k^{e_k}$, for prime number p_i, and positive integer exponent e_i. Computing the decomposition $p_1, e_1, \ldots, p_k, e_k$ from n is the factoring problem.

Factoring has been studied for many hundreds of years, and exponential time algorithms for it were found to include trial division, Lehman's method, Pollard's ρ method, and Shank's class group method [1]. With the invention of the RSA public-key cryptosystem in the late 1970s, the problem became practically important and started receiving much more attention. The security of RSA is closely related to the complexity of factoring, and in particular, it is only secured if factoring does not have an efficient algorithm. The first subexponential-time algorithm is due to Morrison and Brillhard [4] using a continued fraction algorithm. This was succeeded by the quadratic sieve method of Pomerance and the elliptic curve method of Lenstra [5]. The number field sieve [2, 3], found in 1989, is the best-known classical algorithm for factoring and runs in time $\exp(c(\log n)^{1/3}(\log \log n)^{2/3})$ for some constant c. Shor's result is a polynomial-time quantum algorithm for factoring.

Key Results

Theorem 1 ([2, 3]) *There is a subexponential-time classical algorithm that factors the integer n in time* $\exp(c(\log n)^{1/3}(\log \log n)^{2/3})$.

Theorem 2 ([6]) *There is a polynomial-time quantum algorithm that factors integers. The algorithm factor n in time $O((\log n)^2(\log n \log n)(\log \log \log n))$ plus polynomial in $\log n$ post-processing which can be done classically.*

Applications

Computationally hard number theoretic problems are useful for public-key cryptosystems.

The RSA public-key cryptosystem, as well as others, requires that factoring not to have an efficient algorithm. The best-known classical algorithms for factoring can help determine how secure the cryptosystem is and what key sizes to choose. Shor's quantum algorithm for factoring can break these systems in polynomial time using a quantum computer.

Open Problems

It is open whether there is a polynomial-time classical algorithm for factoring.

Cross-References

- ▸ Quantum Algorithm for Solving Pell's Equation
- ▸ Quantum Algorithm for the Discrete Logarithm Problem
- ▸ Quantum Algorithms for Class Group of a Number Field

Recommended Reading

1. Cohen H (1993) A course in computational algebraic number theory. Graduate texts in mathematics, vol 138. Springer, Berlin/Heidelberg/New York
2. Lenstra A, Lenstra H (eds) (1993) The development of the number field sieve. Lecture notes in mathematics, vol 1544. Springer, Berlin
3. Lenstra AK, Lenstra HW Jr, Manasse MS, Pollard JM (1990) The number field sieve. In: Proceedings of the twenty second annual ACM symposium on theory of computing, Baltimore, 14–16 May 1990, pp 564–572
4. Morrison M, Brillhart J A method of factoring and the factorization of F_7
5. Pomerance C Factoring. In: Pomerance C (ed) Cryptology and computational number theory. Proceedings of symposia in applied mathematics, vol 42. American Mathematical Society, Providence, p 27
6. Shor PW (1997) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J Comput 26:1484–1509

Quantum Algorithm for Finding Triangles

Stacey Jeffery[1] and Peter C. Richter[2,3]
[1]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada
[2]Department of Combinatorics and Optimization, and Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada
[3]Department of Computer Science, Rutgers, The State University of New Jersey, New Brunswick, NJ, USA

Keywords

Quantum query complexity; Triangle finding

Years and Authors of Summarized Original Work

2014; Le Gall

Problem Definition

A *triangle* is a clique of size three in an undirected graph. Triangle finding has been the subject of extensive study as a basic search problem whose quantum query complexity is still open, in contrast to unstructured search [6] and element distinctness [1].

This survey concerns quantum query algorithms for triangle finding. A *quantum query algorithm* for a search problem $P = \{M_f\}_f$ is a sequence of unitary operators $Q_f = U_k O_f U_{k-1} O_f U_1 O_f U_0$ such that if $M_f \neq \emptyset$, measuring $|\psi_f\rangle = Q_f|0\rangle$ yields a member of M_f, the set of objects associated with input f, with probability $\geq 2/3$. The operators O_f are *oracle queries*, $O_f : |x\rangle|a\rangle \mapsto |x\rangle|a \oplus f(x)\rangle$, which yield information about f, whereas the U_j are independent of f. The quantum query complexity of P is the minimum number of

oracle queries required by a quantum query algorithm for P.

In the context of triangle finding, the function f is the adjacency matrix of an undirected graph on vertices $[n]$, $G \subseteq [n]^2$, with $m = |G|$ edges, where $(a, b) \in G \Rightarrow (b, a) \in G$, by convention. The associated set, M_G, is the set of triangles in G.

Problem 1 (Triangle finding)

INPUT: The adjacency matrix f of a graph G on n vertices.

OUTPUT: A triangle: $(a, b, c) \in [n]^3$ such that $(a, b), (b, c), (a, c) \in G$, if one exists.

A lower bound of $\Omega(n)$ on the quantum query complexity of the triangle finding problem follows from a reduction from search [5]. It is easy to see that the randomized query complexity of the triangle finding problem is $\Theta(n^2)$.

Key Results

Progress on the quantum query complexity of triangle finding has closely followed the development of quantum algorithmic techniques for search problems. The first upper bounds were based on increasingly clever use of the structure of the problem, combined with amplitude amplification [4]. The first bound to go beyond the amplitude amplification framework, achieving $\tilde{O}(n^{13/10})$ [10], was one of the first applications of the quantum walk search technique introduced by Ambainis in his element distinctness algorithm [1] and extended in [13] and [11]. The next bound of $O(n^{35/27})$ was the first application of a new quantum algorithmic technique, the learning graph framework [3]. This finding led to the development of extensions to the quantum walk search technique to give a $\tilde{O}(n^{35/27})$ quantum walk algorithm for triangle finding [7]. The next improvement to $O(n^{9/7})$ also used the learning graph framework [9], whereas the most recent upper bound of $O(n^{5/4})$ uses, once again, a quantum walk search algorithm [8].

An $O(n + \sqrt{nm})$ Algorithm Using Amplitude Amplification

A trivial application of Grover's quantum search algorithm solves the triangle finding problem with $O(n^{3/2})$ quantum queries by searching over $[n]^3$. Buhrman et al. [5] improved this upper bound in the special case where G is sparse (i.e., $m = o(n^2)$).

The algorithm searches for an edge $(a, b) \in G$ in $O(\sqrt{|[n]^2|/m}) = O(n/\sqrt{m})$ quantum queries and then for $c \in [n]$ such that (a, b, c) is a triangle in $O(\sqrt{n})$ quantum queries. The second step succeeds when (a, b) is a triangle edge, which happens with probability at least $1/m$ when G contains a triangle, so applying amplitude amplification to this procedure gives a $O(\sqrt{m}(n/\sqrt{m} + \sqrt{n})) = O(n + \sqrt{nm})$ upper bound:

Theorem 1 (Buhrman et al. [5]) *Using quantum amplitude amplification, the triangle finding problem can be solved in $O(n + \sqrt{nm})$ quantum queries.*

An $\tilde{O}(n^{10/7})$ Algorithm Using Amplitude Amplification

The algorithm of Szegedy et al. [10, 12] is also based on amplitude amplification; however, it exploits additional combinatorial structure in the triangle finding problem.

For $A \subseteq [n]$ and $w \in [n]$, define $\Delta_G(A, w) := \{(u, v) \in A^2 : (u, w), (v, w) \in G\}$. Choose a random subset $X \subseteq [n]$ of size $n^\chi \log n$, for $\chi = 3/7$. Query $X \times [n]$ and search for an edge in $E_X := \bigcup_{w \in X} \Delta_G([n], w)$, which can be determined from $G \cap (X \times [n])$, using $O(|X|n + \sqrt{|E_X|}) = \tilde{O}(n^{1+\chi})$ queries. Either a triangle is found, or $E_X \cap G = \emptyset$.

Let $G' := [n]^2 \setminus E_X$. If a triangle is not found in the first step, then $G \subseteq G'$. Fix $\alpha = \beta = 1/7$. Szegedy et al. show that for most X, G' can be partitioned into (T, E), such that T has $O(n^{3-\alpha})$ triangles and $|E \cap G| = O(n^{2-\beta} + n^{2-\chi+\alpha+\beta})$, in $\tilde{O}(n^{1+\alpha+\beta})$ queries (or a triangle is found in the process). If $G \subseteq G'$, any triangle in G either lies in T, in which case it can be found in $O(\sqrt{n^{3-\alpha}})$ queries using quantum search, or intersects E, in which case it can be found in

$O(n + \sqrt{n|G \cap E|})$ queries using the algorithm of Buhrman et al. This gives the following:

Theorem 2 (Szegedy; Magniez, Santha and Szegedy [10, 12]) *Using amplitude amplification, the triangle finding problem can be solved in* $\tilde{O}(n^{10/7})$ *quantum queries.*

An $\tilde{O}(n^{13/10})$ Algorithm Using Quantum Walks

A more efficient algorithm for the triangle finding problem was obtained by Magniez et al. [10], using the quantum walk search technique introduced by Ambainis [1].

Given oracle access to a function defining a relation $M \subseteq [n]^k$, Ambainis' quantum walk search procedure finds $(a_1, \ldots, a_k) \in M$ if $M \neq \emptyset$. The algorithm walks on sets $A \subseteq [n]$ of size n^α, keeping track of some data structure $D(A)$ for the current state A and transitioning, in superposition, from A to A' for $A' \subseteq [n]$ of size n^α such that $|A \setminus A'| = 1$. Assume access to a quantum procedure Φ that determines if $A^k \cap M \neq \emptyset$ using $D(A)$, with *checking cost* C queries. Suppose $D(A)$ can be constructed from scratch at *setup cost* S queries and modified from $D(A)$ to $D(A')$ when $|A \setminus A'| = 1$ at an *update cost* U. Then the procedure finds an element of M in $O(\mathsf{S} + (\frac{n}{n^\alpha})^{k/2}(\sqrt{n^\alpha}\mathsf{U} + \mathsf{C}))$ quantum queries. (For details, see the encyclopedia entry on element distinctness.)

For a fixed graph $G \subseteq [n]^2$, consider the *graph collision* problem on G, where an input f defines the binary relation $M_f \subseteq [n]^2$ satisfying $(u, u') \in M_f$ if $f(u) = f(u') = 1$ and $(u, u') \in G$. Setting $k = 2$, it is a simple exercise to see that a quantum walk search algorithm solves this problem with $O(n^\alpha + (\frac{n}{n^\alpha})(\sqrt{n^\alpha} \cdot 1 + 0)) = O(n^\alpha + n^{1-\alpha/2})$ queries. Setting $\alpha = 2/3$ gives an upper bound of $O(n^{2/3})$ quantum queries for graph collision.

Magniez et al. [10] solve triangle finding using a quantum walk algorithm whose checking subroutine is based on graph collision. Let M be the set of triangle edges. Define $D(A) = G \cap A^2$. Then $\mathsf{S} = n^{2\alpha}$ initial queries are needed to set up $D(A)$, and $\mathsf{U} = n^\alpha$ new queries are needed to update $D(A)$, where α is now $3/5$. The checking step consists of an algorithm that, given a known subgraph $H = G \cap A^2$ on n^α vertices, decides if H contains a triangle edge using $\mathsf{C} = \tilde{O}(\sqrt{n}n^{2/3\alpha})$ queries, as follows. For any $v \in [n]$, define f_v on A by $f_v(u) = 1$ if $(u, v) \in G$. An edge $(a, b) \in A^2$ is a graph collision in f_v on $G \cap A^2$ if and only if (a, b, v) is a triangle, so searching for $v \in [n]$ for which f_v has a graph collision, using $O(\sqrt{n}(n^\alpha)^{2/3})$ quantum queries, is equivalent to deciding if $G \cap A^2$ contains a triangle edge. Repeat $\Theta(\log n)$ times, to decrease the error to $1/n^{\Theta(1)}$, since the subroutine is called many times. This gives the following:

Theorem 3 (Magniez, Santha, and Szegedy [10]) *Using a quantum walk search procedure, the triangle finding problem can be solved in* $\tilde{O}(n^{13/10})$ *quantum queries.*

An $O(n^{35/27})$ Algorithm Using Learning Graphs

The learning graph framework, introduced by Belovs [3], allows for the construction of a quantum algorithm from a particular type of edge-weighted graph called a *learning graph*. For further details, refer to [3]. The first application of this framework was a new upper bound on the quantum query complexity of triangle finding.

A learning graph may be constructed in stages, corresponding to searching for more and more specialized structures, which will eventually contain a 1-certificate for the problem being solved. In Belovs' application to triangle finding, the first part of the learning graph corresponds to searching for an n^α-vertex subset of $[n]$, A, containing two triangle vertices a and b. The next two stages correspond to searching for an $n^{2\alpha-\sigma}$-edge graph on A, H, which contains the triangle edge $\{a, b\}$. The final stages correspond to the graph collision subroutine used in [10] to decide if any edge of the queried subgraph H is a triangle edge. Using $\alpha = 2/3$ and $\sigma = 1/27$ gives the following:

Theorem 4 (Belovs [3]) *Using a learning graph algorithm, the triangle finding problem can be solved in* $O(n^{35/27}) = O(n^{1.2963})$ *quantum queries.*

Additionally, a quantum walk search algorithm based on this learning graph construction solves triangle finding in $\tilde{O}(n^{35/27})$ queries [7].

An $O(n^{9/7})$ Algorithm Using Learning Graphs

The next upper bound on the quantum query complexity of triangle finding, due to Lee et al. [9], also uses a learning graph. The first part of their learning graph corresponds to searching for an n^{α}-vertex subset $A \subseteq [n]$, containing a triangle vertex a. The next part corresponds to searching for an n^{β}-vertex subset $B \subseteq [n]$, containing a vertex, b, from the same triangle as a. The final part corresponds to the graph collision subroutine used in [10], but optimized for an unbalanced bipartite graph, used to decide if any edge of $G \cap (A \times B)$ is a triangle edge. Using $\alpha = 4/7$ and $\beta = 5/7$ gives the following:

Theorem 5 (Lee, Magniez and Santha [9]) *Using a learning graph algorithm, the triangle finding problem can be solved in* $O(n^{9/7}) = O(n^{1.2858})$ *quantum queries.*

As with the previous algorithm, there exists a quantum walk search algorithm based on this learning graph construction that solves triangle finding in $\tilde{O}(n^{9/7})$ queries [7].

An $\tilde{O}(n^{5/4})$ Algorithm Using Quantum Walks

The best known upper bound on the quantum query complexity of triangle finding is an algorithm by Le Gall [8]. Le Gall's algorithm uses the quantum walk search technique, as in the $\tilde{O}(n^{13/10})$-query algorithm, combined with a more clever utilization of the combinatorial structure of triangle finding, similar to that of the $\tilde{O}(n^{10/7})$-query algorithm, and a quantum search algorithm of Ambainis that finds an x such that $\Phi(x) = 1$ in cost $O(\sqrt{\sum_x C(x)^2})$, where $C(x)$ is the cost to compute $\Phi(x)$ [2].

The algorithm begins, like the $\tilde{O}(n^{10/7})$ algorithm, by choosing a random $X \subseteq [n]$ of size $n^{\chi} \log n$ and searching for a triangle in $X \times [n]^2$. This is done by quantum search on $X \times [n]^2$, using $O(\sqrt{|X \times [n]^2|}) = \tilde{O}(n^{1+\chi/2})$ quantum queries. If no triangle is found, as in the $\tilde{O}(n^{10/7})$ algorithm, the rest of the algorithm will make use of the fact that $E_X \cap G = \emptyset$, although in this case, since $X \times [n]$ is not queried, E_X is not known.

The rest of the algorithm consists of the following four levels of recursion:

1. Using a quantum walk search algorithm, search for a set $A \subseteq [n]$ of size n^{α} such that A^2 contains a triangle edge. Maintain a data structure, $D(A)$, encoding $G \cap (A \times X)$.
2. For any $A \subseteq [n]$, to check if A^2 contains a triangle edge, search for a vertex $c \in [n]$ such that $A^2 \times \{c\}$ contains a triangle.
3. For any $A \subseteq [n]$ and $c \in [n]$, to check if $A^2 \times \{c\}$ contains a triangle, use a quantum walk search algorithm to search for a set $B \subseteq A$ of size n^{β} such that $B^2 \times \{c\}$ contains a triangle. Maintain a data structure, $D^c(B)$, encoding $G \cap (B \times \{c\})$.
4. For any $B \subseteq [n]$ and $c \in [n]$, to check if $B^2 \times \{c\}$ contains a triangle, search for an edge in $\Delta_G(B, c) \setminus E_X$. Here the algorithm exploits the fact that there is no edge in E_X. The set $E_X \cap B^2$ can be determined from $G \cap (A \times X)$.

Constructing $D(A)$ costs $\mathsf{S} = |A \times X| = \tilde{O}(n^{\alpha+\chi})$ queries. Mapping $D(A)$ to $D(A')$ costs $\mathsf{U} = 2|X| = \tilde{O}(n^{\chi})$ queries. Let $\Phi(A) = 1$ if A^2 has a triangle edge. Then if C is the quantum query complexity of computing $\Phi(A)$, the quantum query complexity of finding a triangle in $G \setminus E_X$ is $O\left(\mathsf{S} + \frac{n}{n^{\alpha}}\left(\sqrt{n^{\alpha}}\mathsf{U} + \mathsf{C}\right)\right) = \tilde{O}\left(n^{\alpha+\chi} + n^{1+\chi-\alpha/2} + n^{1-\alpha}\mathsf{C}\right)$.

Let $\Phi_A(c) = 1$ if $A^2 \times \{c\}$ contains a triangle. To compute $\Phi(A)$, search for $c \in [n]$ such that $\Phi_A(c) = 1$. Let $\mathsf{C}'(c)$ be the cost of computing $\Phi_A(c)$, which will vary in c. Then by [2], $\mathsf{C} = O\left(\sqrt{\sum_{c \in [n]} \mathsf{C}'(c)}\right)$.

Let $\Phi^c(D^c(B)) = 1$ if $B^2 \times \{c\}$ has a triangle. To compute $\Phi_A(c)$, search for $B \subseteq A$ such that $\Phi^c(D^c(B)) = 1$. Creating $D^c(B)$ costs $\mathsf{S}'' = |B \times \{c\}| = n^{\beta}$ queries. Mapping $D^c(B)$ to $D^c(B')$ costs $\mathsf{U}'' = 2$ queries. If computing $\Phi^c(D^c(B))$ costs $\mathsf{C}''(c)$, computing $\Phi_A(c)$ costs $\mathsf{C}'(c) = \mathsf{S}'' + \frac{n^{\alpha}}{n^{\beta}}\left(\sqrt{n^{\beta}}\mathsf{U}'' + \mathsf{C}''(c)\right) = O\left(n^{\beta} + n^{\alpha-\beta/2} + n^{\alpha-\beta}\mathsf{C}''(c)\right)$ queries.

Observe that $\Phi^c(D^c(B)) = 1$ if and only if $\Delta_G(B,c)$ contains an edge. Since $G \cap E_X = \emptyset$, one need only search $\Delta_G(B,c) \setminus E_X$ for an edge. The set $\Delta_G(B,c)$ can be determined from $D^c(B)$, and $E_X \cap A^2$ can be determined from $D(A)$, so $\Delta_G(B,c) \setminus E_X$ is known. Thus, $\mathsf{C}''(c) = O(\sqrt{|\Delta_G(B,c) \setminus E_X|})$.

Using combinatorial arguments, Le Gall proves an upper bound on $|\Delta_G(B,c) \setminus E_X|$ relative to $|\Delta_G(A,c) \setminus E_X|$ for *most* B, allowing him to use further combinatorial arguments to show an upper bound on $\mathsf{C} = \sqrt{\sum_{c\in[n]} \mathsf{C}'(c)^2}$ of $O(n^{1/2+\chi} + n^{1/2+\beta} + n^{1/2+\alpha-\beta/2} + n^{1/2+\alpha-\chi/2})$. Setting $\chi = \beta = 1/2$ and $\alpha = 3/4$ then gives the following:

Theorem 6 (Le Gall [8]) *Using a quantum walk search algorithm, the triangle finding problem can be solved in $\tilde{O}(n^{5/4})$ quantum queries.*

The quantum query complexity of triangle finding is still open, as the best known lower bound is $\Omega(n)$.

Cross-References

- Quantum Algorithm for Element Distinctness
- Quantum Algorithm for the Collision Problem
- Quantum Analogues of Markov Chains

Recommended Reading

1. Ambainis A (2007) Quantum walk algorithm for element distinctness. SIAM J Comput 37(1):210–239
2. Ambainis A (2010) Quantum search with variable times. Theory Comput Syst 47(3):786–807
3. Belovs A (2012) Span programs for functions with constant-sized 1-certificates. In: Proceeding of STOC, New York, pp 77–84
4. Brassard G, Høyer P, Mosca M, Tapp A (2002) Quantum amplitude amplification and estimation. In: Quantum computation and quantum information: a millennium volume. AMS contemporary mathematics series millennium volume, vol 305. American Mathematical Society, Providence, pp 53–74
5. Buhrman H, Dürr C, Heiligman M, Høyer P, Santha M, Magniez F, de Wolf R (2005) Quantum algorithms for element distinctness. SIAM J Comput 34(6):1324–1330
6. Grover LK (1996) A fast quantum mechanical algorithm for database search. In: Proceeding of STOC, Philadelphia, pp 212–219
7. Jeffery S, Kothari R, Magniez F (2013) Nested quantum walks with quantum data structures. In: Proceeding of SODA, New Orleans, pp 1474–1485
8. Le Gall F (2014) Improved quantum algorithm for triangle finding via combinatorial arguments. In: Proceeding of FOCS, Philapelphia, pp 216–225. quant-ph/1407.0085
9. Lee T, Magniez F, Santha M (2013) Improved quantum query algorithms for triangle finding and associativity testing. In: Proceeding of SODA, New Orleans, pp 1486–1502
10. Magniez F, Santha M, Szegedy M (2007) Quantum algorithms for the triangle problem. SIAM J Comput 37(2):413–424
11. Magniez F, Nayak A, Roland J, Santha M (2011) Search via quantum walk. SIAM J Comput 40(1):142–164. quant-ph/0608026
12. Szegedy M (2003) On the quantum query complexity of detecting triangles in graphs. quant-ph/0310107
13. Szegedy M (2004) Quantum speed-up of Markov chain based algorithms. In: Proceeding of FOCS, Rome, pp 32–41

Quantum Algorithm for Search on Grids

Andris Ambainis
Faculty of Computing, University of Latvia, Riga, Latvia

Keywords

Spatial search

Years and Authors of Summarized Original Work

2005; Ambainis, Kempe, Rivosh

Problem Definition

Consider an $\sqrt{N} \times \sqrt{N}$ grid, with each location storing a bit that is 0 or 1. The locations on the grid are indexed by (i, j), where $i, j \in$

$\{0, 1, \ldots, \sqrt{N}-1\} \cdot a_{i,j}$ denotes the value stored at the location (i, j).

The task is to find a location storing $a_{i,j} = 1$. This problem is as an abstract model for search in a two-dimensional database, with each location storing a variable $x_{i,j}$ with more than two values. The goal is to find $x_{i,j}$ that satisfies certain constraints. One can then define new variables $a_{i,j}$ with $a_{i,j} = 1$ if $x_{i,j}$ satisfies the constraints and search for i, j satisfying $a_{i,j} = 1$.

The grid is searched by a "robot," which at any moment of time is at one location i, j. In one time unit, the robot can either examine the current location or move one step in one of the four directions (left, right, up, or down).

In a probabilistic version of this model, the robot is probabilistic. It makes its decisions (querying the current location or moving) randomly according to prespecified probability distributions. At any moment of time, such a robot is at a probability distribution over the locations of the grid. In the quantum case, one has a "quantum robot" [5] which can be in a quantum superposition of locations (i, j) and is allowed to perform transformations that move it at most one step at a time.

There are several ways to make this model of a "quantum robot" precise [1] and they all lead to similar results.

The simplest to define is the Z-local model of [1]. In this model, the robot's state space is spanned by states $|i, j, a\rangle$ with i, j representing the current location and a being the internal memory of the robot. The robot's state $|\psi\rangle$ can be any quantum superposition of those: $|\psi\rangle = \sum_{i,j,a} \alpha_{i,j,a} |i, j, a\rangle$, where $\alpha_{i,j,a}$ are complex numbers such that $\sum_{i,j,a} |\alpha_{i,j,a}|^2 = 1$. In one step, the robot can either perform a query of the value at the current location or a Z-local transformation.

A query is a transformation that leaves i, j parts of a state $|i, j, a\rangle$ unchanged and modifies the a part in a way that depends only on the value $a_{i,j}$. A Z-local transformation is a transformation that maps any state $|i, j, a\rangle$ to a superposition that involves only states with robot being either at the same location or at one of the four adjacent locations ($|i, j, b\rangle$, $|i-1, j, b\rangle$, $|i+1, j, b\rangle$, $|i, j-1, b\rangle$ or $|i, j+1, b\rangle$ where the content of the robot's memory b is arbitrary).

The problem generalizes naturally to d-dimensional grid of size $N^{1/d} \times N^{1/d} \times \ldots \times N^{1/d}$, with robot being allowed to query or move one step in one of the d directions in one unit of time.

Key Results

Early Results

This problem was first studied by Benioff [5] who considered the use of the usual quantum search algorithm by Grover [9] in this setting. Grover's algorithm allows to search a collection of N items $a_{i,j}$ with $O(\sqrt{N})$ queries. However, it does not respect the structure of a grid. Between any two queries, it performs a transformation that may require the robot to move from any location (i, j) to any other locations (i', j'). In the robot model, where the robot in only allowed to move one step in one time unit, such transformation requires $O(\sqrt{N})$ steps to perform. Implementing Grover's algorithm, which requires $O(\sqrt{N})$ such transformations, therefore, takes $O(\sqrt{N}) \times O(\sqrt{N}) = O(N)$ time, providing no advantage over the naive classical algorithm.

The first algorithm improving over the naive use of Grover's search was proposed by Aaronson and Ambainis [1] who achieved the following results:

- Search on $\sqrt{N} \times \sqrt{N}$ grid, if it is known that the grid contains exactly one $a_{i,j} = 1$ in $O(\sqrt{N} \log^{3/2} N)$ steps.
- Search on $\sqrt{N} \times \sqrt{N}$ grid, if the grid may contain an arbitrary number of $a_{i,j} = 1$ in $O(\sqrt{N} \log^{5/2} N)$ steps.
- Search on $N^{1/d} \times N^{1/d} \times \ldots \times N^{1/d}$ grid, for $d \geq 3$, in $O(\sqrt{N})$ steps.

They also considered a generalization of the problem, search on a graph G, in which the robot moves on the vertices v of the graph G and searches for a variable $a_v = 1$. In one step, the

robot can examine the variable a_v corresponding to the current vertex v or move to another vertex w adjacent to v. Aaronson and Ambainis [1] gave an algorithm for searching an arbitrary graph with grid-like expansion properties in $O(N^{1/2+o(1)})$ steps. The main technique in those algorithms was the use of Grover's search and its generalization, amplitude amplification [6], in combination with "divide-and-conquer" methods recursively breaking up a grid into smaller parts.

Quantum Walks

The next algorithms were based on quantum walks [3,7,8]. Ambainis, Kempe, and Rivosh [3] presented an algorithm, based on a discrete time quantum walk, which searches the two-dimensional $\sqrt{N} \times \sqrt{N}$ in $O(\sqrt{N} \log N)$ steps, if the grid is known to contain exactly one $a_{i,j} = 1$ and in $O(\sqrt{N} \log^2 N)$ steps in the general case. Childs and Goldstone [8] achieved a similar performance, using continuous time quantum walk. Curiously, it turned out that the performance of the walk crucially depended on the particular choice of the quantum walk, both in the discrete and continuous time, and some very natural choices of quantum walk (e.g., one in [7]) failed.

Besides providing an almost optimal quantum speedup, the quantum walk algorithms also have an additional advantage: their simplicity. The discrete quantum walk algorithm of [3] uses just two bits of quantum memory. Its basis states are $|i, j, d\rangle$, where (i, j) is a location on the grid and d is one of the four directions: $\leftarrow$, $\rightarrow$, $\uparrow$, and $\downarrow$. The basic algorithm consists of the following simple steps:

1. Generate the state $\sum_{i,j,d} \frac{1}{2\sqrt{N}} |i, j, d\rangle$.
2. $O(\sqrt{N \log N})$ times repeat
   1. Perform the transformation

$$C_0 = \begin{pmatrix} -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \end{pmatrix}$$

   2. On the states $|i, j, \leftarrow\rangle$, $|i, j, \rightarrow\rangle$, $|i, j, \uparrow\rangle$, $|i, j, \downarrow\rangle$, if $a_{i,j} = 0$ and the transformation $C_1 = -I$ on the same four states if $a_{i,j} = 1$.
   3. Move one step according to the direction register and reverse the direction:

$$|i, j, \rightarrow\rangle \rightarrow |i + 1, j, \leftarrow\rangle,$$
$$|i, j, \leftarrow\rangle \rightarrow |i - 1, j, \rightarrow\rangle,$$
$$|i, j, \uparrow\rangle \rightarrow |i, j - 1, \downarrow\rangle,$$
$$|i, j, \downarrow\rangle \rightarrow |i, j + 1, \uparrow\rangle.$$

In case, if $a_{i,j} = 1$ for one location (i, j), a significant part of the algorithm's final state will consist of the four states $|i, j, d\rangle$ for the location (i, j) with $a_{i,j} = 1$. This can be used to detect the presence of such location. More precisely, if we run the algorithm for $O(\sqrt{N \log N})$ steps and measure the state, we obtain one of the four states $|i, j, d\rangle$ with probability $\Theta(1/\log N)$.

We can increase the probability of algorithm finding the right location *(i, j)* by either repeating the algorithm or using quantum amplitude amplification. Quantum amplitude amplification [6] takes a quantum algorithm that succeeds with a small probability ε and increases the success probability to 3/4, by repeating the quantum algorithm $O(1/\sqrt{\epsilon})$ times. In our case, $\epsilon = \Theta(1/\log N)$ which means that it suffices to repeat the basic algorithm $O(\sqrt{\log N})$ times. This increases the running time from $O(\sqrt{N \log N})$ for the basic algorithm to $O(\sqrt{N} \log N)$.

A quantum algorithm for search on a grid can be also derived by designing a classical algorithm that finds $a i, j = 1$ by performing a random walk on the grid and then applying Szegedy's general translation of classical random walks to quantum random chains, with a quadratic speedup over the classical random walk algorithm [15]. The resulting algorithm is similar to the algorithm of [3] described above and has the same running time.

For an overview on related quantum algorithms using similar methods, see [2,10].

Further Developments

The running time of the algorithm has been improved to $O(\sqrt{N \log N})$ time steps if the grid is known to contain exactly one *(i, j)* with $a_{i,j} = 1$ and $O(\sqrt{N} \log^{1.5} N)$ steps in the general case. This can be achieved in two different ways. First, Tulsi [16] showed how to modify the quantum walk so that, after $O(\sqrt{N \log N})$ steps, it finds the right *(i, j)* with a constant probability. This eliminates the need to use amplitude amplification.

Second, Ambainis et al. [4] showed that the same result can be achieved without modifying the quantum walk, by a simple classical postprocessing. That is, even if the quantum walk does not find the right *(i, j)*, its final state is much more likely to be (i', j') that is close to *(i, j)*. One can then run the quantum walk for $O(\sqrt{N \log N})$ steps once, measure the result, obtain a location (i', j'), and search the nearby locations for *(i, j)* with $a_{i,j} = 1$.

Search algorithms similar to the original 2D search algorithm have been analyzed for a number of other graphs (e.g., for hierarchical networks [12]).

Applications

The quantum algorithm for search on the grid by Ambainis, Kempe, and Rivosh [3] has been generalized by Szegedy [15], obtaining a general procedure for speeding up classical Markov chains (described in more detail in the article on Quantization of Markov Chains). Szegedy's generalization concerns a class of algorithms called *Search by Random Walk* in which one performs a random walk on some search space until finding an element with a certain property. Szegedy [15] showed that if a classical random walk finds a marked element in T steps (on average), there is a quantum algorithm that detects the existence of a marked element in $O(\sqrt{T})$ steps.

It is an open problem to extend Szegedy's algorithm so that it not only detects the existence of an element with the desired property but also finds it in $O(\sqrt{T})$ time steps. (This is known as the "finding problem".) A step in this direction was made by Magniez et al. [11] who generalized Tulsi's algorithm for search on the grid [16] to solve the finding problem in $O(\sqrt{T})$ steps whenever the classical random walk is vertex transitive and the search space has a unique element with the desired property.

Quantum algorithms for spatial search are also useful for designing quantum communication protocols for the set disjointness problem. In the set disjointness problem, one has two parties holding inputs $x \in \{0,1\}^N$ and $y \in \{0,1\}^N$ and they have to determine if there is $i \in \{1, \ldots, N\}$ for which $x_i = y_i = 1$. (One can think of x and y as representing subsets $X, Y \subseteq \{1, \ldots, N\}$ with $x_i = 1 (y_i = 1)$ if $i \in X (i \in Y)$. Then, determining if $x_i = y_i = 1$ for some i is equivalent to determining if $X \cap Y \neq \emptyset$.)

The goal is to solve the problem, communicating as few bits between the two parties as possible. Classically, $\Omega(N)$ bits of communication are required [13]. The optimal quantum protocol [1] uses $O(\sqrt{N})$ quantum bits of communication and its main idea is to reduce the problem to spatial search. As shown by the $\Omega(\sqrt{N})$ lower bound of [14], this algorithm is optimal.

Cross-References

- Quantum Algorithm for Element Distinctness
- Quantum Analogues of Markov Chains
- Quantum Search

Recommended Reading

1. Aaronson S, Ambainis A (2003) Quantum search of spatial regions. In: Proceedings of the 44th annual IEEE symposium on foundations of computer science (FOCS), Cambridge, pp 200–209
2. Ambainis A (2003) Quantum walks and their algorithmic applications. Int J Quantum Inf 1:507–518
3. Ambainis A, Kempe J, Rivosh A (2005) Coins make quantum walks faster. In: Proceedings of SODA'05, Vancouver, pp 1099–1108
4. Ambainis A, Backurs A, Nahimovs N, Ozols R, Rivosh A (2012) Search by quantum walks on two-dimensional grid without amplitude amplification. In: Proceedings of TQC'2012, Tokyo, pp 87–97

5. Benioff P (2002) Space searches with a quantum robot. In: Quantum computation and information, Washington, DC, 2000. Contemporary mathematics, vol 305. American Mathematical Society, Providence, pp 1–12
6. Brassard G, Høyer P, Mosca M, Tapp A (2002) Quantum amplitude amplification and estimation. In: Quantum computation and information, Washington, DC, 2000. Contemporary mathematics, vol 305. American Mathematical Society, Providence, pp 53–74
7. Childs AM, Goldstone J (2004) Spatial search by quantum walk. Phys Rev A 70:022314
8. Childs AM, Goldstone J (2004) Spatial search and the Dirac equation. Phys Rev A 70:042312
9. Grover L (1996) A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th STOC, Philadelphia. ACM, New York, pp 212–219
10. Kempe J (2003) Quantum random walks – an introductory overview. Contemp Phys 44(4):302–327
11. Magniez F, Nayak A, Richter P, Santha M (2012) On the hitting times of quantum versus random walks. Algorithmica 63(1–2):91–116
12. Marquezino F, Portugal R, Boettcher S (2011) Quantum search algorithms on hierarchical networks. In: IEEE information theory workshop (ITW), Paraty, pp 247–251
13. Razborov A (1992) On the distributional complexity of disjointness. Theor Comput Sci 106(2):385–390
14. Razborov AA (2002) Quantum communication complexity of symmetric predicates. Izv Russ Acad Sci Math 67:145–159
15. Szegedy M (2004) Quantum speed-up of Markov Chain based algorithms. In: Proceedings of FOCS'04, Rome, pp 32–41
16. Tulsi A (2008) Faster quantum-walk algorithm for the two-dimensional spatial search. Phys Rev A 78:012310

Quantum Algorithm for Solving Pell's Equation

Sean Hallgren
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, State College, PA, USA

Years and Authors of Summarized Original Work

2002; Hallgren

Problem Definition

Pell's equation is one of the oldest studied problem in number theory. For a positive square-free integer d, Pell's equation is $x^2 - dy^2 = 1$, and the problem is to compute integer solutions x, y of the equation [8, 10]. The earliest algorithm for it uses the continued fraction expansion of $\sqrt{d}$ and dates back to 1000 a.d. by Indian mathematicians. Lagrange showed that there are an infinite number of solutions of Pell's equation. All solutions are of the form $x_n + y_n\sqrt{d} = (x_1 + y_1\sqrt{d})^n$, where the smallest solution, (x_1, y_1), is called the fundamental solution. The solution (x_1, y_1) may have exponentially many bits in general in terms of the input size, which is $\log d$, and so cannot be written down in polynomial time. To resolve this difficulty, the computational problem is recast as computing the integer closest to the regulator $R = \ln(x_1 + y_1\sqrt{d})$. In this representation, solutions of Pell's equation are positive integer multiples of R.

Solving Pell's equation is a special case of computing the unit group of number field. For a positive non-square integer Δ congruent to 0 or 1 mod 4, $K = \mathbb{Q}(\sqrt{\Delta})$ is a real quadratic number field. Its subring $\mathcal{O} = \mathbb{Z}\left[\frac{\Delta+\sqrt{\Delta}}{2}\right] \subseteq Q(\sqrt{\Delta})$ is called the quadratic order of discriminant Δ. The unit group is the set of invertible elements of $\mathcal{O}$. Units have the form $\pm\varepsilon^k$, where $k \in \mathbb{Z}$, for some $\varepsilon > 1$ called the fundamental unit. The fundamental unit ε can have exponentially many bits, so an approximation of the regulator $R = \ln\varepsilon$ is computed. In this representation the unit group consists of integer multiples of R. Given the integer closest to R there are classical polynomial-time algorithms to compute R to any precision. There are also efficient algorithms to test if a given number is a good approximation to an integer multiple of a unit or to compute the least significant digits of $\varepsilon = e^R$ [1, 3].

Two related and potentially more difficult problems are the principal ideal problem and computing the class group of a number field. In the principal ideal problem, a number field and an ideal I of $\mathcal{O}$ are given, and the problem is to decide if the ideal is principal, i.e., whether

there exists α such that $I = \alpha\mathcal{O}$. If it is principal, then one can ask for an approximation of $\ln\alpha$. There are efficient classical algorithms to verify that a number is close to $\ln\alpha$ [1, 3]. The class group of a number field is the finite abelian group defined by taking the set of fractional ideals modulo the principal fractional ideals. The class number is the size of the class group. Computing the unit group, computing the class group, and solving the principal ideal problems are three of the main problems of computational algebraic number theory [3]. Assuming the GRH, they are in NP ∩ CoNP [9].

Key Results

The best known classical algorithms for the problems defined in the last section take subexponential time, but there are polynomial-time quantum algorithms for them [5, 7].

Theorem 1 *Given a quadratic discriminant Δ, there is a classical algorithm that computes an integer multiple of the regulator to within one. Assuming the GRH, this algorithm computes the regulator to within one and runs in expected time* $\exp\left(\sqrt{(\log\Delta)\log\log\Delta}\right)^{O(1)}$.

Theorem 2 *There is a polynomial-time quantum algorithm that, given a quadratic discriminant Δ, approximates the regulator to within δ of the associated order $\mathcal{O}$ in time polynomial in* $\log\Delta$ *and* $\log\delta$ *with probability exponentially close to one.*

Corollary 1 *There is a polynomial-time quantum algorithm that solves Pell's equation.*

The quantum algorithm for Pell's equation uses the existence of a periodic function on the reals which has period R and is one-to-one within each period [5, 7]. There is a discrete version of this function that can be computed efficiently. This function does not have the same periodic property since it cannot be evaluated at arbitrary real numbers such as R, but it does approximate the situation well enough for the quantum algorithm. In particular, computing the approximate period of this function gives R to the closest integer or, in other words, computes a generator for the unit group.

Theorem 3 *There is a polynomial-time quantum algorithm that solves the principal ideal problem in real quadratic number fields.*

Corollary 2 *There is a polynomial-time quantum algorithm that can break the Buchmann-Williams key-exchange protocol in real quadratic number fields.*

Theorem 4 *The class group and class number of a real quadratic number field can be computed in quantum polynomial time assuming the GRH.*

In general, one can ask to find the unit group of an arbitrary degree number field $\mathbb{Q}(\theta)$, where θ is the root of a polynomial with rational coefficients. There are two parameters associated with this problem. The first is the discriminant, which generalizes parameter above. The second is the degree n of the number field as a vector space over the rational numbers. In the above example the degree is fixed at 2. The unit group of an arbitrary degree number can also be computed efficiently by a quantum algorithm.

Theorem 5 ([4]) *The unit group of a number field can be computed by a quantum algorithm in time polynomial in log the discriminant, and the degree n.*

This last result uses a major generalization of the hidden subgroup problem to continuous functions. A new method is used to compute the function that is polynomial time in the degree of the number field and solves the hidden subgroup problem for continuous groups.

Applications

Computationally hard number theoretic problems are useful for public key cryptosystems. There are reductions from factoring to Pell's equation and Pell's equation to the principal ideal problem, but no reductions are known in the opposite direction. The principal ideal problem forms the basis of the Buchmann-Williams key-exchange protocol

[2]. Identification schemes based on this problem have been proposed by Hamdy and Maurer [6]. The classical exponential-time algorithms help determine which parameters to choose for the cryptosystem. The best known algorithm for Pell's equation is exponentially slower than the best factoring algorithm. Systems based on these harder problems were proposed as alternatives in case factoring turns out to be polynomial time solvable. The efficient quantum algorithms can break these cryptosystems.

Open Problems

Lattice-based cryptography is the leading class of candidates for primitives secure against quantum computers. Recent systems have used lattices from number fields in order to make them more efficient. It is an open question whether lattice-based systems are secure against quantum computers, given that quantum computers have an exponential advantage over classical computers for some problems in number fields.

Cross-References

- ▸ Quantum Algorithm for Factoring
- ▸ Quantum Algorithms for Class Group of a Number Field

Recommended Reading

1. Buchmann J, Thiel C, Williams HC (1995) Short representation of quadratic integers. In: Bosma W, van der Poorten AJ (eds) Computational algebra and number theory, Sydney 1992. Mathematics and its applications, vol 325. Kluwer Academic, Dordrecht, pp 159–185
2. Buchmann JA, Williams HC (1989) A key exchange system based on real quadratic fields (extended abstract). In: Brassard G (ed) Advances in cryptology-CRYPTO '89, 20–24 Aug 1990. Lecture notes in computer science, vol 435. Springer, Berlin, pp 335–343
3. Cohen H (1993) A course in computational algebraic number theory. Graduate texts in mathematics, vol 138. Springer, Berlin/Heidelberg
4. Eisentraeger K, Hallgren S, Kitaev A, Song F (2014) A quantum algorithm for computing the unit group of an arbitrary degree number field. In: Proceedings of the 46th ACM symposium on theory of computing
5. Hallgren S (2007) Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. J ACM 54(1):1–19
6. Hamdy S, Maurer M (1999) Feige-fiat-shamir identification based on real quadratic fields. Techincal report TI-23/99. Technische Universität Darmstadt, Fachbereich Informatik. http://www.informatik.tu-darmstadt.de/TI/Veroeffentlichung/TR/
7. Jozsa R (2003) Notes on Hallgren's efficient quantum algorithm for solving Pell's equation. Techincal report, quant-ph/0302134
8. Lenstra HW Jr (2002) Solving the Pell equation. Not Am Math Soc 49:182–192
9. Thiel C (1995) On the complexity of some problems in algorithmic algebraic number theory. Ph.D. thesis, Universität des Saarlandes, Saarbrücken
10. Williams HC (2002) Solving the Pell equation. In: Proceedings of the millennial conference on number theory, pp 397–435

Quantum Algorithm for the Collision Problem

Gilles Brassard[1], Peter Høyer[2], and Alain Tapp[1]
[1]Université de Montréal, Montréal, QC, Canada
[2]University of Calgary, Calgary, AB, Canada

Keywords

Collision; Grover's algorithm; Quantum algorithm

Years and Authors of Summarized Original Work

1998; Brassard, Høyer, Tapp

Problem Definition

A function F is said to be *r-to-one* if every element in its image has exactly r distinct preimages.

Input : an r-to-one function F.
Output : x_1 and x_2 such that $F(x_1) = F(x_2)$.

Key Results

The algorithm presented here finds collisions in arbitrary r-to-one functions F after only $O(\sqrt[3]{N/r})$ expected evaluations of F. The algorithm uses the function as a black box, that is, the only thing the algorithm requires is the capacity to evaluate the function. Again assuming the function is given by a black box, the algorithm is optimal [1], and it is more efficient than the best *possible* classical algorithm which has query complexity $\Omega(\sqrt{N/r})$. The result is stated precisely in the following theorem and corollary.

Theorem 1 *Given an r-to-one function $F:X\to Y$ with $r\geq 2$ and an integer $1\leq k\leq N=|X|$, algorithm* **Collision**(F,k) *returns a collision after an expected number of $O(k+\sqrt{N/(rk)})$ evaluations of F and uses space $\Theta(k)$. In particular, when $k=\sqrt[3]{N/r}$, then* **Collision**(F,k) *uses an expected number of $O(\sqrt[3]{N/r})$ evaluations of F and space $\Theta(\sqrt[3]{N/r})$.*

Corollary 1 *There exists a quantum algorithm that can find a collision in an arbitrary r-to-one function $F:X\to Y$, for any $r\geq 2$, using space S and an expected number of $O(T)$ evaluations of F for every $1\leq S\leq T$ subject to $ST^2\geq |F(X)|$ where $F(X)$ denotes the image of F.*

The algorithm uses as a procedure a version of Grover's search algorithm. Given a function H with domain size n and a target y, **Grover**(H,y) returns an x such that $H(x)=y$ in expected $O(\sqrt{n})$ evaluations of H.

Collision(F,k):

1. Pick an arbitrary subset $K\subseteq X$ of cardinality k. Construct a table L of size k where each item in L holds a distinct pair $(x,F(x))$ with $x\in K$.
2. Sort L according to the second entry in each item of L.
3. Check if L contains a collision, that is, check if there exist distinct elements $(x_0,F(x_0))$, $(x_1,F(x_1))\in L$ for which $F(x_0)=F(x_1)$. If so, go to step 6.
4. Compute $x_1=$ **Grover**$(H,1)$ where $H:X\to\{0,1\}$ denotes the function defined by $H(x)=1$ if and only if there exists $x_0\in K$ so that $(x_0,F(x))\in L$ but $x\neq x_0$. (Note that x_0 is unique if it exists since we already checked that there are no collisions in L.)
5. Find $(x_0,F(x_1))\in L$.
6. Output the collision $\{x_0,x_1\}$.

Applications

This problem is of particular interest for cryptology because some functions known as *hash functions* are used in various cryptographic protocols. The security of these protocols crucially depends on the presumed difficulty of finding collisions in such functions.

Recommended Reading

1. Aaronson S, Shi Y (2004) Quantum lower bounds for the collision and the element distinctness problems. J ACM (JACM) 51(4):595–605
2. Brassard G, Høyer P, Tapp A (1998) Quantum algorithm for the collision problem. In: 3rd Latin American theoretical informatics symposium (LATIN'98). LNCS, vol 1380. Springer, Berlin, pp 163–169
3. Brassard G, Høyer P, Mosca M, Tapp A (2002) Quantum amplitude amplification and estimation. In: Lomonaco SJ (ed) Quantum computation & quantum information science. AMS contemporary mathematics series millennium volume, vol 305, pp 53–74
4. Boyer M, Brassard G, Høyer P, Tapp A (1996) Tight bounds on quantum searching. In: Proceedings of the fourth workshop on physics of computation, Boston, pp 36–43
5. Carter JL, Wegman MN (1979) Universal classes of hash functions. J Comput Syst Sci 18(2):143–154
6. Grover LK (1996) A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th annual ACM symposium on theory of computing, Philadelphia, pp 212–219

7. Nielsen MA, Chuang IL (2000) Quantum computation and quantum information. Cambridge University Press, Cambridge
8. Stinson DR (1995) Cryptography: theory and practice. CRC, Boca Raton

Quantum Algorithm for the Discrete Logarithm Problem

Pranab Sen
School of Technology and Computer Science,
Tata Institute of Fundamental Research,
Mumbai, India

Keywords

Logarithms in groups

Years and Authors of Summarized Original Work

1994; Shor

Problem Definition

Given positive real numbers $a \neq 1, b$, the logarithm of b to base a is the unique real number s such that $b = a^s$. The notion of the *discrete logarithm* is an extension of this concept to general groups.

Problem 1 (Discrete logarithm)

INPUT: Group $G, a, b \in G$ such that $b = a^s$ for some positive integer s.

OUTPUT: The smallest positive integer s satisfying $b = a^s$, also known as the *discrete logarithm* of b to the base a in G.

The usual logarithm corresponds to the discrete logarithm problem over the group of positive reals under multiplication. The most common case of the discrete logarithm problem is when the group $G = \mathbb{Z}_p^*$, the multiplicative group of integers between 1 and $p-1$ modulo p, where p is a prime. Another important case is when the group G is the group of points of an elliptic curve over a finite field.

Key Results

The discrete logarithm problem in $\mathbb{Z}_p^*$, where p is a prime, as well as in the group of points of an elliptic curve over a finite field is believed to be intractable for randomized classical computers. That is, any, possibly randomized, algorithm for the problem running on a classical computer will take time that is superpolynomial in the number of bits required to describe an input to the problem. The best classical algorithm for finding discrete logarithms in $\mathbb{Z}_p^*$, where p is a prime, is Gordon's [4] adaptation of the number field sieve which runs in time $\exp(O((\log p)^{1/3}(\log\log p)^{2/3}))$.

In a breakthrough result, Shor [9] gave an efficient quantum algorithm for the discrete logarithm problem in any group G; his algorithm runs in time that is polynomial in the bit size of the input.

Result 1 ([9]) There is a quantum algorithm solving the discrete logarithm problem in any group G on n-bit inputs in time $O(n^3)$ with probability at least 3/4.

Description of the Discrete Logarithm Algorithm

Shor's algorithm [9] for the discrete logarithm problem makes essential use of an efficient quantum procedure for implementing a unitary transformation known as the *quantum Fourier transform*. His original algorithm gave an efficient procedure for performing the quantum Fourier transform only over groups of the form $\mathbb{Z}_r$, where r is a "smooth" integer, but nevertheless, he showed that this itself sufficed to solve the discrete logarithm in the general case. In this article, however, a more modern description of Shor's algorithm is given. In particular, a result by Hales and Hallgren [5] is used which shows that the quantum Fourier transform over any finite cyclic group $\mathbb{Z}_r$ can be

efficiently approximated to inverse-exponential precision.

A description of the algorithm is given below. A general familiarity with quantum notation on the part of the reader is assumed. A good introduction to quantum computing can be found in the book by Nielsen and Chuang [8]. Let $(G, a, b, \bar{r})$ be an instance of the discrete logarithm problem, where $\bar{r}$ is a supplied upper bound on the order of a in G. That is, there exists a positive integer $r \leq \bar{r}$ such that $a^r = 1$. By using an efficient quantum algorithm for order finding also discovered by Shor [9], one can assume that the order of a in G is known, that is, the smallest positive integer r satisfying $a^r = 1$. Shor's order-finding algorithm runs in time $O((\log \bar{r})^3)$. Let $\epsilon > 0$. The discrete logarithm algorithm works on three registers, of which the first two are each t qubits long, where $t := O(\log r + \log(1/\epsilon))$, and the third register is big enough to store an element of G. Let U denote the unitary transformation

$$U : |x\rangle|y\rangle|z\rangle \mapsto |x\rangle|y\rangle|z \otimes (b^x a^y)\rangle,$$

where $\otimes$ denotes bitwise XOR. Given access to a reversible oracle for group operations in G, U can be implemented reversibly in time $O(t^3)$ by repeated squaring.

Let $\mathbb{C}[\mathbb{Z}_r]$ denote the Hilbert space of functions from $\mathbb{Z}_r$ to complex numbers. The computational basis of $\mathbb{C}[\mathbb{Z}_r]$ consists of the delta functions $\{|l\rangle\}_{0 \leq l \leq r-1}$, where $|\rangle$ is the function that sends the element l to 1 and the other elements of $\mathbb{Z}_r$ to 0. Let $\mathrm{QFT}_{\mathbb{Z}_r}$ denote the *quantum Fourier transform* over the cyclic group $\mathbb{Z}_r$ defined as the following unitary operator on $\mathbb{C}[\mathbb{Z}_r]$:

$$\mathrm{QFT}_{\mathbb{Z}_r} : |x\rangle \mapsto r^{-1/2} \sum_{y \in \mathbb{Z}_r} e^{-2\pi i x y / r} |y\rangle.$$

It can be implemented in quantum time $O(t \log(t/\epsilon) + \log^2(1/\epsilon))$ up to an error of ϵ using one t-qubit register [5]. Note that for any $k \in \mathbb{Z}_r$, $\mathrm{QFT}_{\mathbb{Z}_r}$ transforms the state $r^{-1/2} \sum_{x \in \mathbb{Z}_r} e^{-2\pi i k x / r} |x\rangle$ to the state $|k\rangle$. For any integer $l, 0 \leq l \leq r-1$, define

$$|\hat{l}\rangle := r^{-1/2} \sum_{k=0}^{r-1} e^{-2\pi i l k / r} |a^k\rangle. \qquad (1)$$

Observe that $\{|\hat{l}\rangle\}_{0 \leq l \leq r-1}$ forms an orthonormal basis of $\mathbb{C}[\langle a \rangle]$, where $\langle a \rangle$ is the subgroup generated by a in G and is isomorphic to $\mathbb{Z}_r$, and $\mathbb{C}[\langle a \rangle]$ denotes the Hilbert space of functions from $\langle a \rangle$ to complex numbers.

Algorithm 1 (Discrete logarithm)

INPUT: Elements $a, b \in G$, a quantum circuit for U, the order r of a in G.

OUTPUT: With constant probability, the discrete logarithm s of b to the base a in G.

RUNTIME: A total of $O(t^3)$ basic gate operations, including four invocations of $\mathrm{QFT}_{\mathbb{Z}_r}$ and one of U.

PROCEDURE:

1. Repeat Steps (a)–(e) twice, obtaining $(sl_1 \bmod r, l_1)$ and $(sl_2 \bmod r, l_2)$.
   (a) $|0\rangle|0\rangle|0\rangle$
   (b) $\mapsto r^{-1} \sum_{x,y \in \mathbb{Z}_r} |x\rangle|y\rangle|0\rangle$
   Apply $\mathrm{QFT}_{\mathbb{Z}_r}$ to the first two registers:
   (c) $\mapsto r^{-1} \sum_{x,y \in \mathbb{Z}_r} |x\rangle|y\rangle|b^x a^y\rangle$
   Apply U
   (d) $\mapsto r^{-1/2} \sum_{l=0}^{r-1} |sl \mod r\rangle|l\rangle|\hat{l}\rangle$
   Apply $\mathrm{QFT}_{\mathbb{Z}_r}$ to the first two registers:
   (e) $\mapsto (sl \mod r, l)$
   Measure the first two registers:
2. If l_1 is not coprime to l_2, abort.
3. Let k_1, k_2 be integers such that $k_1 l_1 + k_2 l_2 = 1$. Then, output $s = k_1(sl_1) + k_2(sl_2) \bmod r$.

The working of the algorithm is explained below. From Eq. (1), it is easy to see that

$$|b^x a^y\rangle = r^{-1/2} \sum_{l=0}^{r-1} e^{2\pi i l(sx+y)/r} |\hat{l}\rangle.$$

Thus, the state in Step 1(c) of the above algorithm can be written as

$$
\begin{aligned}
& r^{-1} \sum_{x,y \in \mathbb{Z}_r} |x\rangle|y\rangle|b^x a^y\rangle \\
& \quad = r^{-3/2} \sum_{l=0}^{r-1} \sum_{x,y \in \mathbb{Z}_r} \mathrm{e}^{2\pi i l(sx+y)/r} |x\rangle|y\rangle|\hat{l}\rangle \\
& \quad = r^{-3/2} \sum_{l=0}^{r-1} \left[\sum_{x \in \mathbb{Z}_r} e^{2\pi i s l x/r} |x\rangle \right] \cdot \left[\sum_{y \in \mathbb{Z}_r} e^{2\pi i l y/r} |y\rangle \right] |\hat{l}\rangle .
\end{aligned}
$$

Now, applying $\mathrm{QFT}_{\mathbb{Z}_r}$ to the first two registers gives the state in Step 1(d) of the above algorithm. Measuring the first two registers gives ($sl \bmod r, l$) for a uniformly distributed $l, 0 \le l \le r-1$ in Step 1(e). By elementary number theory, it can be shown that if integers l_1, l_2 are uniformly and independently chosen between 0 and $l-1$, they will be coprime with constant probability. In that case, there will be integers k_1, k_2 such that $k_1 l_1 + k_2 l_2 = 1$, leading to the discovery of the discrete logarithm s in Step 3 of the algorithm with constant probability. Since actually only an ϵ-approximate version of $\mathrm{QFT}_{\mathbb{Z}_r}$ can be applied, ϵ can be set to be a sufficiently small constant, and this will still give the correct discrete logarithm s in Step 3 of the algorithm with constant probability. The success probability of Shor's algorithm for the discrete logarithm problem can be boosted to at least 3/4 by repeating it a constant number of times.

Generalizations of the Discrete Logarithm Algorithm

The discrete logarithm problem is a special case of a more general problem called the *hidden subgroup problem* [8]. The ideas behind Shor's algorithm for the discrete logarithm problem can be generalized in order to yield an efficient quantum algorithm for hidden subgroups in Abelian groups (see [1] for a brief sketch). It turns out that finding the discrete logarithm of b to the base a in G reduces to the hidden subgroup problem in the group $\mathbb{Z}_r \times \mathbb{Z}_r$ where r is the order of a in G. Besides the discrete logarithm problem, other cryptographically important functions like integer factoring, finding the order of permutations, as well as finding self-shift-equivalent polynomials over finite fields can be reduced to instances of a hidden subgroup in Abelian groups.

Applications

The assumed intractability of the discrete logarithm problem lies at the heart of several cryptographic algorithms and protocols. The first example of public-key cryptography, namely, the Diffie-Hellman key exchange [2], uses discrete logarithms, usually in the group $\mathbb{Z}_p^*$ for a prime p. The security of the US national standard Digital Signature Algorithm (see [7] for details and more references) depends on the assumed intractability of discrete logarithms in $\mathbb{Z}_p^*$, where p is a prime. The ElGamal public-key cryptosystem [3] and its derivatives use discrete logarithms in appropriately chosen subgroups of $\mathbb{Z}_p^*$, where p is a prime. More recent applications include those in elliptic curve cryptography [6], where the group consists of the group of points of an elliptic curve over a finite field.

Cross-References

- Abelian Hidden Subgroup Problem
- Quantum Algorithm for Factoring

Recommended Reading

1. Brassard G, Høyer P (1997) An exact quantum polynomial-time algorithm for Simon's problem. In: Proceedings of the 5th Israeli symposium on theory of computing and systems, Ramat-Gan, 17–19 June 1997, pp 12–23

2. Diffie W, Hellman M (1976) New directions in cryptography. IEEE Trans Inf Theor 22:644–654
3. ElGamal T (1985) A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans Inf Theor 31(4):469–472
4. Gordon D (1993) Discrete logarithms in GF(p) using the number field sieve. SIAM J Discret Math 6(1):124–139
5. Hales L, Hallgren S (2000) An improved quantum Fourier transform algorithm and applications. In: Proceedings of the 41st annual IEEE symposium on foundations of computer science, Redondo Beach, pp 515–525
6. Hankerson D, Menezes A, Vanstone S (2004) Guide to elliptic curve cryptography. Springer, New York
7. Menezes A, van Oorschot P, Vanstone S (1997) Handbook of applied cryptography. CRC Press, Boca Raton
8. Nielsen M, Chuang I (2000) Quantum computation and quantum information. Cambridge University Press, Cambridge
9. Shor P (1997) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J Comput 26(5):1484–1509

Quantum Algorithm for the Parity Problem

Yaoyun Shi
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

Keywords

Deutsch algorithm; Deutsch-Jozsa algorithm; Parity

Years and Authors of Summarized Original Work

1985; Deutsch

Problem Definition

The *parity* of n bits $x_0, x_1, \cdots, x_{n-1} \in \{0,1\}$ is

$$x_0 \oplus x_1 \oplus \cdots \oplus x_{n-1} = \sum_{i=0}^{n-1} x_i \mod 2.$$

As an elementary Boolean function, parity is important not only as a building block of digital logic but also for its instrumental roles in several areas such as error correction, hashing, discrete Fourier analysis, pseudorandomness, communication complexity, and circuit complexity. The feature of parity that underlies its many applications is its maximum sensitivity to the input: flipping any bit in the input changes the output. The computation of parity from its input bits is quite straightforward in most computation models. However, two settings deserve attention.

The first is the circuit complexity of parity when the gates are restricted to AND, OR, and NOT gates. It is known that parity cannot be computed by such a circuit of a polynomial size and a constant depth, a groundbreaking result proved independently by Furst, Saxe, and Sipser [7] and Ajtai [1] and improved by several subsequent works.

The second, and the focus of this article, is in the decision tree model (also called the query model or the black-box model), where the input bits $x = x_0 x_1 \cdots x_{n-1} \in \{0,1\}^n$ are known to an oracle only, and the algorithm needs to ask questions of the type "x_i =?" to access the input. The complexity is measured by the number of queries. Specifically, a quantum query is the application of the following query gate:

$$O_x : |i, b\rangle \mapsto |i, b \oplus x_i\rangle,$$
$$i \in \{0, \cdots, n-1\}, b \in \{0,1\}.$$

Key Results

Proposition 1 *There is a quantum query algorithm computing the parity of 2 bits with probability 1 using 1 query.*

Proof Denote by $|\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$. The initial state of the algorithm is

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |-\rangle.$$

Apply a query gate, using the first register for the index slot and the second register for the answer slot. The resulting state is

$$\frac{1}{\sqrt{2}}((-1)^{x_0}|0\rangle + (-1)^{x_1}|1\rangle) \otimes |-\rangle.$$

Applying a Hadamard gate $H = |+\rangle\langle 0| + |-\rangle\langle 1|$ on the first register brings the state to

$$(-1)^{x_0}|x_0 + x_1\rangle \otimes |-\rangle.$$

Thus measuring the first register gives $x_0 + x_1$ with certainty.

Corollary 1 *There is a quantum query algorithm computing the parity of n bits with probability 1 using $\lceil n/2 \rceil$ queries.*

The above quantum upper bound for parity is tight, even if the algorithm is allowed to err with a probability bounded away from 1/2 [6]. In contrast, any classical randomized algorithm with bounded error probability requires n queries. This follows from the fact that on a random input, any classical algorithm not knowing all the input bits is correct with precisely 1/2 probability.

Applications

The quantum speedup for computing parity was first observed by Deutsch [4]. His algorithm uses $|0\rangle$ in the answer slot, instead of $|-\rangle$. After one query, the algorithm has 3/4 chance of computing the parity, better than any classical algorithm (1/2 chance). The presented algorithm is actually a special case of the Deutsch-Jozsa Algorithm, which solves the following problem now referred to as the Deutsch-Jozsa Problem.

Problem 1 (Deutsch-Jozsa Problem) Let $n \geq 1$ be an integer. Given an oracle function $f : \{0,1\}^n \to \{0,1\}$ that satisfies either (a) $f(x)$ is constant on all $x \in \{0,1\}^n$ or (b) $|\{x : f(x) = 1\}| = |\{x : f(x) = 0\}| = 2^{n-1}$, determine which case it is.

When $n = 1$, the above problem is precisely parity of 2 bits. For a general n, the Deutsch-Jozsa Algorithm solves the problem using only once the following query gate:

$$O_f : |x, b\rangle \mapsto |x, f(x) \oplus b\rangle,\ x \in \{0,1\}^n,\ b \in \{0,1\}.$$

The algorithm starts with

$$|0^n\rangle \otimes |-\rangle.$$

It applies $H^{\otimes n}$ on the index register (the first n qubits), changing the state to

$$\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes |-\rangle.$$

The oracle gate is then applied, resulting in

$$\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)}|x\rangle \otimes |-\rangle.$$

For the second time, $H^{\otimes n}$ is applied on the index register, bringing the state to

$$\sum_{y \in \{0,1\}^n} \left(\frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x)+x\cdot y} \right) |y\rangle \otimes |-\rangle. \tag{1}$$

Finally, the index register is measured in the computational basis. The Algorithm returns "Case (a)" if 0^n is observed, otherwise returns "Case (b)."

By direct inspection, the amplitude of $|0^n\rangle$ is 1 in Case (a) and 0 in Case (b). Thus the algorithm is correct with probability 1. It is easy to see that any deterministic algorithm requires $n/2 + 1$ queries in the worst case; thus the algorithm provides the first exponential quantum versus deterministic speedup.

Note that $O(1)$ expected a number of queries are sufficient for randomized algorithms to solve the Deutsch-Jozsa Problem with a constant success probability arbitrarily close to 1. Thus the Deutsch-Jozsa Algorithm does not have much advantage compared with error-bounded random-

ized algorithms. One might also feel that the saving of one query for computing the parity of 2 bits by Deutsch-Jozsa Algorithm is due to the artificial definition of one quantum query. Thus the significance of the Deutsch-Jozsa Algorithm is not in solving a practical problem, but in its pioneering use of quantum Fourier transform (QFT), of which $H^{\otimes n}$ is one, in the pattern

$$\text{QFT} \rightarrow \text{Query} \rightarrow \text{QFT}.$$

The same pattern appears in many subsequent quantum algorithms, including those found by Bernstein and Vazirani [2], Simon [9], and Shor [8].

The Deutsch-Jozsa Algorithm is also referred to as Deutsch Algorithm. The algorithm as presented above is actually the result of the improvement by Cleve, Ekert, Macchiavello, and Mosca [3] and independently by Tapp (unpublished) on the algorithm in [5].

Cross-References

▸ Greedy Set-Cover Algorithms

Recommended Reading

1. Ajtai M (1983) $\sum_1^1$-formulae on finite structures. Ann Pure Appl Log 24(1):1–48
2. Bernstein E, Vazirani U (1997) Quantum complexity theory. SIAM J Comput 26(5):1411–1473
3. Cleve R, Ekert A, Macchiavello C, Mosca M (1998) Quantum algorithms revisited. Proc R Soc Lond A454:339–354
4. Deutsch D (1985) Quantum theory, the Church-Turing principle and the universal quantum computer. Proc R Soc Lond A400:97–117
5. Deutsch D, Jozsa R (1992) Rapid solution of problems by quantum computation. Proc R Soc Lond A439:553–558
6. Farhi E, Goldstone J, Gutmann S, Sipser M (1998) A limit on the speed of quantum computation in determining parity. Phys Rev Lett 81:5442–5444
7. Furst M, Saxe J, Sipser M (1984) Parity, circuits, and the polynomial time hierarchy. Math Syst Theor 17(1):13–27
8. Shor PW (1997) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J Comput 26(5):1484–1509
9. Simon DR (1997) On the power of quantum computation. SIAM J Comput 26(5):1474–1483

Quantum Algorithms for Class Group of a Number Field

Sean Hallgren
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, State College, PA, USA

Years and Authors of Summarized Original Work

2005; Hallgren

Problem Definition

Associated with each number field is a finite abelian group called the class group. The order of the class group is called the class number. Computing the class number and the structure of the class group of a number field is among the main tasks in computational algebraic number theory [4].

A number field F can be defined as a subfield of the complex numbers $\mathbb{C}$ which is generated over the rational numbers $\mathbb{Q}$ by an algebraic number, i.e., $F = \mathbb{Q}(\theta)$ where θ is the root of a polynomial with rational coefficients. The ring of integers $\mathcal{O}$ of F is the subset consisting of all elements that are roots of monic polynomials with integer coefficients. The ring $\mathcal{O} \subseteq F$ can be thought of as a generalization of $\mathbb{Z}$, the ring of integers in $\mathbb{Q}$. In particular, one can ask whether $\mathcal{O}$ is a principal ideal domain and whether elements in $\mathcal{O}$ have unique factorization. Another interesting problem is computing the unit group $\mathcal{O}^*$, which is the set of invertible algebraic integers inside F, that is, elements $\alpha \in \mathcal{O}$ such that α^{-1} is also in $\mathcal{O}$.

Ever since the class group was discovered by Gauss in 1798, it has been an interesting object of study. The class group of F is the set of equivalence classes of fractional ideals of F, where two ideals I and J are equivalent if there exists $\alpha \in F^*$ such that $J = \alpha I$. Multiplication of two ideals I and J is defined as the ideal generated by all products ab, where $a \in I$ and $b \in J$. Much is still unknown about number fields, such as whether there exist infinitely many number fields with trivial class group. The question of the class group being trivial is equivalent to asking whether the elements in the ring of integers $\mathcal{O}$ of the number field have unique factorization.

In addition to computing the class number and the structure of the class group, computing the unit group and determining whether given ideals are principal, called the principal ideal problem, are also central problems in computational algebraic number theory.

Key Results

The best known classical algorithms for the class group take subexponential time [1, 2, 4]. Assuming the GRH, computing the class group, the unit group, and solving the principal ideal problem are in NP ∩ CoNP [10].

The following theorems state that the three problems defined above have efficient quantum algorithms [7, 9].

Theorem 1 *There is a polynomial-time quantum algorithm that computes the unit group of a constant degree number field.*

Theorem 2 *There is a polynomial-time quantum algorithm that solves the principal ideal problem in constant degree number fields.*

Theorem 3 *The class group and class number of a constant degree number field can be computed in quantum polynomial time assuming the GRH.*

Computing the class group means computing the structure of a finite abelian group given a set of generators for it. When it is possible to efficiently multiply group elements (including computing large powers of elements) and efficiently compute unique representations of each group element, then this problem reduces to the standard hidden subgroup problem over the integers and therefore has an efficient quantum algorithm. Ideal multiplication is efficient in number fields. For imaginary number fields, there are efficient classical algorithms for computing group elements with a unique representation, and therefore there is an efficient quantum algorithm for computing the class group.

For real number fields, there is no known way to efficiently compute unique representations of class group elements. As a result, the classical algorithms typically compute the unit group and class group at the same time. A quantum algorithm [7] is able to efficiently compute the unit group of a number field and then use the principal ideal algorithm to compute a unique quantum representation of each class group element. Then the standard quantum algorithm can be applied to compute the class group structure and class number.

Applications

There are factoring algorithms based on computing the class group of an imaginary number field. One is exponential time and the other is subexponential time [4].

Computationally hard number theoretic problems are useful for public key cryptosystems. Pell's equation reduces to the principal ideal problem, which forms the basis of the Buchmann-Williams key-exchange protocol [3]. Identification schemes have also been based on this problem by Hamdy and Maurer [8]. The classical exponential-time algorithms help determine which parameters to choose for the cryptosystem. Factoring reduces to Pell's equation, and the best known algorithm for it is exponentially slower than the best factoring algorithm. Systems based on these harder problems were proposed as alternatives in case factoring turns out to be polynomial time solvable. The efficient quantum algorithms can break these cryptosystems.

Open Problems

The unit group of an arbitrary degree number field has an efficient quantum algorithm [6], and computing the class group and solving the principal ideal problem are related to this problem. One open problem is to compute certain towers of number fields with special properties, such as an infinite family with constant root discriminant [5].

Cross-References

- Quantum Algorithm for Factoring
- Quantum Algorithm for Solving Pell's Equation

Recommended Reading

1. Biasse JF, Fieker C Subexponential class group and unit group computation in large degree number fields. LMS J Comput Math 17:385–403
2. Buchmann J (1990) A subexponential algorithm for the determination of class groups and regulators of algebraic number fields. In: Goldstein C (ed) Séminaire de Théorie des Nombres, Paris 1988–1989. Progress in mathematics, vol 91. Birkhäuser, Boston, pp 27–41
3. Buchmann JA, Williams HC (1990) A key exchange system based on real quadratic fields (extended abstract). In: Brassard G (ed) Advances in cryptology-CRYPTO '89, 20–24 Aug 1989. Lecture notes in computer science, vol 435. Springer, Berlin, pp 335–343
4. Cohen H (1993) A course in computational algebraic number theory. Graduate texts in mathematics, vol 138. Springer, Berlin/Heidelberg
5. Eisentraeger K, Hallgren, S (2010) Algorithms for ray class groups and Hilbert class fields. In: Proceedings of the 21st ACM-SIAM symposium on discrete algorithms (SODA)
6. Eisentraeger K, Hallgren S, Kitaev A, Song F (2014) A quantum algorithm for computing the unit group of an arbitrary degree number field. In: Proceedings of the 46th ACM symposium on theory of computing
7. Hallgren S (2005) Fast quantum algorithms for computing the unit group and class group of a number field. In: Proceedings of the 37th ACM symposium on theory of computing
8. Hamdy S, Maurer M (1999) Feige-fiat-shamir identification based on real quadratic fields. Techincal report TI-23/99. Technische Universität Darmstadt, Fachbereich Informatik. http://www.informatik.tu-darmstadt.de/TI/Veroeffentlichung/TR/
9. Schmidt A, Vollmer U (2005) Polynomial time quantum algorithm for the computation of the unit group of a number field. In: Proceedings of the 37th ACM symposium on theory of computing
10. Thiel C (1995) On the complexity of some problems in algorithmic algebraic number theory. Ph.D. thesis, Universität des Saarlandes, Saarbrücken

Quantum Algorithms for Graph Connectivity

Aleksandrs Belovs
Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA

Keywords

Finding small subgraphs within large graphs; UPATH; USTCON

Years and Authors of Summarized Original Work

2012; Belovs, Reichardt

Problem Definition

The input is an undirected simple graph G on n vertices. The graph is given by its adjacency matrix: For any two vertices u and v, one can query whether u and v are connected by an edge. (Note that classical algorithms usually have access to G in the form of incidence lists. However, specification of the input graph in the form of adjacency matrix is standard in quantum algorithms.) Two special vertices of the graph, s and t, are selected. The task is to detect whether s and t lie in the same connected component of G. Quantum algorithms for this problem are described.

Classically, this problem can be solved in quadratic time by a variety of algorithms. It is easy to see that this is optimal. Also, the st-connectivity problem is a canonical example of a problem in RL (the class of problems solvable

in randomized logspace) [1]. Later, it was shown to be in L (deterministic logspace) [8].

Key Results

Previous Algorithm

Dürr et al. [5] gave a quantum algorithm with the following properties. Its query complexity is $O(n^{3/2})$, and its time complexity is the same up to logarithmic factors. The algorithm repeatedly executes a quantum subroutine that uses $O(\log n)$ qubits and requires quantum read-only access to $O(n \log n)$ classical bits. This memory is changed between the runs of the quantum subroutine.

The algorithm is based on Borůvka's algorithm [4]. It solves a more general problem of finding a minimum spanning tree of G. In particular, the algorithm outputs a list of the connected components of G.

Main Algorithm

Theorem 1 ([3]) *Consider the st-connectivity problem on an n-vertex graph G with the additional promise: Either s and t lie in different components of G, or they are connected by a path of length at most d. The above problem can be solved by a quantum algorithm in $\tilde{O}(n\sqrt{d})$ time, $O(n\sqrt{d})$ queries, and $O(\log n)$ space.*

Thus, in the worst case of $d = n - 1$, the complexity of the algorithm is the same as of the algorithm by Dürr et al. But if d is small, this algorithm performs better. This promise appears quite naturally in practice.

The algorithm is based on the quantum algorithm for evaluating span programs [6, 7].

Applications

The st-connectivity algorithm or its modifications can be used as a quantum version of dynamic programming. In general, quantum algorithms provide no advantage in implementing dynamic programming. The algorithm of Theorem 1, although it does not have the full power of dynamic programming, attains a quadratic speedup (for small values of d). In [3], this algorithm is combined with the color-coding approach [2] to solve the problem of finding small subgraphs.

For example, consider the problem of detecting the presence of a k-path in an input graph G given by its adjacency matrix. (We assume that $k = O(1)$.) Color each vertex of G in a color from $\{0, 1, \ldots k\}$ independently and uniformly at random. Leave only those edges of G that connect vertices whose colors differ by exactly 1. Add two new vertices s and t, connect s to all vertices of color 0, and connect t to all vertices of color k. Denote the resulting graph by G'.

We say that a k-path in G is colored correctly, if the colors of its vertices go from 0 to k starting with one of its end points. Thus, for any k-path of G, the probability it is colored correctly is $\Omega(1)$.

Execute the algorithm of Theorem 1 on G' with $d = k + 2$. If G contains a correctly colored k-path, then G' has a path of length $k+2$ from s to t; hence, the algorithm accepts. On the other hand, if s and t are connected by a path in G', then G contains a k-path (not necessary correctly colored). Hence, if there is no k-path in G, the algorithm rejects for any coloring of G. By repeating the algorithm constant a number of times with different colorings, it is possible to distinguish these two cases.

Classically, color coding is capable of finding a subgraph H in the input graph, if H is an arbitrary fixed tree. In the quantum case, the class of graphs is narrower.

Problem 1 (Subgraph/not-a-minor promise problem) Let H be a fixed simple graph. The input is a graph G given by its adjacency matrix. The task is to distinguish two cases:

- The graph G contains H as a subgraph.
- The graph G does not contain H as a minor.

Classically, this problem requires $\Omega(n^2)$ queries even if H is a single edge. The quantum query lower bound is $\Omega(n)$.

Theorem 2 ([3]) *Assume that H is a triangle or an edge-subdivision of a star. The subgraph/not-a-minor promise problem for H on an n-vertex input graph can be solved by a quantum algorithm in $\tilde{O}(n)$ time. The algorithm uses $O(n)$ queries. If H is an edge-subdivision of a star, the algorithm uses $O(\log n)$ space.*

Corollary 1 *Assume that H is a path or an edge-subdivision of a claw (a 3-star). There exists a quantum algorithm that detects whether an n-vertex input graph contains H as a subgraph in $\tilde{O}(n)$ time. The algorithm uses $O(n)$ queries and $O(\log n)$ space.*

Cross-References

- Color Coding
- Single-Source Shortest Paths

Recommended Reading

1. Aleliunas R, Karp RM, Lipton RJ, Lovasz L, Rackoff C (1979) Random walks, universal traversal sequences, and the complexity of maze problems. In: Proceedings of 20th IEEE FOCS, San Juan, pp 218–223
2. Alon N, Yuster R, Zwick U (1995) Color-coding. J ACM 42:844–856
3. Belovs A, Reichardt BW (2012) Span programs and quantum algorithms for st-connectivity and claw detection. In: Proceedings of 20th ESA, Ljubljana. LNCS, vol 7501, pp 193–204
4. Borůvka O (1926) O jistém problému minimálním (About a certain minimal problem). Práce mor Přírodověd spol v Brně (Acta Societ Scient Natur Moravicae) 3:37–58 (In Czech)
5. Dürr C, Heiligman M, Høyer P, Mhalla M (2004) Quantum query complexity of some graph problems. In: Proceedings of 31st ICALP, Turku. LNCS, vol 3142, pp 481–493. Springer
6. Reichardt BW (2009) Span programs and quantum query complexity: the general adversary bound is nearly tight for every boolean function. arXiv:0904.2759
7. Reichardt BW (2011) Reflections for quantum query algorithms. In: Proceedings of 22nd ACM-SIAM SODA, San Francisco, pp 560–569
8. Reingold O (2008) Undirected connectivity in logspace. J ACM 55(4):17

Quantum Algorithms for Matrix Multiplication and Product Verification

Robin Kothari[1,2] and Ashwin Nayak[3]
[1]Center for Theoretical Physics, Massachusetts Institute of Technology, Cambridge, MA, USA
[2]David R. Cheriton School of Computer Science, Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada
[3]Department of Combinatorics and Optimization, and Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada

Keywords

Boolean matrix multiplication; Matrix product verification; Quantum algorithms

Years and Authors of Summarized Original Work

2006; Buhrman, Špalek
2012; Jeffery, Kothari, Magniez

Problem Definition

Let S be any algebraic structure over which matrix multiplication is defined, such as a field (e.g., real numbers), a ring (e.g., integers), or a semiring (e.g., the Boolean semiring). If we use $+$ and $\cdot$ to denote the addition and multiplication operations over S, then the matrix product C of two $n \times n$ matrices A and B is defined as $C_{ij} := \sum_{k=1}^{n} A_{ik} \cdot B_{kj}$ for all $i, j \in \{1, 2, \ldots, n\}$. Over the Boolean semiring, the addition and multiplication operations are the logical OR and logical AND operations, respectively, and thus, the matrix product C is defined as $C_{ij} := \bigvee_{k=1}^{n} (A_{ik} \wedge B_{kj})$. In this article we consider the following problems.

Problem 1 (Matrix multiplication)

INPUT: Two $n \times n$ matrices A and B with entries from S.
OUTPUT: The matrix $C := AB$.

Problem 2 (Matrix product verification)

INPUT: Three $n \times n$ matrices A, B, and C with entries from S.
OUTPUT: A bit indicating whether or not $C = AB$.

The matrix multiplication problem is a well-studied problem in classical computer science. The straightforward algorithm for matrix multiplication that computes each entry separately using its definition uses $O(n^3)$ operations. In 1969, Strassen [17] presented an algorithm that multiplies matrices over any ring using only $O(n^{2.807})$ operations, showing that the straightforward approach was suboptimal. Since then there have been many improvements and the complexity of matrix multiplication remains an area of active research.

Surprisingly, the matrix product verification problem can be solved faster. In 1979, Freivalds [6] presented an optimal $O(n^2)$ time bounded-error probabilistic algorithm to solve the matrix product verification problem over any ring using a randomized fingerprinting technique, which has found numerous other applications in theoretical computer science (see, e.g., Ref. [15]).

In the quantum setting, these problems are traditionally studied in the model of quantum query complexity, where we assume the entries of the input matrices are provided by a black box or an oracle. The query complexity of an algorithm is the number of queries made to the oracle. The bounded-error quantum query complexity of a problem is the minimum query complexity of any quantum algorithm that solves the problem with bounded error, i.e., it outputs the correct answer with probability greater than (say) $2/3$. The time complexity of an algorithm refers to the time required to implement the remaining non-query operations. In this article we only consider bounded-error quantum algorithms.

Key Results

It is not known if quantum algorithms can improve the time complexity of the general matrix multiplication problem compared to classical algorithms. Improvements are possible for matrix product verification and special cases of the matrix multiplication problem, as described below.

Matrix Product Verification over Rings

According to Buhrman and Špalek [3], matrix product verification was first studied (in an unpublished paper) by Ambainis, Buhrman, Høyer, Karpinski, and Kurur. Using a recursive application of Grover's algorithm [7], they gave an $O(n^{7/4})$ query algorithm for the problem. The first published work on the topic is due to Buhrman and Špalek [3], who gave an $O(n^{5/3})$ query algorithm for matrix product verification over any ring using a generalization of Ambainis' element distinctness algorithm [1]. This algorithm also achieves the same query complexity over semirings and more general algebraic structures. The algorithm can easily be cast in the quantum walk search framework of Magniez, Nayak, Roland, and Santha [14] as explained in the survey by Santha [16]. More interestingly, they presented an algorithm with time complexity $\tilde{O}(n^{5/3})$ for the problem over fields and integral domains. Their algorithm uses the same technique used by Freivalds [6] and is therefore also time efficient over arbitrary rings. Buhrman and Špalek also proved a lower bound showing that any bounded-error quantum algorithm must make at least $\Omega(n^{3/2})$ queries to solve the problem over the field $\mathbb{F}_2$. This lower bound can be extended to all rings [10].

Theorem 1 (Matrix product verification over rings) *The matrix product verification problem over any ring can be solved by a quantum algorithm with query complexity $O(n^{5/3})$ and time complexity $\tilde{O}(n^{5/3})$. Furthermore, any quantum algorithm must make $\Omega(n^{3/2})$ queries to solve the problem over a ring.*

Buhrman and Špalek also studied the relationship between the complexity of their algorithm and the number of incorrect entries in the purported product, C, and showed that their algorithm performs better when C has a large number of incorrect entries [3].

Matrix Multiplication over Rings

The quantum query complexity of multiplying two $n \times n$ matrices is easy to characterize in terms of the input size. Clearly the query complexity is upper bounded by the input size, $O(n^2)$. On the other hand, if A equals the identity matrix, then $C = B$ and in this case the matrix multiplication problem is equivalent to learning all the bits of an input of size n^2, which requires $\Omega(n^2)$ queries. This follows, for example, from the fact that computing the parity of n^2 bits requires $\Omega(n^2)$ queries [2, 5]. This shows that the quantum query complexity of matrix multiplication is $\Theta(n^2)$, which is the same as the classical query complexity. Similarly, no quantum algorithm is known to improve the time complexity of matrix multiplication over rings compared to classical algorithms.

Buhrman and Špalek [3] studied the matrix multiplication problem in terms of n and an additional parameter ℓ, the number of nonzero entries in the output matrix C, and showed the following result.

Theorem 2 *The matrix multiplication problem over any ring can be solved by a quantum algorithm with query and time complexity upper bounded by*

$$\begin{array}{ll} \tilde{O}(n^{5/3}\ell^{2/3}) & \text{when } 1 \le \ell \le \sqrt{n}, \\ \tilde{O}(n^{3/2}\ell) & \text{when } \sqrt{n} \le \ell \le n, \text{ and} \\ \tilde{O}(n^2\sqrt{\ell}) & \text{when } n \le \ell \le n^2, \end{array}$$

where ℓ is the number of nonzero entries in the output matrix C.

When ℓ is small, this algorithm achieves subquadratic time complexity and when ℓ approaches n^2, its time complexity is close to $O(n^3)$, which is trivial and slower than known classical algorithms. A detailed comparison of this quantum algorithm with classical algorithms may be found in Ref. [3].

Boolean Matrix Product Verification

Buhrman and Špalek [3] also studied the matrix product verification problem over the Boolean semiring and showed that the problem can be solved with query and time complexity $O(n^{3/2})$. On the other hand, the best known lower bound is only $\Omega(n^{1.055})$ queries due to Childs, Kimmel, and Kothari [4].

Theorem 3 (Boolean matrix product verification) *The Boolean matrix product verification problem can be solved by a quantum algorithm with query complexity $O(n^{3/2})$ and time complexity $\tilde{O}(n^{3/2})$. Furthermore, any quantum algorithm must make $\Omega(n^{1.055})$ queries to solve the problem.*

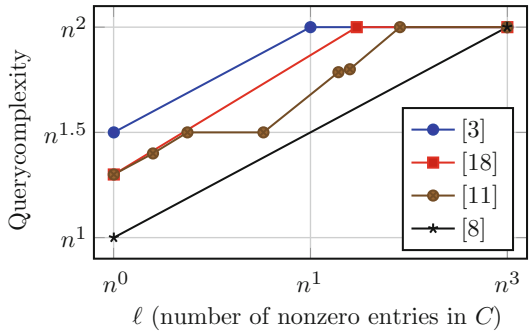
Boolean Matrix Multiplication

As before, the quantum query complexity of multiplying two $n \times n$ Boolean matrices is $\Theta(n^2)$, since it is at least as hard as learning n^2 input bits. The time complexity of Boolean matrix multiplication can be improved to $\tilde{O}(n^{2.5})$ by observing that the inner product of two Boolean vectors of length n can be computed with $O(\sqrt{n})$ queries using Grover's algorithm [7]. This observation also speeds up matrix multiplication over some other semirings.

Similar to the matrix multiplication problem over rings, Boolean matrix multiplication can be studied in terms of an additional parameter ℓ, the number of nonzero entries in the output matrix. Indeed, the problem has been extensively studied in this setting.

Buhrman and Špalek [3] observed that two Boolean matrices can be multiplied with query complexity $O(n^{3/2}\sqrt{\ell})$. This upper bound was improved by Vassilevska Williams and Williams [18], who presented an algorithm with query complexity $\tilde{O}(\min\{n^{1.3}\ell^{17/30}, n^2 + n^{13/15}\ell^{47/60}\})$, which was then improved by Le Gall [11]. Finally, Jeffery, Kothari, and Magniez [8] presented a quantum algorithm for Boolean matrix multiplication that makes $\tilde{O}(n\sqrt{\ell})$ queries. These upper bounds are depicted in Fig. 1. The log factors present in their algorithm were later removed to yield an algorithm with query complexity $O(n\sqrt{\ell})$ [9]. Jeffery, Kothari, and Magniez [8] also proved a matching lower bound of $\Omega(n\sqrt{\ell})$ when $\ell \le \epsilon n^2$ for any constant $\epsilon < 1$. Their algorithm can also be modified to achieve time complexity $\tilde{O}(n\sqrt{\ell} + \ell\sqrt{n})$ [12].

Q



Quantum Algorithms for Matrix Multiplication and Product Verification, Fig. 1 Upper bounds on the quantum query complexity of Boolean matrix multiplication

Theorem 4 (Boolean matrix multiplication) *The Boolean matrix multiplication problem can be solved by a quantum algorithm with query complexity* $O(n\sqrt{\ell})$. *Furthermore, any quantum algorithm that solves the problem must make* $\Omega(n\sqrt{\ell})$ *queries when* $\ell \leq \epsilon n^2$ *for any constant* $\epsilon < 1$. *Boolean matrix multiplication can be solved in time* $\tilde{O}(n\sqrt{\ell} + \ell\sqrt{n})$.

Recently the problem has also been studied in terms of the sparsity of the input matrix. Le Gall and Nishimura [13] present algorithms with improved time complexity in this case. Their algorithm's time complexity is a complicated function of the parameters and the reader is referred to Ref. [13] for details.

Matrix Multiplication over Other Semirings

Le Gall and Nishimura [13] recently initiated the study of matrix multiplication over semirings other than the Boolean semiring and presented algorithms with improved time complexity for the (max, min)-semiring and related semirings.

Open Problems

Several open problems remain in the time and query complexity settings. In the time complexity setting, a major open problem is whether quantum algorithms can solve the matrix multiplication problem faster than classical algorithms over any ring. In the query complexity setting, the complexity of matrix product verification over rings and the Boolean semiring remains open. The best upper and lower bounds are presented in Theorems 1 and 3. A more comprehensive survey of the quantum query complexity of matrix multiplication and its relation to other problems studied in quantum query complexity such as triangle finding and graph collision can be found in the first author's PhD thesis [10], which also contains additional open problems.

Cross-References

- Quantum Algorithm for Element Distinctness
- Quantum Analogues of Markov Chains
- Quantum Search

Recommended Reading

1. Ambainis A (2007) Quantum walk algorithm for element distinctness. SIAM J Comput 37(1):210–239
2. Beals R, Buhrman H, Cleve R, Mosca M, de Wolf R (2001) Quantum lower bounds by polynomials. J ACM 48(4):778–797
3. Buhrman H, Špalek R (2006) Quantum verification of matrix products. In: Proceedings of 17th ACM-SIAM symposium on discrete algorithms, Miami, pp 880–889
4. Childs AM, Kimmel S, Kothari R (2012) The quantum query complexity of read-many formulas. In: Algorithms – ESA 2012. Volume 7501 of lecture notes in computer science. Springer, Heidelberg, pp 337–348
5. Farhi E, Goldstone J, Gutmann S, Sipser M (1998) Limit on the speed of quantum computation in determining parity. Phys Rev Lett 81(24):5442–5444
6. Freivalds R (1979) Fast probabilistic algorithms. In: Mathematical foundations of computer science. Volume 74 of lecture notes in computer science. Springer, Berlin, pp 57–69
7. Grover LK (1996) A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th ACM symposium on theory of computing (STOC 1996), Philadelphia, pp 212–219
8. Jeffery S, Kothari R, Magniez F (2012) Improving quantum query complexity of boolean matrix multiplication using graph collision. In: Automata, languages, and programming. Volume 7391 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 522–532
9. Kothari R (2014) An optimal quantum algorithm for the oracle identification problem. In: Proceedings

of the 31st international symposium on theoretical aspects of computer science (STACS 2014), Lyon. Volume 25 of Leibniz international proceedings in informatics (LIPIcs), pp 482–493
10. Kothari R (2014) Efficient algorithms in quantum query complexity. PhD thesis, University of Waterloo
11. Le Gall F (2012) Improved output-sensitive quantum algorithms for Boolean matrix multiplication. In: Proceedings of the 23rd ACM-SIAM symposium on discrete algorithms (SODA 2012), Kyoto, pp 1464–1476
12. Le Gall F (2012) A time-efficient output-sensitive quantum algorithm for Boolean matrix multiplication. In: Algorithms and computation. Volume 7676 of lecture notes in computer science. Springer, Berlin, pp 639–648
13. Le Gall F, Nishimura H (2014) Quantum algorithms for matrix products over semirings. In: Algorithm theory – SWAT 2014. Volume 8503 of lecture notes in computer science. Springer, Berlin, pp 331–343
14. Magniez F, Nayak A, Roland J, Santha M (2011) Search via quantum walk. SIAM J Comput 40(1):142–164
15. Motwani R, Raghavan P (1995) Randomized algorithms. Cambridge University Press, New York
16. Santha M (2008) Quantum walk based search algorithms. In: Theory and applications of models of computation. Volume 4978 of lecture notes in computer science. Springer, New York, pp 31–46
17. Strassen V (1969) Gaussian elimination is not optimal. Numerische Mathematik 13:354–356
18. Williams VV, Williams R (2010) Subcubic equivalences between path, matrix and triangle problems. In: Proceedings of the 51st IEEE symposium on foundations of computer science (FOCS 2010), Las Vegas, pp 645–654

Quantum Algorithms for Simulated Annealing

Sergio Boixo[1] and Rolando D. Somma[2]
[1]Quantum A.I. Laboratory, Google, Venice, CA, USA
[2]Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM, USA

Keywords

Adiabatic quantum state transformations; Combinatorial optimization; Quantum algorithms; Simulated annealing

Years and Authors of Summarized Original Work

2008; Somma, Boixo, Barnum, Knill
2009; Boixo, Knill, Somma
2014; Chiang, Xu, Somma

Problem Definition

This problem is concerned with the development of quantum methods to speed up classical algorithms based on simulated annealing (SA).

SA is a well-known and powerful strategy to solve discrete combinatorial optimization problems [1]. The search space $\Sigma = \{\sigma_0, \ldots, \sigma_{d-1}\}$ consists of d configurations σ_i, and the goal is to find the (optimal) configuration that corresponds to the global minimum of a given cost function $E : \Sigma \to \mathbf{R}$. Monte Carlo implementations of SA generate a stochastic sequence of configurations via a sequence of Markov processes that converges to the low-temperature Gibbs (probability) distribution, $\pi_{\beta_m}(\Sigma) \propto \exp(-\beta_m E(\Sigma))$. If β_m is sufficiently large, sampling from the Gibbs distribution outputs an optimal configuration with large probability, thus solving the combinatorial optimization problem. The annealing process depends on the choice of an annealing schedule, which consists of a sequence of $d \times d$ stochastic matrices (transition rules) $S(\beta_1)$, $S(\beta_2), \ldots,$ $S(\beta_m)$. Such matrices are determined, e.g., by using Metropolis-Hastings [2]. The real parameters β_j denote a sequence of "inverse temperatures." The implementation complexity of SA is given by m, the number of times that transition rules must be applied to converge to the desired Gibbs distribution (within arbitrary precision). Commonly, the stochastic matrices are sparse, and each list of nonzero conditional probabilities and corresponding configurations, $\{\Pr_\beta(\sigma_j|\sigma_i), j : \Pr_\beta(\sigma_j|\sigma_i) > 0\}$, can be efficiently computed on input (i, β). This implies an efficient Monte Carlo implementation of each Markov process. When a lower bound on the spectral gap of the stochastic matrices (i.e., the difference between the two largest eigenvalues)

is known and given by $\Delta > 0$, one can choose $(\beta_{k+1} - \beta_k) \propto \Delta / E_{\max}$ and $\beta_0 = 0$, $\beta_m \propto \log \sqrt{d}$. $E_{\max}$ is an upper bound on $\max_\sigma |E(\sigma)|$. The constants of proportionality depend on the error probability ϵ, which is the probability of not finding an optimal solution after the transition rules have been applied. These choices result in a complexity $m \propto E_{\max} \log \sqrt{d}/\Delta$ for SA [3].

Quantum computers can theoretically solve some problems, such as integer factorization, more efficiently than classical computers [4]. This work addresses the question of whether quantum computers could also solve combinatorial optimization problems more efficiently or not. The answer is satisfactory in terms of Δ (Section "Key Results"). The complexity of a quantum algorithm is determined by the number of elementary steps needed to prepare a quantum state that allows one to sample from the Gibbs distribution after measurement. Similar to SA, such a complexity is given by the number of times a unitary corresponding to the stochastic matrix is used. For simplicity, we assume that the stochastic matrices are sparse and disregard the cost of computing each list of nonzero conditional probabilities and configurations, as well as the cost of computing $E(\sigma)$. We also assume $d = 2^n$ and the space of configurations Σ is represented by n-bit strings. Some assumptions can be relaxed.

Problem

INPUT: *An objective function* $E : \Sigma \to \mathbf{R}$, *sparse stochastic matrices* $S(\beta)$ *satisfying the detailed balance condition, a lower bound* $\Delta > 0$ *on the spectral gap of* $S(\beta)$, *an error probability* $\epsilon > 0$.

OUTPUT: *A random configuration* $\sigma_i \in \Sigma$ *such that* $\Pr(\sigma_i \in S_0) \geq 1 - \epsilon$, *where* S_0 *is the set of optimal configurations that minimize* E.

Key Results

The main result is a quantum algorithm, referred to as quantum simulated annealing (QSA), that solves a combinatorial optimization problem with high probability using $m_Q \propto E_{\max} \log \sqrt{d}/\sqrt{\Delta}$ unitaries corresponding to the stochastic matrices [5]. The quantum speedup is in the spectral gap, as $1/\sqrt{\Delta} \ll 1/\Delta$ when $\Delta \ll 1$.

Computationally hard combinatorial optimization problems are typically manifest in a spectral gap that decreases exponentially fast in $\log d$, the problem size. The quadratic improvement in the gap is then most significant in hard instances. The QSA algorithm is based on ideas and techniques from quantum walks and the quantum Zeno effect. The quantum Zeno effect can be implemented by evolution randomization [6]. Nevertheless, recent results on "spectral gap amplification" allow for other quantum algorithms that result in a similar complexity scaling [7].

Quantum Walks for QSA

A quantization of the classical random walk is obtained by first defining a $d^2 \times d^2$ unitary matrix that satisfies [8–10]

$$X|\sigma_i\rangle|\mathbf{0}\rangle = \sum_{j=0}^{d-1} \sqrt{\Pr_\beta(\sigma_j|\sigma_i)}|\sigma_i\rangle|\sigma_j\rangle \ . \quad (1)$$

The configuration $\mathbf{0}$ represents a simple configuration, e.g., $\mathbf{0} \equiv \sigma_0 = 0 \ldots 0$ (the n-bit string), and $\Pr_\beta(\sigma_j|\sigma_i)$ are the entries of the stochastic matrix $S(\beta)$. The other $d^2 \times d^2$ unitary matrices used by QSA are P, the permutation (swap) operator that transforms $|\sigma_i\rangle|\sigma_j\rangle$ into $|\sigma_j\rangle|\sigma_i\rangle$, and $R = \mathbb{1} - 2|\mathbf{0}\rangle\langle\mathbf{0}|$, the reflection operator over $|\mathbf{0}\rangle$.

The quantum walk is $W = X^\dagger PXPRPX^\dagger PXR$, and the detailed balance condition implies [5]

$$W \sum_{i=0}^{d-1} \sqrt{\pi_\beta(\sigma_i)}|\sigma_i\rangle|\mathbf{0}\rangle = \sum_{i=0}^{d-1} \sqrt{\pi_\beta(\sigma_i)}|\sigma_i\rangle|\mathbf{0}\rangle \ , \quad (2)$$

where $\pi_\beta(\sigma_i)$ are the probabilities given by the Gibbs distribution. (X, $X^\dagger$, and W also depend on β.) The goal of QSA is to prepare the corresponding eigenstate of W in Eq. 2, within certain precision $\epsilon > 0$, and for inverse temperature $\beta_m \propto \log d$. A projective quantum measurement

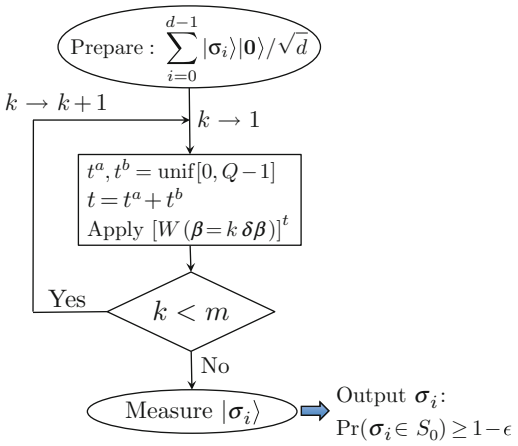
of $|\sigma_i\rangle$ on such a state outputs an optimal solution in the set S_0 with probability $\Pr(S_0) \geq 1 - \epsilon$.

Evolution Randomization and QSA Implementation

The QSA is based on the idea of adiabatic state transformations [6, 11]. For $\beta = 0$, the initial eigenstate of W is $\sum_{i=0}^{d-1} |\sigma_i\rangle|\mathbf{0}\rangle/\sqrt{d}$, which can be prepared easily on a quantum computer. The purpose of QSA is then to drive this initial state towards the eigenstate of W for inverse temperature β_m, within given precision. This is achieved by applying the sequence of unitary operations $[W(\beta_m)]^{t_m} \ldots [W(\beta_2)]^{t_2}[W(\beta_1)]^{t_1}$ to the initial state (Fig. 1). In contrast to SA, $(\beta_{k+1} - \beta_k) \propto 1/E_{\max}$ [11], but the initial and final inverse temperatures are also $\beta_0 = 0$ and $\beta_m \propto \log\sqrt{d}$. This implies that the number of different inverse temperatures in QSA is $m \propto E_{\max} \log\sqrt{d}$, where the constant of proportionality depends on ϵ. The nonnegative integers t_k can be sampled randomly according to several distributions [6]. One way is to obtain t_k after sampling multiple (but constant) times from a uniform distribution on integers between 0 and $Q - 1$, where $Q = \lceil 2\pi/\sqrt{\Delta}\rceil$. The average cost of QSA is then $m\langle t_k\rangle \propto E_{\max} \log\sqrt{d}/\sqrt{\Delta}$. One can use Markov's inequality to avoid those (improbable) instances where the cost is significantly greater than the average cost. The QSA and the values of the constants are given in detail in Fig. 1.

Analytical Properties of W

The quantum walk W has eigenvalues $e^{\pm i\phi_j}$, for $j = 0, \ldots, d-1$, in the relevant subspace. In particular, $\phi_0 = 0 < \phi_1 \leq \ldots \leq \phi_{d-1}$ and $\phi_1 \geq \sqrt{\Delta}$ [5, 7–9]. This implies that the relevant spectral gap for methods based on quantum adiabatic state transformations is of order $\sqrt{\Delta}$. The quantum speedup follows from the fact that the complexity of such methods, recently discussed in [6, 11–13], depends on the inverse of the relevant gap.



Quantum Algorithms for Simulated Annealing, Fig. 1 Flow diagram for the QSA. Under the assumptions, the input state can be easily prepared on a quantum computer by applying a sequence of n Hadamard gates on n qubits. unif$[0, Q-1]$ is the uniform distribution on nonnegative integers in that range and $Q = \lceil 2\pi/\sqrt{\Delta}\rceil$. $\delta\beta = \beta_{k+1} - \beta_k = \epsilon/(2E_{\max})$ and $m = \lceil 2\beta_m E_{\max}/\epsilon\rceil$. Like SA, the final inverse temperature is $\beta_m = (\gamma/2)\log(2\sqrt{d}/\epsilon)$, where γ is the gap of E, that is, $\gamma = \min_{\sigma\notin S_0} E(\sigma) - E(S_0)$. The average cost of the QSA is then $mQ = \lceil 2\pi\gamma E_{\max}\log(2\sqrt{d}/\epsilon)/(\epsilon\sqrt{\Delta})\rceil$, and dependence on ϵ can be made fully logarithmic by repeated executions of the algorithm. A quantum computer implementation of W can be efficiently done by using the algorithm that computes the nonzero conditional probabilities of the stochastic matrix $S(\beta)$

Applications

Like SA, QSA can be applied to solve general discrete combinatorial optimization problems [14]. QSA is often more efficient than exhaustive search in finding the optimal configuration. Examples of problems where QSA can be powerful include the simulation of equilibrium states of Ising spin glasses or Potts models, solving satisfiability problems or solving the traveling salesman problem.

Open Problems

Some (classical) Monte Carlo implementations do not require varying an inverse temperature and apply the same (time-independent) transition rule

S to converge to the Gibbs distribution. The number of times the transition rule must be applied is the so-called mixing time, which depends on the inverse spectral gap of S [15]. The development of quantum algorithms to speed up this type of Monte Carlo algorithms remains open. Also, the technique of spectral gap amplification outputs a Hamiltonian $H(\beta)$ on input $S(\beta)$. The relevant eigenvalue of such a Hamiltonian is zero, and the remaining eigenvalues are $\pm\sqrt{\lambda_i}$, where $\lambda_i \geq \Delta$. This opens the door to a quantum adiabatic version of the QSA, in which $H(\beta)$ is changed slowly and the quantum system remains in an "excited" eigenstate of eigenvalue zero at all times. The speedup is also due to the increase in the eigenvalue gap. Nevertheless, finding a different Hamiltonian path with the same gap, where the adiabatic evolution occurs within the lowest energy eigenstates of the Hamiltonians, is an open problem.

Cross-References

▸ Quantum Algorithms for Simulated Annealing

Recommended Reading

1. Kirkpatrick S, Gelett CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671
2. Hastings WK (1970) Monte Carlo sampling methods using Markov Chains and their applications. Biometrika 57(1):97–109
3. Aldous DJ (1982) Some inequalities for reversible Markov Chains. J Lond Math Soc s2–25:564
4. Shor P (1994) Proceedings of the 35th annual symposium on foundations of computer science, Santa Fe
5. Somma R, Boixo S, Barnum H, Knill E (2008) Quantum simulations of classical annealing processes. Phys Rev Lett 101:130504
6. Boixo S, Knill E, Somma R (2009) Eigenpath traversal by phase randomization. Quantum Inf Comput 9:0833
7. Somma R, Boixo S (2013) Spectral gap amplification. SIAM J Comput 42:593
8. Ambainis A (2004) Proceedings of the 45th symposium on foundations of computer science, Rome
9. Szegedy M (2004) Proceedings of the 45th IEEE symposium on foundations of computer science, Rome
10. Magniez F, Nayak A, Roland J, Santha M (2007) Proceedings of the 39th annual ACM symposium on theory of computing, San Diego
11. Chiang HT, Xu G, Somma R (2014) Improved bounds for eigenpath traversal. Phys Rev A 89:012314
12. Wocjan P, Abeyensinghe (2008) Speedup via quantum sampling. Phys Rev A 78:042336
13. Boixo S, Knill E, Somma R (2010). arXiv:1005.3034
14. Cook WJ, Cunningham WH, Pulleyblank WR (1998) Combinatorial optimization. Wiley, New York
15. Levin DA, Peres Y, Wilmer EL, Markov Chains and mixing times. Available at: http://research.microsoft.com/en-us/um/people/peres/markovmixing.pdf

Quantum Algorithms for Systems of Linear Equations

Aram W. Harrow
Department of Physics, Massachusetts Institute of Technology, Cambridge, MA, USA

Keywords

Filtering; Hamiltonian simulation; Linear algebra; Matrix inversion; Phase estimation

Years and Authors of Summarized Original Work

2009; Harrow, Hassidim, Lloyd
2012; Ambainis

Problem Definition

The problem is to find a vector $x \in \mathbb{C}^N$ such that $Ax = b$, for some given inputs $A \in \mathbb{C}^{N\times N}$ and $b \in \mathbb{C}^N$. Several variants are also possible, such as rectangular matrices A, including overdetermined and underdetermined systems of equations.

Unlike in the classical case, the output of this algorithm is a quantum state on $\log(N)$ qubits whose amplitudes are proportional to the entries of x, along with a classical estimate of $\|x\| :=$

$\sqrt{\sum_i |x_i|^2}$. Similarly, the input b is given as a quantum state. The matrix A is specified implicitly as a row-computable matrix. Specifying the input and output in this way makes it possible to find x in time sublinear, or even polylogarithmic, in N. The next section has more discussion of the relation of this algorithm to classical linear systems solvers.

Key Results

Suppose that:

- $A \in \mathbb{C}^{N \times N}$ is Hermitian, has all eigenvalues in the range $[-1, -1/\kappa] \cup [1/\kappa, 1]$ for some known $\kappa \geq 1$, and has $\leq s$ nonzero entries per row. The parameter κ is called the *condition number* (defined more generally to be the ratio of the largest to the smallest singular value) and s is the *sparsity*.
- There is a quantum algorithm running in time T_A that takes an input $i \in [N]$ and outputs the nonzero entries of the ith row, together with their location.
- Assume that $\|b\| = 1$ and that there is a corresponding quantum state to produce the state $|b\rangle$ that runs in time T_B.

Define $x' := A^{-1}|b\rangle$ and $x = \frac{x'}{\|x'\|}$.

We use the notation x to refer to the vector as a mathematical object and $|x\rangle$ to refer to the corresponding quantum state on $\log(N)$ qubits. For a variable T, let $\tilde{O}(T)$ denote a quantity upper bounded by $T \cdot \text{poly}\log(T)$. The norm of a vector $\|x\|$ is the usual Euclidean norm $\sqrt{\sum_i |x_i|^2}$, while for a matrix $\|A\|$ is the operator norm $\max_{\|x\|=1} \|Ax\|$, or equivalently the largest singular value of A.

Quantum Algorithm for Linear Systems

The main result is that $|x\rangle$ and $\|x'\|$ can be produced, both up to error ϵ, in time $\text{poly}(\kappa, s, \epsilon^{-1}, \log(N), T_A, T_B)$. More precisely, the following run-times are known:

$$\tilde{O}(\kappa T_B + \log(N) s^2 \kappa^2 T_A / \epsilon) \quad [5] \quad (1a)$$

$$\tilde{O}(\kappa T_B + \log(N) s^2 \kappa T_A / \epsilon^3) \quad [1] \quad (1b)$$

A key subroutine is Hamiltonian simulation, and the run-times in (1) are based on the recent improvements in this component due to [3].

Hardness Results and Comparison to Classical Algorithms

These algorithms are analogous to classical algorithms for solving linear systems of equations, but do not achieve exactly the same thing. Most classical algorithms output the entire vector x as a list of N numbers, while the quantum algorithms output the state $|x\rangle$, i.e., a superposition on $\log(N)$ qubits whose N amplitudes equal x. This allows potentially faster algorithms but for some tasks will be weaker. This resembles the difference between the Quantum Fourier Transform and the classical Fast Fourier Transform.

To compare the classical and quantum complexities for this problem, we should consider classical tasks (with classical output) that can be solved with the help of quantum linear equations algorithms. One can show that better classical algorithms for such tasks exist only if *all* quantum algorithms could be simulated more quickly by classical algorithms. This is because the linear systems problem is BQP-complete, i.e., solving large sparse well-conditioned linear systems of equations is equivalent in power to general purpose quantum computing.

To make this precise, define LinearSystemSample$(N, \kappa, \epsilon, T_A)$ to be the problem of producing a sample $i \in [N]$ from a distribution p satisfying $\sum_{i=1}^{N} |p_i - |x_i|^2| \leq \epsilon$, where $x = x'/\|x'\|$, $x' = A^{-1}b$, and $b = e_1$ (i.e., one in the first entry and zero elsewhere). Additionally the eigenvalues of A should have absolute value between $1/\kappa$ and 1, and there should exist a classical algorithm for computing the entries of a row of A that runs in time T_A. This problem differs slightly from the version described above, but only in ways that make it easier, so that it still makes sense to talk about a matching hardness result.

Theorem 1 *Consider a quantum circuit on n qubits that applies two-qubit unitary gates $U_1, \ldots, U_T$ to the $|0\rangle^{\otimes n}$ state and concludes by outputting the result of measuring the first qubit. It is possible to simulate this measurement outcome up to error ϵ by reducing to* LinearSystemSample$(N, \kappa, \epsilon/2, T_A)$ *with* $N = O(2^n T/\epsilon)$, $\kappa = O(T/\epsilon)$, *and* $T_A = \text{poly}\log(N)$.

In other words, LinearSystemSample is at least as hard to solve as any quantum computation of the appropriate size. This result is nearly tight. In other words, when combined with the algorithm of [1], the relation between N, κ (for linear system solving) and n, T (for quantum circuits) is known to be nearly optimal, while the correct ϵ dependence is known up to a polynomial factor.

Theorem 1 can also rule out classical algorithms for LinearSystemSample$(N, \kappa, \epsilon, T_A)$. Known algorithms for the problem (assuming for simplicity that A is s-sparse) run in time $\text{poly}(N)\,\text{poly}\log(\kappa/\epsilon) + NT_A$ (direct solvers), $N\,\text{poly}(\kappa)\,\text{poly}\log(1/\epsilon)T_A$ (iterative methods), or even $s^{\kappa \ln(1/\epsilon)}\,\text{poly}\log(N)$ (direct expansion of $x \approx \sum_{n \le \kappa \ln(1/\epsilon)} (I - A)^n b$, assuming A is positive semidefinite). Depending on the parameters N, κ, ϵ, s, a different one of these may be optimal. And from Theorem 1 it follows (a) that any nontrivial improvement in these algorithms would imply a general improvement in the ability of classical computers to simulate quantum mechanics and (b) that such improvement is impossible for algorithms that use the function describing A in a black-box manner (i.e., as an oracle).

Applications and Extensions

Linear system solving is usually a subroutine in a larger algorithm, and the following algorithms apply it to a variety of settings. Complexity analyses can be found in the cited papers, but since hardness results are not known for them, we cannot say definitively whether they outperform all possible classical algorithms.

Machine Learning

A widely used application of linear systems of equations is to performing least-squares estimation of a model [6]. In this problem, we are given a matrix $A \in \mathbb{R}^{n \times p}$ with $n \ge p$ (for an overdetermined model) along with a vector $b \in \mathbb{R}^n$, and we wish to compute $\arg\min_{x \in \mathbb{R}^p} \|Ax - b\|$. If A is well conditioned, sparse, and implicitly specified, then the state $|x\rangle$ can be found quickly [6], and from this features of x can be extracted by measurement.

Differential Equations

Consider the differential equation [2]

$$\dot{x}(t) = A(t)x(t) + b(t) \qquad x(t) \in \mathbb{R}^N. \quad (2)$$

One of the simplest ways to solve this is to discretize time to take values $t_1 < \ldots < t_m$ and approximate

$$x(t_{i+1}) \approx x(t_i) + (A(t_i)x(t_i) + b(t_i))(t_{i+1} - t_i). \quad (3)$$

By treating $(x(t_1), \ldots, x(t_m))$ as a single vector of size Nm, we can find this vector as a solution of the linear system of equations specified by (3). More sophisticated higher-order solvers can also be made quantum; see [2] for details.

Boundary-Value Problems

The solution to PDEs can also be expressed in terms of the solution to a linear system of equations [4]. For example, in Poisson's equation we are given a function $Q : \mathbb{R}^3 \to \mathbb{R}$ and want to find $u : \mathbb{R}^3 \to \mathbb{R}$ such that $-\nabla^2 u = Q$. By defining x and b to be discretized versions of u, Q, this PDE becomes an equation of the form $Ax = b$. One challenge is that if A is the finite-difference operator (i.e., discretized second derivative) for an $L \times L \times L$ box, then its condition number will scale as L^2. Since the total number of points is $O(L^3)$, this means the quantum algorithm cannot achieve a substantial speedup. Classically this condition number is typically reduced by using preconditioners. A method for using preconditioners with the quantum linear system solver was presented in [4], along with an application to an electromagnetic scattering

problem. The resulting complexity is still not known.

Cross-References

▸ Quantum Analogues of Markov Chains

Recommended Reading

1. Ambainis A (2012) Variable time amplitude amplification and quantum algorithms for linear algebra problems. In: STACS, Paris, vol 14, pp 636–647
2. Berry DW (2014) High-order quantum algorithm for solving linear differential equations. J Phys A 47(10):105301
3. Berry DW, Childs AM, Cleve R, Kothari R, Somma RD (2014) Exponential improvement in precision for simulating sparse hamiltonians. In: Proceedings of STOC 2014, New York, pp 283–292
4. Clader BD, Jacobs BC, Sprouse CR (2013) Preconditioned quantum linear system algorithm. Phys Rev Lett 110:250504
5. Harrow AW, Hassidim A, Lloyd S (2009) Quantum algorithm for solving linear systems of equations. Phys Rev Lett 15(103):150502
6. Wiebe N, Braun D, Lloyd S (2012) Quantum algorithm for data fitting. Phys Rev Lett 109:050505

Quantum Analogues of Markov Chains

Ashwin Nayak[1], Peter C. Richter[1,2], and Mario Szegedy[1]
[1]Department of Combinatorics and Optimization, and Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada
[2]Department of Computer Science, Rutgers, The State University of New Jersey, New Brunswick, NJ, USA

Keywords

Element distinctness; Hitting time; Markov chains; Quantum algorithms; Quantum search; Quantum walks; Search problem; Spatial search; Triangle finding

Years and Authors of Summarized Original Work

2004; Szegedy

Problem Definition

Spatial Search and Walk Processes

Spatial search by *quantum walk* is database search with the additional constraint that one is required to move through the search space that obeys some locality structure. For example, the data items may be stored at the vertices of a two-dimensional grid. The requirement of moves along the edges of the grid captures the cost of accessing different items starting from some fixed position in the database.

One of possible ways of carrying out spatial search is by performing a random walk on the search space or its quantum analog, a quantum walk. The complexity of spatial search by quantum walk is strongly tied to the *quantum hitting time* [19] of the walk.

Let S, with $|S| = n$, be a finite set of *states*. Assume that a subset $M \subseteq S$ of states are *marked*. We are given a procedure $\mathcal{C}$ that, on input $x \in S$ and an associated data structure $d(x)$, checks whether the state x is marked. The goal is either to find a marked state when promised that $M \neq \emptyset$ (*search version*) or to determine whether M is nonempty (*decision version*).

The algorithm progresses in stages. In the *setup stage*, we access some state of S (usually a random state). In the walk stage we move from state to state, performing a spatial walk as described below. The moves are called *updates*. In addition, in the walk stage we perform *checks* to see if the current state is marked at steps selected by the algorithm.

In the classical setting, the *transition probabilities* of the spatial walk are described by a stochastic matrix $P = (p_{x,y})_{x,y \in S}$. This makes the walk a *Markov chain*. In every move the

algorithm *must perform* a random transition according to P. The possible $x \to y$ moves, i.e., those with $p_{x,y} \neq 0$, form the edges of a (directed) graph G, and we say that the Markov chain P has *locality structure* G.

We define the search problem in the classical setting, which carries over to the quantum case with little modification:

INPUT: Markov chain P on set S, marked subset $M \subseteq S$ that is implicitly specified by a checking procedure $\mathcal{C}$, and the associated costs:

Cost type	Setup	Update	Checking
Notation	S	U	C

OUTPUT: a marked state if one exists (search version) or a Boolean return value that indicates whether M is empty or not (decision version).

The algorithm is required to be correct with probability at least $2/3$ in either case, the search or the decision problem. The significance of the setup cost, which is incurred only once, will be clearer when we see some applications. Often we can choose between several competing walks, and we would like to design the one with minimum total cost.

In the *quantum* case, the random process P is replaced by a *quantum walk* W_P that has the same locality structure as P. The costs $\mathsf{S}, \mathsf{U}, \mathsf{C}$ reflect the costs of quantum operations.

The Quantum Walk Algorithm

Designing a quantum analog of P is not so straightforward, since stochastic matrices have no immediate unitary equivalents. One either needs to abandon the discrete-time nature of the walk [15] or define the walk operator on a space other than $\mathbb{C}^S$. Here we take the second route.

We say that a Markov chain P is *irreducible* if its underlying digraph is strongly connected. Let P be an irreducible Markov chain, let π be its unique stationary distribution, and let P^* (with $P^* = (p^*_{x,y})$) denote the *time-reversed Markov chain*, where $p^*_{x,y} := \pi_y p_{y,x}/\pi_x$. Define the following vectors in the vector space $\mathbb{C}^S$:

$$|p_x\rangle := \sum_{y \in X} \sqrt{p_{x,y}}\,|y\rangle \qquad \text{and}$$

$$|p^*_y\rangle := \sum_{x \in X} \sqrt{p^*_{y,x}}\,|x\rangle \ .$$

Define the unitary operator $W_P := R_1 R_2$ on $\mathbb{C}^{S \times S}$ as the product of the two reflections $R_2 := \sum_{x \in S} |x\rangle\langle x| \otimes (2|p_x\rangle\langle p_x| - \mathrm{I})$ and $R_1 := \sum_{y \in S} (2|p^*_y\rangle\langle p^*_y| - \mathrm{I}) \otimes |y\rangle\langle y|$. The operator W_P is called the *quantum analog* of P, or the *discrete-time quantum walk operator* arising from P, and may be viewed as a walk on the *edges* of the underlying graph G. We define a "checking" operator on $\mathbb{C}^S$, based on whether or not the current state is marked: $O_M := \sum_{x \notin M} |x\rangle\langle x| - \sum_{x \in M} |x\rangle\langle x|$.

In the above description, we have suppressed the data structure associated with a state in the Markov chain for the sake of simplicity. The precise description of the operators can be derived via the isometry $|x\rangle \mapsto |x\rangle|d(x)\rangle$ between the appropriate spaces (see, e.g., Refs. [28, 29]). The data structure becomes especially significant in the context of the complexity of the operators.

A search algorithm by quantum walk is described by a quantum circuit that acts on "registers" or "wires" which are associated with the space $\mathbb{C}^S \otimes \mathbb{C}^S \otimes \mathbb{C}^k$, for some $k \geq 0$. We again suppress the registers carrying the data structure. The first two registers hold the current edge, and the last register holds auxiliary information, or work space, that drives the quantum walk. The quantum circuit implements the composition $X := X_t X_{t-1} \cdots X_1$, where each X_i is either W_P or O_M acting on the edge registers, possibly controlled by the auxiliary register, or a unitary operator independent of P and M acting on any of the registers. The circuit X is applied to a suitably constructed initial state $|\phi_0\rangle$.

We associate a cost with each operator as a measure of its complexity, with respect to a resource of interest. The resource could be circuit size or in the query model (which is the more typical application) the number of queries. We denote the cost of implementing W_P as a quantum circuit in the units of the resource of interest by U (*update cost*), the cost of construct-

ing O_M by C (*checking cost*), and the cost of preparing the initial state, $|\phi_0\rangle$, of the algorithm by S (*setup cost*). Every time an operator is used, we incur the cost associated with it. This abstraction, implicit in Ref. [3] and made explicit in Ref. [28], allows W_P and O_M to be treated as black-box operators and provides a convenient way to capture *time complexity* or, in the quantum query model, *query complexity*. The cost of the sequence $X_t X_{t-1} \cdots X_1$ is the sum of the costs of the individual operators. The *observation probability* is the probability that we observe an element of M on measuring the first register of the final state, $|\phi_t\rangle := X|\phi_0\rangle$, in the standard basis $(|x\rangle)_{x\in S}$. In the decision version of the problem, we measure a fixed single qubit of the auxiliary register in the standard basis to obtain the output of the algorithm.

Key Results

Walk Definitions

Quantum walks were first introduced by David Meyer and John Watrous to study quantum cellular automata and quantum logspace, respectively. Discrete-time quantum walks were investigated for their own sake by Ambainis, Bach, Nayak, Vishwanath, and Watrous [4, 32] and Aharonov, Ambainis, Kempe, and Vazirani [2] on the infinite line and the n-cycle, respectively. The central issues in the early development of quantum walks included the definition of the walk operator, notions of mixing and hitting times, and the speedup achievable compared to the classical setting.

Hitting Time

Exponential quantum speedup of the hitting time between antipodes of the hypercube was shown by Kempe [19]. Childs, Cleve, Deotto, Farhi, Gutmann, and Spielman [13] presented the first oracle problem solvable exponentially faster by a quantum walk-based algorithm than by any (not necessarily walk-based) classical algorithm.

The first systematic studies of quantum hitting time on the hypercube and the d-dimensional torus were conducted by Shenvi, Kempe, and Whaley [34] and Ambainis, Kempe, and Rivosh [5]. Improving upon the Grover search-based spatial search algorithm of Aaronson and Ambainis, Ambainis et al. [5] showed that the d-dimensional torus with n nodes can be searched by quantum walk in $\sqrt{n}$ steps with observation probability $\Omega(1)$ for $d \geq 3$ and in $\sqrt{n \log n}$ in steps and observation probability $\Omega(1/\log n)$ for $d = 2$ (see also Ref. [11]). Combining the algorithm for $d = 2$ with amplitude amplification [9], we get an algorithm with observation probability $\Omega(1)$, at a cost that is a multiplicative factor of $\sqrt{\log n}$ larger.

In the results in Refs. [13, 19], the algorithm has implicit knowledge of the target state, as the walk starts from a state whose location is "related" to that of the target. It is not known if we can achieve an exponential speedup when the walk starts in a state that is independent of the target.

Element Distinctness

The first result that used a quantum walk to solve a natural algorithmic problem, the so-called *element distinctness problem*, was due to Ambainis [3]. The problem is to find out if among the set of s elements of a database, two are identical. Ambainis constructed a walk on the *Johnson graph* $J(r, s)$ whose vertices are the r-size subsets of a universe of size s (in his case the universe corresponds to the set of all database elements), with two subsets connected iff their symmetric difference has size two. A subset is marked, i.e., it is an element of M, if it captures two identical database elements. In the quantum (but also the classical) query model, the setup cost is r, which stands for the cost of downloading r (random) database elements. Update incurs a constant cost, as it requires reading a new database element and forgetting an old one. Furthermore, since we are in the query model, the checking cost is zero, since whether a state is marked can be deduced from the currently held database elements without any further download. Ambainis ingeniously balanced the costs of S and U finding that in the quantum case, the optimum choice for r is $s^{2/3}$, leading to a query complexity of $s^{2/3}$ (this is a nontrivial balance: in the classical case, the same walk gives no speedup).

In contrast, the Grover algorithm, the inspiration behind Ambainis' work, has no balancing option: its setup and update costs are zero in the query model. (The Grover search may be viewed as a quantum walk on the complete graph.) It turns out that the above walk-based quantum query algorithm with complexity $O(s^{2/3})$ matches the lower bound due to Aaronson and Shi [1].

General Markov Chains

Ambainis's result is based on the quantum hitting time of $J(r,s)$ for a marked set of relative size $\left(\frac{r}{s}\right)^2$. In Ref. [35], Szegedy investigates the hitting time of quantum walks arising from general Markov chains. His definitions (walk operator, hitting time) are abstracted directly from Ref. [3] and are consistent with prior literature, although slightly different in presentation.

For a Markov chain P, the (classical) *average hitting time* of M can be expressed in terms of the *leaking walk matrix* P_M, which is obtained from P by deleting all rows and columns indexed by states of M. Let $v_1, \ldots, v_{n-m}$, be the normalized eigenvectors of P_M, and let $\lambda_1, \ldots, \lambda_{n-m}$ be the associated eigenvalues, where $m = |M|$. Let $h(x, M)$ denote the expected time to reach M from x. Let $\mu : S \to \mathbb{R}^+$ be the initial distribution from which we start and μ' its restriction to $S \setminus M$. Denote the vector $(\sqrt{\mu'(x)})_{x \in S \setminus M}$ by u. Then the average hitting time of M is $h := \sum_{x \in S} \mu(x)\, h(x, M) = \sum_{k=1}^{n-m} \frac{|(v_k, u)|^2}{1-\lambda_k}$. Although the leaking walk matrix P_M is not stochastic, one can consider the *absorbing walk* matrix $P' = \left[\begin{smallmatrix} P_M & P'' \\ 0 & I \end{smallmatrix}\right]$, where P'' is the matrix obtained from P by deleting the rows indexed by M and the columns indexed by $S \setminus M$. The walk P' behaves like P but is absorbed by the first marked state it hits. Consider the quantum analog $W_{P'}$ of P' and $|\phi_0\rangle := \sum_{x \in S} \sqrt{\pi(x)}|x\rangle|p_x\rangle$, where π is the stationary distribution of P. The state $|\phi_0\rangle$ is stationary for W_P, i.e., an eigenvector with eigenvalue 1. Define the *quantum hitting time*, H, of set M to be the smallest t for which $\|W_{P'}^t|\phi_0\rangle - |\phi_0\rangle\| \geq 0.1$. Note that the cost of $W_{P'}$ is proportional to $\mathsf{U} + \mathsf{C}$.

The motivation behind this definition of quantum hitting time is the following. The classical hitting time measures the number of iterations of the absorbing walk P' required to noticeably skew the uniform starting distribution. Similarly, the quantum hitting time bounds the number of iterations of the following quantum algorithm for detecting whether M is nonempty: At each step, apply operator $W_{P'}$. If M is empty, then $P' = P$ and the starting state is left invariant. If M is nonempty, then the angle between $W_{P'}^t|\phi_0\rangle$ and $W_P^t|\phi_0\rangle$ gradually increases (for t not too large). Using an additional *control register* to apply either $W_{P'}$ or W_P with quantum control, the divergence of these two states (should M be nonempty) can be detected. The required number of iterations is characterized by H.

It remains to compute H. When P is symmetric and *ergodic*, the expression for the classical hitting time has a quantum analog [35] (we assume $m \leq n/2$ for technical reasons):

$$H \leq \sum_{k=1}^{n-m} \frac{v_k^2}{\sqrt{1-\lambda_k}}, \tag{1}$$

where $v_k = (v_k, u)$. Note that $u = \frac{1}{\sqrt{n}}(1, \ldots, 1)$, since P is symmetric, so v_k sum of the coordinates of v_k divided by $1/\sqrt{n}$. From (1) and the expression for h, one can derive an amazing connection between the classical and quantum hitting times:

Theorem 1 (Szegedy [35]) *Let P be symmetric and ergodic, and let h be the classical hitting time for marked set M and uniform starting distribution. Then the quantum hitting time of M is at most $\sqrt{h}$. Therefore, the cost of solving the decision version of the problem is of order $\mathsf{S} + \sqrt{h}(\mathsf{U} + \mathsf{C})$.*

One can further show:

Theorem 2 (Szegedy [35]) *If P is state-transitive and $|M| = 1$, then the marked state is observed with probability at least n/h with cost $O(\mathsf{S} + \sqrt{h}(\mathsf{U} + \mathsf{C}))$.*

The observation probability n/h can be increased to $\Theta(1)$ with $\sqrt{h/n}$ iterations of the

algorithm from Theorem 2, using amplitude amplification [9]. Theorems 1 and 2 imply most quantum hitting time results of the previous section *directly*, relying only on estimates of the corresponding classical hitting times. Expression (1) is based on a fundamental connection between the eigenvalues and eigenvectors of P and W_P. Notice that $p^*_{y,x} = p_{y,x}$ for symmetric P, so $|p^*_y\rangle = |p_y\rangle$. So R_1 and R_2 are reflections through the subspaces generated by $\{|p_x\rangle \otimes |x\rangle |\ x \in S\}$ and $\{|x\rangle \otimes |p_x\rangle |\ x \in S\}$, respectively. The eigenvalues of $R_1 R_2$ can be expressed in terms of the eigenvalues of the mutual Gram matrix $D(P)$ of these systems. This matrix $D(P)$, the *discriminant matrix* of P, equals P when P is symmetric. The formula remains fairly simple even when P is not symmetric. In particular, the absorbing walk P' has discriminant matrix $\left[\begin{smallmatrix} P_M & 0 \\ 0 & 1 \end{smallmatrix}\right]$. Finally, the relation between $D(P)$ and the spectral decomposition of W_P is given by:

Theorem 3 (Szegedy [35]) *Let P be an arbitrary Markov chain on a finite state space S and let $\cos\theta_1 \geq \cdots \geq \cos\theta_l$ be those singular values of $D(P)$ lying in the open interval $(0, 1)$, with associated singular vector pairs v_j, w_j for $1 \leq j \leq l$. Then the nontrivial eigenvalues of W_P (namely, those other than 1 and -1) and their corresponding eigenvectors are $(e^{-2i\theta_j}, R_1 w_j - e^{-i\theta_j} R_2 v_j)$ and $(e^{2i\theta_j}, R_1 w_j - e^{i\theta_j} R_2 v_j)$ for $1 \leq j \leq l$.*

Subsequent Developments

Magniez, Nayak, Roland, and Santha [29] used the Szegedy quantum analog W_P of an ergodic walk P, rather than that of its absorbing version P', to develop a *search* algorithm in the style of Ambainis [3].

Theorem 4 (Magniez, Nayak, Roland, Santha [29]) *Let P be reversible and ergodic with spectral gap $\delta > 0$. Let M have probability either zero or $\varepsilon > 0$ under the stationary distribution of P. There is a quantum algorithm solving the search problem with cost $\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C})$.*

The main idea here is to apply quantum phase estimation [14, 21] to the quantum walk W_P in order to implement an approximate reflection operator about the initial state. This operator is then used along with the checking operator O_M in an amplitude amplification scheme to get the final algorithm.

The average classical hitting time h may be bounded by $1/\delta\varepsilon$ (with δ, M, ε as in Theorem 4), and this bound is tight for most known applications. In these applications, the above algorithm *finds* marked elements with complexity at most that of the Szegedy algorithm. In other applications, for instance, Triangle Finding [28], where the checking cost C is much larger than the update cost U, the complexity of the algorithm in Theorem 4 is asymptotically smaller.

In the case of the two-dimensional square grid with n vertices, the average classical hitting time h is $n \log n$. This is asymptotically lesser than $1/\delta\varepsilon$ when there is a single marked element. (In this case, $1/\delta\varepsilon = n^2$.) Algorithms due to Ambainis et al. [5] and Szegedy [35] find a unique marked state with $O(\sqrt{n}\ \log n)$ steps of quantum walk, a $\sqrt{\log n}$ factor larger than $\sqrt{h}$. Tulsi [36] showed how we may find a unique marked element in $O(\sqrt{h})$ steps. Magniez, Nayak, Richter, and Santha [30] extended this result to show that for any state-transitive Markov chain, a unique marked state can be found in $O(\sqrt{h})$ steps. They also devised a detection algorithm that solves the decision version of the problem for any reversible Markov chain and any number of marked elements, in $O(\sqrt{h})$ steps (thus extending Theorem 1).

Krovi, Magniez, Ozols, and Roland [23] presented a different quantum algorithm for finding multiple marked elements in any *reversible* Markov chain. They introduced a notion of interpolation between any reversible chain P and its absorbing counterpart P' and used the quantum analog of the interpolated walk. In the case of a unique marked element, the resulting algorithm solves the search version of the problem with cost $\mathsf{S} + \sqrt{h}(\mathsf{U} + \mathsf{C})$. The precise relationship between the number of steps of the quantum walk taken by the algorithm in the case of more than one marked element and the corresponding

Q

classical hitting time remains open. It is known that for certain choices of P and M, the former may be asymptotically larger than $\sqrt{h}$.

The schema due to Magniez et al. [29] described above has been extended in different ways. Jeffery, Kothari, and Magniez [17] use a *quantum state* as the data structure $d(x)$ associated with a state $x \in S$ in quantum algorithms with nested walks. In this manner, they avoid the repeated overhead of setup cost in the inner quantum walks used for checking marked states. They solve several problems, including Triangle Finding, with query as well as time complexity matching, up to polylogarithmic factors, the performance of algorithms previously derived from *learning graphs* [7, 26]. Childs, Jeffery, Kothari, and Magniez [8] introduced the use of a data structure that depends on the state transition in the walk. Using this, they develop quantum algorithms with nested walks, where the recursion occurs in the update operation. The cost incurred is essentially what we would expect from Theorem 4. This extension leads to algorithms that are as efficient in *time* as in query complexity, for applications such as 3-Distinctness. Independently, Belovs designed a different quantum walk algorithm [8], which leads to a similar result for 3-Distinctness.

Applications

We list some quantum walk-based results for search problems that represent speedups over Grover search-based solutions. All are inspired by Ambainis' algorithm for element distinctness.

Triangle Finding

Suppose we are given the adjacency matrix A of a graph on n vertices and are required to determine if the graph contains a triangle (i.e., a clique of size 3), using as few queries as possible to the entries of A. The classical query complexity of this problem is $\Theta(n^2)$. Magniez, Santha, and Szegedy [28] gave an $\tilde{O}(n^{1.3})$ algorithm. This upper bound has been improved by a sequence of results [7, 25, 26, 29] (see also Ref. [17]) to $\tilde{O}(n^{5/4})$. Several of these algorithms, including the current best algorithm due to Le Gall [25], are based on the quantum walk search framework.

Matrix Product Verification and Matrix Multiplication

Suppose we are given three $n \times n$ matrices A, B, C over a ring and are required to determine if $AB \neq C$, i.e., if there exist i, j such that $\sum_k A_{ik} B_{kj} \neq C_{ij}$. We would like to make as few queries as possible to the entries of A, B, and C. This problem has classical query complexity $\Theta(n^2)$. Buhrman and Špalek [10] gave an $O(n^{5/3})$ quantum query algorithm. They also observed that two Boolean matrices can be multiplied with query complexity $O(n^{3/2}\sqrt{\ell})$, where ℓ is the number of nonzero entries in the product. This has since been improved in a sequence of results [16, 24, 37] to $O(n\sqrt{\ell})$. The algorithm due to Le Gall [24] builds upon quantum walk algorithms. We refer the reader to Ref. [22] for further work on this topic.

Group Commutativity Testing

Suppose we are presented with a black-box group specified by its k generators and are required to determine if the group commutes using as few queries as possible to the group product operation (i.e., queries of the form "What is the product of elements g and h?"). The classical query complexity is $\Theta(k)$ group operations. Magniez and Nayak [27] gave an (essentially optimal) $\tilde{O}(k^{2/3})$ quantum query algorithm for this problem. The algorithm involves a quantum walk on the product of two graphs whose vertices are ordered l-tuples of distinct generators.

Forbidden Subgraph Property

A property of graphs is called *minor closed* when the following condition holds: if a graph has the property, then all its minors also possess the property. A graph property (which need not be minor closed) is called a *forbidden subgraph property* (FSP) if it can be described by a finite set of forbidden subgraphs. Suppose we are given the adjacency matrix A of a graph on n vertices and are required to determine if the graph has a minor closed property Π, using as few queries as possible to the entries of A. Childs and Kothari [12]

show that if Π is nontrivial and is *not* FSP, then it has query complexity in $\Theta(n^{3/2})$. They complement this with a more efficient algorithm for any minor closed property Π that *is* FSP. The algorithm has query complexity $O(n^{\alpha})$ for some $\alpha < 3/2$ and is based on the quantum walk search framework.

3-Distinctness

This is a generalization of the element distinctness problem. Suppose we are given elements $x_1, \ldots, x_m \in \{1, \ldots, m\}$ and are asked if there exist *three* distinct indices i, j, k such that $x_i = x_j = x_k$. The Ambainis quantum walk algorithm achieves query and time complexity $O(m^{3/4})$. The *query* complexity was improved to $O(m^{5/7})$ by Belovs [6] using a new technique – learning graphs, while the best time complexity remained unchanged. Childs et al. [8] later designed *time*-efficient query algorithms with complexity $\tilde{O}(m^{5/7})$, using extensions of the quantum walk search framework.

Open Problems

Many issues regarding quantum analogs of Markov chains remain unresolved, both for the search problem and the closely related mixing problem.

Search Problem

Can the quadratic quantum speedup of hitting time for the decision version of the problem be extended from all reversible Markov chains to all *ergodic* ones? Can quantum walks also *find* marked elements quadratically faster than classical walks, in the case of reversible Markov chains with *multiple* marked states? What other algorithmic applications of search by quantum walk can be found?

Sampling Problem

Another wide use of Markov chains in classical algorithms is in generating samples from certain probability distributions. In particular, *Markov chain Monte Carlo* algorithms work by running a carefully designed ergodic Markov chain. After a number of steps given by the *mixing time* of P, the distribution over states is guaranteed to be ϵ-close to its stationary distribution π. Such algorithms form the basis of most randomized algorithms for approximating #P-complete problems (see, e.g., Ref. [18]). The sampling problem may be formalized as follows:

INPUT: Markov chain P, tolerance $\epsilon \in (0, 1)$.
OUTPUT: A sample from a distribution that is ϵ-close to π in total variation distance.

Notions of quantum mixing time were first proposed and analyzed on the line, the cycle, and the hypercube [2,4,31,32]. Kendon and Tregenna [20] and Richter [33] have investigated the use of decoherence in improving mixing of quantum walks. Two fundamental questions about quantum mixing time remain open: What is the "most natural" definition? And when is there a quantum speedup over the classical mixing time?

Cross-References

▸ Quantum Algorithm for Element Distinctness
▸ Quantum Algorithm for Finding Triangle

Recommended Reading

1. Aaronson S, Shi Y (2004) Quantum lower bounds for the collision and the element distinctness problems. J ACM 51(4):595–605
2. Aharonov D, Ambainis A, Kempe J, Vazirani U (2001) Quantum walks on graphs. In: Proceedings of the thirty-third annual ACM Symposium on Theory of Computing, STOC '01, New York. ACM, pp 50–59
3. Ambainis A (2007) Quantum walk algorithm for Element Distinctness. SIAM J Comput 37(1):210–239
4. Ambainis A, Bach E, Nayak A, Vishwanath A, Watrous J (2001) One-dimensional quantum walks. In: Proceedings of the thirty-third annual ACM Symposium on Theory of Computing, STOC '01, New York. ACM, pp 37–49
5. Ambainis A, Kempe J, Rivosh A (2005) Coins make quantum walks faster. In: Proceedings of the sixteenth annual ACM-SIAM Symposium on Discrete Algorithms, SODA '05, Philadelphia. Society for Industrial and Applied Mathematics, pp 1099–1108

6. Belovs A (2012) Learning-graph-based quantum algorithm for k-Distinctness. In: Proceedings of the 53rd annual IEEE Symposium on Foundations of Computer Science. IEEE Computer Society, Los Alamitos, pp 207–216
7. Belovs A (2012) Span programs for functions with constant-sized 1-certificates: extended abstract. In: Proceedings of the forty-fourth annual ACM Symposium on Theory of Computing, STOC '12, New York. ACM, pp 77–84
8. Belovs A, Childs AM, Jeffery S, Kothari R, Magniez F (2013) Time-efficient quantum walks for 3-Distinctness. In: Fomin FV, Freivalds R, Kwiatkowska M, Peleg D (eds) Automata, Languages, and Programming. Volume 7965 of Lecture Notes in Computer Science. Springer, Berlin/Heidelberg, pp 105–122
9. Brassard G, Høyer P, Mosca M, Tapp A (2002) Quantum amplitude amplification and estimation. In: Quantum Computation and Information (Washington, DC, 2000). Volume 305 of Contemporary Mathematics. American Mathematical Society, Providence, pp 53–74
10. Buhrman H, Špalek R (2006) Quantum verification of matrix products. In: Proceedings of the seventeenth annual ACM-SIAM Symposium on Discrete Algorithms, SODA '06, Philadelphia. Society for Industrial and Applied Mathematics, pp 880–889
11. Childs A, Goldstone J (2004) Spatial search by quantum walk. Phys Rev A 70:022314
12. Childs AM, Kothari R (2012) Quantum query complexity of minor-closed graph properties. SIAM J Comput 41(6):1426–1450
13. Childs AM, Cleve R, Deotto E, Farhi E, Gutmann S, Spielman DA (2003) Exponential algorithmic speedup by a quantum walk. In: Proceedings of the thirty-fifth annual ACM Symposium on Theory of Computing, STOC '03, New York. ACM, pp 59–68
14. Cleve R, Ekert A, Macchiavello C, Mosca M (1998) Quantum algorithms revisited. Proc R Soc A Math Phys Eng Sci 454(1969):339–354
15. Farhi E, Gutmann S (1998) Quantum computation and decision trees. Phys Rev A 58:915–928
16. Jeffery S, Kothari R, Magniez F (2012) Improving quantum query complexity of Boolean Matrix Multiplication using Graph Collision. In: Czumaj A, Mehlhorn K, Pitts A, Wattenhofer R (eds) Automata, Languages, and Programming. Volume 7391 of Lecture Notes in Computer Science. Springer, Berlin/Heidelberg, pp 522–532
17. Jeffery S, Kothari R, Magniez F (2013) Nested quantum walks with quantum data structures. In: Proceedings of the twenty-fourth annual ACM-SIAM Symposium on Discrete Algorithms, SODA '13. SIAM, Philadelphia, pp 1474–1485
18. Jerrum M, Sinclair A (1997) The Markov Chain Monte Carlo method: an approach to approximate counting and integration. In: Hochbaum DS (ed) Approximation algorithms for NP-hard problems. PWS Publishing Co., Boston, pp 482–520
19. Kempe J (2005) Discrete quantum walks hit exponentially faster. Probab Theory Relat Fields 133(2):215–235
20. Kendon V, Tregenna B (2003) Decoherence can be useful in quantum walks. Phys Rev A 67:042315
21. Kitaev A (1995) Quantum measurements and the Abelian stabilizer problem. Technical report. quant-ph/9511026, arXiv.org
22. Kothari R (2014) Efficient algorithms in quantum query complexity. PhD thesis, University of Waterloo, Waterloo
23. Krovi H, Magniez F, Ozols M, Roland J (2014) Quantum walks can find a marked element on any graph. Technical report. arXiv:1002.2419v2, arXiv.org
24. Le Gall F (2012) Improved output-sensitive quantum algorithms for boolean matrix multiplication. In: Proceedings of the twenty-third annual ACM-SIAM Symposium on Discrete Algorithms, SODA '12. SIAM, Philadelphia, pp 1464–1476
25. Le Gall F (2014) Improved quantum algorithm for triangle finding via combinatorial arguments. In: Proceedings of the 55th annual IEEE Symposium on Foundations of Computer Science, Los Alamitos, 18–21 Oct 2014. IEEE Computer Society Press, pp 216–225
26. Lee T, Magniez F, Santha M (2013) Improved quantum query algorithms for triangle finding and associativity testing. In: Proceedings of the twenty-fourth annual ACM-SIAM Symposium on Discrete Algorithms, SODA '13. SIAM, Philadelphia, pp 1486–1502
27. Magniez F, Nayak A (2007) Quantum complexity of testing group commutativity. Algorithmica 48(3):221–232
28. Magniez F, Santha M, Szegedy M (2007) Quantum algorithms for the triangle problem. SIAM J Comput 37(2):413–424
29. Magniez F, Nayak A, Roland J, Santha M (2011) Search via quantum walk. SIAM J Comput 40:142–164
30. Magniez F, Nayak A, Richter PC, Santha M (2012) On the hitting times of quantum versus random walks. Algorithmica 63(1):91–116
31. Moore C, Russell A (2002) Quantum walks on the hypercube. In: Rolim JDP, Vadhan S (eds) Randomization and Approximation Techniques in Computer Science. Volume 2483 of Lecture Notes in Computer Science. Springer, Berlin/Heidelberg, pp 164–178
32. Nayak A, Vishwanath A (2000) Quantum walk on the line. Technical report. quant-ph/0010117, arXiv.org.
33. Richter P (2007) Quantum speedup of classical mixing processes. Phys Rev A 76:042306
34. Shenvi N, Kempe J, Birgitta Whaley K (2003) Quantum random-walk search algorithm. Phys Rev A 67:052307
35. Szegedy M (2004) Quantum speed-up of markov chain based algorithms. In: Proceedings of the 45th annual IEEE Symposium on Foundations of Computer Science. IEEE Computer Society, Los Alamitos, pp 32–41

36. Tulsi A (2008) Faster quantum walk algorithm for the two dimensional spatial search. Phys Rev A 78:012310
37. Williams VV, Williams R (2010) Subcubic equivalences between path, matrix and triangle problems. In: Proceedings of the 51st annual IEEE Symposium on Foundations of Computer Science, FOCS '10, Washington. IEEE Computer Society, pp 645–654

Quantum Approximation of the Jones Polynomial

Zeph Landau
Department of Computer Science, University of California, Berkelely, CA, USA

Keywords

AJL algorithm

Years and Authors of Summarized Original Work

2005; Aharonov, Jones, Landau

Problem Definition

A knot invariant is a function on knots (or links – i.e., circles embedded in R^3) which is invariant under isotopy of the knot, i.e., it does not change under stretching, moving, tangling, etc. (cutting the knot is not allowed). In low dimensional topology, the discovery and use of *knot invariants* is of central importance. In 1984, Jones [12] discovered a new knot invariant, now called the Jones polynomial $V_L(t)$, which is a Laurent polynomial in $\sqrt{t}$ with integer coefficients and which is an invariant of the link L. In addition to the important role it has played in low dimensional topology, the Jones polynomial has found applications in numerous fields, from *DNA* recombination [16] to statistical physics [20].

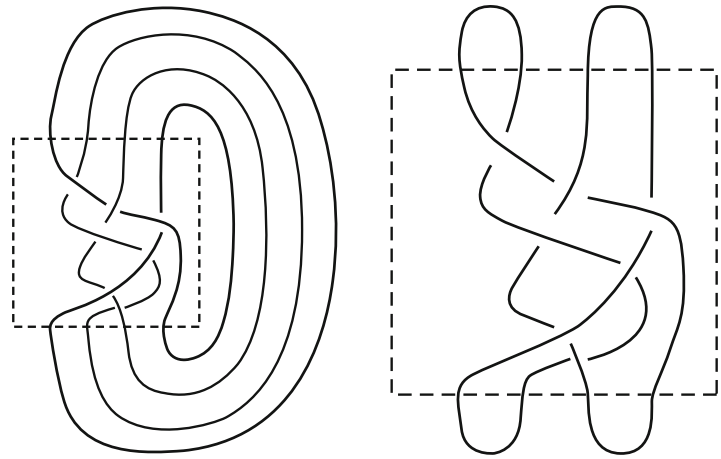
From the moment of the discovery of the Jones polynomial, the question of how hard it is to compute became important. There is a very simple inductive algorithm (essentially due to Conway [5]) to compute it by changing crossings in a link diagram, but, naively applied, this takes exponential time in the number of crossings. It was shown [11] that the computation of $V_L(t)$ is #P-hard for all but a few values of t where $V_L(t)$ has an elementary interpretation. Thus, a polynomial time algorithm for computing $V_L(t)$ for any value of t other than those elementary ones is unlikely. Of course, the #P-hardness of the problem does not rule out the possibility of good approximations. Still, the best classical algorithms to approximate the Jones polynomial at all but trivial values are exponential. Simply stated, the problem becomes:

Problem 1 For what values of t and for what level of approximation can the Jones polynomial $V_L(t)$ be approximated in time polynomial in the number of crossings and links of the link L?

Key Results

As mentioned above, exact computation of the Jones polynomial for most t is #P-hard, and the best known classical algorithms to approximate the Jones polynomial are exponential. The key results described here consider the above problem in the context of quantum rather than classical computation.

The results concern the approximation of links that are given as closures of braids. (All links can be described this way.) Briefly, a braid of n strands and m crossings is described pictorially by n strands hanging alongside each other, with m crossings, each of two adjacent strands. A braid B may be "closed" to form a link by tying its ends together in a variety of ways, two of which are the *trace closure* (denoted by B^{tr}) which joins the ith strand from the top right to the ith strand from the bottom right (for each i) and the *plat closure* (denoted by B^{pl}) which is defined only for braids with an even number of strands by connecting pairs of adjacent strands (beginning at the rightmost strand) on both the top and bottom.



Quantum Approximation of the Jones Polynomial, Fig. 1 The trace closure (*left*) and plat closure (*right*) of the same 4-strand braid

Examples of the trace and plat closure of the same 4-strand braid are given in Fig. 1.

For such braids, the following results have been shown by Aharonov, Jones, and Landau:

Theorem 1 ([2]) *For a given braid B in B_n with m crossings and a given integer k, there is a quantum algorithm which with probability $1 - c^{\Omega(n+m+k)}$ outputs a complex number r with $|r - V_{B^{\text{tr}}}(e^{2\pi i/k})| < \epsilon d^{n-1}$ where $d = 2\cos(\pi/k)$ and ϵ is inverse polynomial in n, k, m, using time that is polynomial in n, m, k.*

Theorem 2 ([2]) *For a given braid B in B_n with m crossings and a given integer k, there is a quantum algorithm which with probability $1 - c^{\Omega(n+m+k)}$ outputs a complex number r with $|r - V_{B^{\text{pl}}}(e^{2\pi i/k})| < \epsilon d^{n-1}$ where $d = 2\cos(\pi/k)$ and ϵ is inverse polynomial in n, k, m, using time that is polynomial in n, m, k.*

The original connection between quantum computation and the Jones polynomial was made earlier in the series of papers [6–9]. A model of quantum computation based on Topological Quantum Field Theory (*TQFT*) and Chern-Simons theory was defined in [6, 9], and Kitaev, Larsen, Freedman, and Wang showed that this model is polynomially equivalent in computational power to the standard quantum computation model in [7, 8]. These results, combined with a deep connection between *TQFT* and the value of the Jones polynomial at particular roots of unity discovered by Witten 13 years earlier [18], implicitly implied (without explicitly formulating) an efficient quantum algorithm for the approximation of the Jones polynomial at the value $e^{2\pi i/5}$.

The approximation given by the above algorithms are additive, namely, the result lies in a given window, whose size is independent of the actual value being approximated. The formulation of this kind of additive approximation was given in [4]; this is much weaker than a multiplicative approximation, which is what one might desire (again, see discussion in [4]). One might wonder if under such weak requirements, the problem remains meaningful at all. It turns out that, in fact, this additive approximation problem is hard for quantum computation, a result originally shown by Freedman, Kitaev, and Wang:

Theorem 3 (Adapted from [8]) *The problem of approximating the Jones polynomial of the plat closure of a braid at $e^{2\pi i/k}$ for constant k, to within the accuracy given in Theorem 2, is BQP-hard.*

A different proof of this result was given in [19], and the result was strengthened by Aharonov and Arad [1] to any k which is polynomial in the size of the input, namely, for all the plat closure cases for which the algorithm is polynomial in the size of the braid.

Understanding the Algorithm

The structure of the solution described by Theorems 1 and 2 consists of four steps:

1. *Mapping the Jones polynomial computation to a computation in the Temperley-Lieb algebra.* There exists a homomorphism of the braid group inside the so-called Temperley-Lieb algebra (this homomorphism was the connection that led to the original discovery of the Jones polynomial in [12]). Using this homomorphism, the computation of the Jones polynomial of either the plat or trace closure of a braid can be mapped to the computation of a particular linear functional (called the Markov trace) of the image of the braid in the Temperley-Lieb algebra (for an essential understanding of a geometrical picture of the Temperley-Lieb algebra, see [14]).
2. *Mapping the Temperley-Lieb algebra calculation into a linear algebra calculation.* Using a representation of the Temperley-Lieb algebra, called the path model representation, the computation in step 1 is shown to be equal to a particular weighted trace of the matrix corresponding to the Temperley-Lieb algebra element coming from the original braid.
3. *Choosing the parameter t corresponding to unitary matrices.* The matrix in step 2 is a product of basic matrices corresponding to individual crossings in the braid group; an important characteristic of these basic matrices is that they have a local structure. In addition, by choosing the values of t as in Theorems 1 and 2, the matrices corresponding to individual crossings become unitary. The result is that the original problem has been turned into a weighted trace calculation of a matrix formed from a product of local unitary matrices – a problem well suited to a quantum computer.
4. *Implementing the quantum algorithm.* Finally the weighted trace calculation of a matrix described in step 3 is formally encoded into a calculation involving local unitary matrices and qubits.

A nice exposition of the algorithm is given in [15].

Applications

Since the publication [2], a number of interesting results have ensued investigating the possibility of quantum algorithms for other combinatorial/topological questions. Quantum algorithms have been developed for the case of the HOMFLY-PT two-variable polynomial of the trace closure of a braid at certain pairs of values [19]. (This entry also extends the results of [2] to a class of more generalized braid closures; it is recommended reading as a complement to [2] or [15] as it gives the representation theory of the Jones-Wenzl representations, thus putting the path model representation of the Temperley-Lieb algebra in a more general context.) A quantum algorithm for the colored Jones polynomial is given in [10].

Significant progress was made on the question of approximating the partition function of the Tutte polynomial of a graph [3]. This polynomial, at various parameters, captures important combinatorial features of the graph. Intimately associated to the Tutte polynomial is the Potts model, a model originating in statistical physics as a generalization of the Ising model to more than 2 states [17,20]; approximating the partition function of the Tutte polynomial of a graph is a very important question in statistical physics. The work of [3] develops a quantum algorithm for additive approximation of the Tutte polynomial for all planar graphs at all points in the Tutte plane and shows that for a significant set of these points (though not those corresponding to the Potts model) the problem of approximating is a complete problem for a quantum computer. Unlike previous results, these results use non-unitary representations.

Open Problems

There remain many unanswered questions related to the computation of the Jones polynomial from both a classical and quantum computational point of view.

From a classical computation point of view, the originally stated Problem 1 remains

wide open for all but trivial choices of t. A result as strong as Theorem 2 for a classical computer seems unlikely since it would imply (via Theorem 3) that classical computation is as strong as quantum computation. A result by Jordan and Shor [13] shows that the approximation given in Theorem 1 solves a complete problem for a presumed (but not proven) weaker quantum model called the one-clean-qubit model. Since this model seems weaker than the full quantum computation model, a classical result as strong as Theorem 1 for the trace closure of a braid is perhaps in the realm of possibility.

From a quantum computational point of view, various open directions seem worthy of pursuit. Most of the quantum algorithms known as of the writing of this entry are based on the quantum Fourier transform and solve problems which are algebraic and number theoretical in nature. Arguably, the greatest challenge in the field of quantum computation (together with the physical realization of large scale quantum computers) is the design of new quantum algorithms based on substantially different techniques. The quantum algorithm to approximate the Jones polynomial is significantly different from the known quantum algorithms in that it solves a problem which is combinatorial in nature, and it does so without using the Fourier transform. These observations suggest investigating the possibility of quantum algorithms for other combinatorial/topological questions. Indeed, the results described in the applications section above address questions of this type. Of particular interest would be progress beyond [3] in the direction of the Potts model, specifically either showing that the approximation given in [3] is non-trivial or providing a different non-trivial algorithm.

Cross-References

- ▸ Fault-Tolerant Quantum Computation
- ▸ Quantum Error Correction

Recommended Reading

1. Aharonov D, Arad I (2006) The BQP-hardness of approximating the Jones Polynomial. Arxiv: quant-ph/0605181
2. Aharonov D, Jones V, Landau Z (2006) A polynomial quantum algorithm for approximating the Jones polynomial. In: Proceedings of the 38th ACM symposium on theory of computing (STOC), Seattle. Arxiv:quant-ph/0511096
3. Aharonov D, Arad I, Eban E, Landau Z (2007) Polynomial quantum algorithms for additive approximations of the Potts model and other points of the Tutte plane. Arxiv:quant-ph/0702008
4. Bordewich M, Freedman M, Lovasz L, Welsh D (2005) Approximate counting and quantum computation. Comb Prob Comput 14(5–6):737–754
5. Conway JH (1970) An enumeration of knots and links, and some of their algebraic properties. In: Computational problems in abstract algebra (Proc. Conf., Oxford, 1967). Pergamon Press, New York, pp 329–358
6. Freedman M (1998) P/NP and the quantum field computer. Proc Natl Acad Sci USA 95:98–101
7. Freedman MH, Kitaev A, Wang Z (2002) Simulation of topological field theories by quantum computers. Commun Math Phys 227:587–603
8. Freedman MH, Kitaev A, Wang Z (2002) A modular Functor which is universal for quantum computation. Commun Math Phys 227(3):605–622
9. Freedman M, Kitaev A, Larsen M, Wang Z (2003) Topological quantum computation. Mathematical challenges of the 21st century. (Los Angeles, CA, 2000). Bull Am Math Soc (NS) 40(1):31–38
10. Garnerone S, Marzuoli A, Rasetti M (2006) An efficient quantum algorithm for colored Jones polynomials. ArXiv.org:quant-ph/0606167
11. Jaeger F, Vertigan D, Welsh D (1990) On the computational complexity of the Jones and Tutte polynomials. Math Proc Camb Philos Soc 108(1):35–53
12. Jones VFR (1985) A polynomial invariant for knots via von Neumann algebras. Bull Am Math Soc 12(1):103–111
13. Jordan S, Shor P (2007) Estimating Jones polynomials is a complete problem for one clean qubit. http://arxiv.org/abs/0707.2831
14. Kauffman L (1987) State models and the Jones polynomial. Topology 26:395–407
15. Kauffman L, Lomonaco S (2006) Topological quantum computing and the Jones polynomial. ArXiv.org:quant-ph/0605004
16. Podtelezhnikov A, Cozzarelli N, Vologodskii A (1999) Equilibrium distributions of topological states in circular DNA: interplay of supercoiling and knotting (English. English summary). Proc Natl Acad Sci USA 96(23):12974–129

17. Potts R (1952) Some generalized order – disorder transformations. Proc Camb Philos Soc 48:106–109
18. Witten E (1989) Quantum field theory and the Jones polynomial. Commun Math Phys 121(3):351–399
19. Wocjan P, Yard J (2008) The Jones polynomial: quantum algorithms and applications in quantum complexity theory. Quantum Inf Comput 8(1 & 2):147–180. ArXiv.org:quant-ph/0603069 (2006)
20. Wu FY (1992) Knot theory and statistical mechanics. Rev Mod Phys 64(4):1099–1131

Quantum Dense Coding

Barbara M. Terhal
JARA Institute for Quantum Information,
RWTH Aachen University, Aachen, Germany

Keywords

Dense coding; Superdense coding

Years and Authors of Summarized Original Work

1992; Bennett, Wiesner

Problem Definition

Quantum information theory distinguishes classical bits from quantum bits or qubits. The quantum state of n qubits is represented by a complex vector in $(\mathbb{C}^2)^{\otimes n}$, where $(\mathbb{C}^2)^{\otimes n}$ is the tensor product of n 2-dimensional complex vector spaces. Classical n-bit strings form a basis for the vector space $(\mathbb{C}^2)^{\otimes n}$. Column vectors in $(\mathbb{C}^2)^{\otimes n}$ are denoted as $|\psi\rangle$ and row vectors are denoted as $|\psi\rangle^\dagger = |\psi\rangle^{*T} \equiv \langle\psi|$. The complex inner product between vectors $|\psi\rangle$ and $|\phi\rangle$ is conveniently written as $\langle\psi|\phi\rangle$.

Entangled quantum states $|\psi\rangle \in (\mathbb{C}^2)^{\otimes n}$ are those quantum states that cannot be written as a product of some vectors $|\psi_i\rangle \in \mathbb{C}^2$, that is, $|\psi\rangle \neq \bigotimes_i |\psi_i\rangle$. The Bell states are four orthogonal (maximally) entangled states defined as

$$|\Psi_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle),$$
$$|\Psi_{10}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle),$$
$$|\Psi_{01}\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle),$$
$$|\Psi_{11}\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle).$$

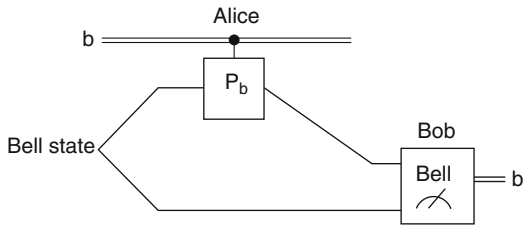
The Pauli matrices X, Y, and Z are three unitary, Hermitian 2×2 matrices. They are defined as $X = |0\rangle\langle 1| + |1\rangle\langle 0|$, $Z = |0\rangle\langle 0| - |1\rangle\langle 1|$ and $Y = iXZ$.

Quantum states can evolve dynamically under inner product preserving unitary operations $U(U^{-1} = U^\dagger)$. Quantum information can be mapped onto observable classical information through the formalism of quantum measurements. In a quantum measurement on a state $|\psi\rangle$ in $(\mathbb{C}^2)^{\otimes n}$, a basis $\{|x\rangle\}$ in $(\mathbb{C}^2)^{\otimes n}$ is chosen. This basis is made observable through an interaction of the qubits with a macroscopic measurement system. A basis vector x is thus observed with probability $\mathbb{P}(x) = |\langle x|\psi\rangle|^2$.

Quantum information theory or more narrowly quantum Shannon theory is concerned with protocols which enable distant parties to efficiently transmit quantum or classical information, possibly aided by the sharing of quantum entanglement between the parties. For a detailed introduction to quantum information theory, see the book by Nielsen and Chuang [12].

Key Results

Superdense coding [3] is the protocol in which two classical bits of information are sent from sender Alice to receiver Bob. This is accomplished by sharing a Bell state $|\Psi_{00}\rangle_{AB}$ between Alice and Bob and the transmission of one qubit. The protocol is illustrated in Fig. 1. Given two bits b_1 and b_2, Alice performs the following unitary transformation on her half of the Bell state:



Quantum Dense Coding, Fig. 1 Dense coding. Alice and Bob use a shared Bell state to transmit two classical bits $b = (b_1, b_2)$ by sending one qubit. *Double lines* are classical bits and *single lines* represent quantum bits

$$P_{b_1 b_2} \otimes I_{\mathrm{B}} |\Psi_{00}\rangle = |\Psi_{b_1 b_2}\rangle, \qquad (1)$$

i.e., one of the four Bell states. Here $P_{00} = I$, $P_{01} = X$, $P_{10} = Z$, and $P_{11} = XZ = -iY$. Alice then sends her qubit to Bob. This allows Bob to do a measurement in the Bell basis. He distinguishes the four states $|\Psi_{b1b2}\rangle$ and learns the value of the two bits b_1 and b_2.

The protocol demonstrates the interplay between classical information and quantum information. No information can be communicated by merely sharing an entangled state such as $|\Psi_{00}\rangle$ without the actual transmission of physical information carriers. On the other hand, it is a consequence of Holevo's theorem [10] that one qubit can encode at most one classical bit of information. The protocol of dense coding shows that the two resources of entanglement and qubit transmission *combined* give rise to a *superdense coding* of classical information. Dense coding is thus captured by the following resource inequality:

$$1 \text{ ebit} + 1 \text{ qubit} \geq 2 \text{ cbits}. \qquad (2)$$

In words, one bit of quantum entanglement (one ebit) in combination with the transmission of one qubit is sufficient for the transmission of two classical bits or cbits.

Dense coding can be generalized to the encoding of continuous variables, namely, the encoding of quadrature variables (x, p) of an electromagnetic field into one half of a two-mode squeezed state [2]. Such a two-mode squeezed state approximates the two-mode EPR state – in which both quadrature variables are perfectly correlated, i.e., $\mathrm{x}_1 = \mathrm{x}_2$ and $\mathrm{p}_1 = -\mathrm{p}_2$ – in the limit of large squeezing. The authors in [2] show that the information transmission capacity through the EPR state is, in the limit of large squeezing, twice that of a direct encoding using a single transmitted mode. The scheme thus exemplifies the notion of dense coding through the use of quantum entanglement.

Quantum teleportation [4] is a protocol that is dual to dense coding. In quantum teleportation, 1 ebit (a Bell state) is used in conjunction with the transmission of two classical bits to send one qubit from Alice to Bob. Thus, the resource relation for quantum teleportation is

$$1 \text{ ebit} + 2 \text{ cbits} \geq 1 \text{ qubit}. \qquad (3)$$

The relation with quantum teleportation allows one to argue that dense coding is optimal. It is not possible to encode $2k$ classical bits in less than $m < k$ quantum bits even in the presence of shared quantum entanglement. Let us assume the opposite and obtain a contradiction. One uses quantum teleportation to convert the transmission of k quantum bits into the transmission of $2k$ classical bits. Then one can use the assumed superdense coding scheme to encode these $2k$ bits into $m < k$ qubits. As a result one can send k quantum bits by effectively transmitting $m < k$ quantum bits (and sharing quantum entanglement) which is known to be impossible.

Applications

Harrow [8] has introduced the notion of a coherent bit or cobit. The notion of a cobit is useful in understanding resource relations and trade-offs between quantum and classical information. The noiseless transmission of a qubit from Alice to Bob can be viewed as the linear map $S_q : |x\rangle_{\mathrm{A}} \rightarrow |x\rangle_{\mathrm{B}}$ for a set of basis states $\{|x\rangle\}$. The transmission of a classical bit can be viewed as the linear map $S_c : |x\rangle_{\mathrm{A}} \rightarrow |x\rangle_{\mathrm{B}} |x\rangle_E$ where E stands for the environment Eve. Eve's copy of every basis state $|x\rangle$ can be viewed as the output of a quantum

measurement, and thus, Bob's state is classical. The transmission of a cobit corresponds to the linear map $S_{co} : |x\rangle_A \to |x\rangle_A|x\rangle_B$. Since Alice keeps a copy of the transmitted data, Bob's state is classical. On the other hand, the cobit can also be used to generate a Bell state between Alice and Bob. Since no qubit can be transmitted via a cobit, a cobit is weaker than a qubit. A cobit is stronger than a classical bit since entanglement can be generated using a cobit.

One can define a *coherent* version of superdense coding and quantum teleportation in which measurements are replaced by unitary operations. In this version of dense coding, Bob replaces his Bell measurement by a rotation of the states $|\Psi_{b1b2}\rangle$ to the states $|b_1 b_2\rangle_B$. Since Alice keeps her input bits, the coherent protocol implements the map $|x_1 x_2\rangle_A \to |x_1 x_2\rangle_A|x_1 x_2\rangle_B$. Thus, one can strengthen the dense coding resource relation to

$$1 \text{ ebit} + 1 \text{ qubit} \geq 2 \text{ cobits}. \tag{4}$$

Similarly, the coherent execution of quantum teleportation gives rise to the modified relation 2 cobits + 1 ebit $\geq$ 1 qubit + 2 ebits. One can omit 1 ebit on both sides of the inequality by using ebits catalytically, i.e., they can be borrowed and returned at the end of the protocol. One can then combine both coherent resource inequalities and obtain a resource *equality*:

$$2 \text{ cobits} = 1 \text{ qubit} + 1 \text{ ebit}. \tag{5}$$

A different extension of dense coding is the notion of superdense coding of quantum states proposed in [9]. Instead of dense coding classical bits, the authors in [9] propose to code quantum bits *whose quantum states are known to the sender Alice*. This last restriction is usually referred to as the remote preparation of qubits, in contrast to the transmission of qubits whose states are unknown to the sender. In remote preparation of qubits, the sender Alice can use the additional knowledge about her states in the choice of encoding. In [9] it is shown that one can obtain the asymptotic resource relation

$$1 \text{ ebit} + 1 \text{ qubit} \geq 2 \text{ remotely prepared qubit(s)}. \tag{6}$$

Such relation would be impossible if the r.h.s. were replaced by 2 qubits. In that case the inequality could be used repeatedly to obtain that 1 qubit suffices for the transmission of an arbitrary number of qubits which is impossible.

The "non-oblivious" superdense coding of quantum states should be compared with the non-oblivious and asymptotic variant of quantum teleportation which was introduced in [5]. In this protocol, referred to as remote state preparation (using classical bits), the quantum teleportation inequality, Eq. (3), is tightened to

$$1 \text{ ebit} + 1 \text{ cbit} \geq 1 \text{ remotely prepared qubit(s)}. \tag{7}$$

These various resource (in)equalities and their underlying protocols can be viewed as the first in a comprehensive theory of resources inequalities. The goal of such theory [6] is to provide a unified and simplified approach to quantum Shannon theory.

Experimental Results

In [11] a partial realization of dense coding was given using polarization states of photons as qubits. The Bell state $|\Psi_{01}\rangle$ can be produced by parametric down-conversion; this state was used in the experiment as the shared entanglement between Alice and Bob. With current experimental techniques, it is not possible to carry out a low-noise measurement in the Bell basis which uniquely distinguishes the four Bell states. Thus, in [11] one of three messages, *a trit*, is encoded into the four Bell states. Using two-particle interferometry, Bob learns the value of the trit by distinguishing two of the four Bell states uniquely and obtaining a third measurement signal for the two other Bell states.

In perfect dense coding, the channel capacity is 2 bits. For the trit-scheme of [11], the ideal channel capacity is $\log 3 \approx 1.58$. Due to the noise in the operations and measurements, the authors

of [11] estimate the experimentally achieved capacity as 1.13 bits. In [1] it is shown how the presence of additional entanglement of the polarized photons in their orbital momentum degree of freedom (hyperentanglement) can assist in distinguishing all 4 Bell states in a modified Bell state analyzer. A capacity of 1.63 bits is reported.

In [13] the complete protocol of dense coding was carried out using two $^9Be^+$ ions confined to an electromagnetic trap. A qubit is formed by two internal hyperfine levels of the $^9Be^+$ ion. Single-qubit and two-qubit operations are carried out using two polarized laser beams. A single qubit measurement is performed by observing a weak/strong fluorescence of $|0\rangle$ and $|1\rangle$. The authors estimate that the noise in the unitary transformations and measurements leads to an overall error rate on the transmission of the bits b of 15 %. This results in an effective channel capacity of 1.16 bits.

In [7] dense coding was carried out using NMR spectroscopy. The two qubits were formed by the nuclear spins of ^1H and ^{13}C of chloroform molecules $^{13}CHCL_3$ in liquid solution at room temperature. The full dense coding protocol was implemented using the technique of temporal averaging and the application of coherent RF pulses; see [12] for details. The authors estimate an overall error rate on the transmission of the bits b of less than 10 %.

Cross-References

▸ Teleportation of Quantum States

Recommended Reading

1. Barreiro J, Wei Tzu-Chieh, Kwiat P (2008) Beating the channel capacity limit for linear photonic superdense coding. Nat Phys 4:282–286
2. Braunstein S, Kimble HJ (2000) Dense coding for continuous variables. Phys Rev A 61(4):042302
3. Bennett CH, Wiesner SJ (1992) Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. Phys Rev Lett 69:2881–2884
4. Bennett CH, Brassard G, Crepeau C, Jozsa R, Peres A, Wootters WK (1993) Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. Phys Rev Lett 70:1895–1899
5. Bennett CH, DiVincenzo DP, Smolin JA, Terhal BM, Wootters WK (2001) Remote state preparation. Phys Rev Lett 87:077902
6. Devetak I, Harrow A, Winter A (2008) A resource framework for quantum Shannon theory. IEEE Trans Inf Theory 54(10), 4587–4618
7. Fang X, Zhu X, Feng M, Mao X, Du F (2000) Experimental implementation of dense coding using nuclear magnetic resonance. Phys Rev A 61:022307
8. Harrow AW (2004) Coherent communication of classical messages. Phys Rev Lett 92:097902
9. Harrow A, Hayden P, Leung D (2004) Superdense coding of quantum states. Phys Rev Lett 92:187901
10. Holevo AS (1973) Bounds for the quantity of information transmitted by a quantum communication channel. Problemy Peredachi Informatsii 9:3–11. English translation in: Probl Inf Transm 9:177–183 (1973)
11. Mattle K, Weinfurter H, Kwiat PG, Zeilinger A (1996) Dense coding in experimental quantum communication. Phys Rev Lett 76:4656–4659
12. Nielsen MA, Chuang IL (2000) Quantum computation and quantum information. Cambridge University Press, Cambridge
13. Schaetz T, Barrett MD, Leibfried D, Chiaverini J, Britton J, Itano WM, Jost JD, Langer C, Wineland DJ (2004) Quantum dense coding with atomic qubits. Phys Rev Lett 93:040505

Quantum Error Correction

Martin Roetteler
Microsoft Research, Redmond, WA, USA

Keywords

Quantum codes; Quantum error-correcting codes; Stabilizer codes

Years and Authors of Summarized Original Work

1995; Shor

Problem Definition

A quantum system can never be seen as being completely isolated from its environment, thereby permanently causing disturbance to the state of the system. The resulting noise problem

threatens quantum computers and their great promise, namely, to provide a computational advantage over classical computers for certain problems (see also the cross-references in the section "Cross-References"). Quantum noise is usually modeled by the notion of a *quantum channel* which generalizes the classical case and, in particular, includes scenarios for communication (space) and storage (time) of quantum information. For more information about quantum channels and quantum information in general, see [19]. A basic channel is the quantum mechanical analog of the classical binary symmetric channel [17]. This quantum channel is called the *depolarizing channel* and depends on a real parameter $p \in [0, 1]$. Its effect is to randomly apply one of the Pauli spin matrices X, Y, and Z to the state of the system, mapping a quantum state ρ of one qubit to $(1 - p)\rho + p/3(X\rho X + Y\rho Y + Z\rho Z)$. It should be noted that it is always possible to map any quantum channel to a depolarizing channel by twirling operations. The basic problem of quantum error correction is to devise a mechanism that allows to recover quantum information that has been sent through a quantum channel, in particular the depolarizing channel.

Key Results

For a long time, it was not known whether it would be possible to protect quantum information against noise. Even some indication in the form of the no-cloning theorem was put forward to support the view that it might be impossible. The no-cloning theorem essentially says that an unknown quantum state cannot be copied perfectly. This dashes hopes that, similar to the classical case, a simple triple-replication and majority voting mechanism may be used in the quantum case as well. Therefore, it came as a surprise when Shor [20] found a quantum code which encodes one qubit into nine qubits in such a way that the resulting state has the ability to be protected against arbitrary single-qubit errors on each of these nine qubits. The idea is to use a concatenation of two threefold repetition codes. One of them protects against bit-flip errors while the other protects against phase-flip errors. The quantum code is a two-dimensional subspace of the 2^9 dimensional Hibert space $(\mathbb{C}^2)^{\otimes 9}$. Two orthogonal basis vectors of this space are identified with the logical 0 and 1 states, respectively, usually called $|0\rangle$ and $|1\rangle$. Explicitly, the code is given by

$$|\underline{0}\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle),$$

$$|\underline{1}\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle) \otimes (|000\rangle - |111\rangle) \otimes (|000\rangle + |111\rangle).$$

The state $\alpha|0\rangle + \beta|1\rangle$ of one qubit is encoded to the state $\alpha|\underline{0}\rangle + \beta|\underline{1}\rangle$ of the nine-qubit system. The reason why this code can correct one arbitrary quantum error is as follows.

First, suppose that a bit-flip error has happened, which in quantum mechanical notation is given by the operator X. Then a majority vote of each block of three qubits 1–3, 4–6, and 7–9 can be computed and the bit flip can be corrected. To correct against phase-flip errors, which are given by the operator Z, the fact is used that the code can be written as $|\underline{0}\rangle = |+++\rangle + |---\rangle$, $|\underline{1}\rangle = |+++\rangle - |---\rangle$, where $|\pm\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$. By measuring each block of three in the basis $\{|+\rangle, |-\rangle\}$, the majority of the phase flips can be detected and one phase-flip error can be corrected. Similarly, it can be shown that Y, which is a combination of a bit flip and a phase flip, can be corrected.

Discretization of Noise

Even though the above procedure seemingly only takes care of bit-flips and phase-flip errors, it actually is true that an *arbitrary* error affecting a single qubit out of the nine qubits can be corrected. In particular, and perhaps surprisingly, this is also the case if one of the nine qubits is completely destroyed. The linearity of quantum mechanics allows this method to work. Linearity implies that whenever operators A and B can be

corrected, so can their sum $A + B$ [8, 20, 22]. Since the (finite) set $\{1_2, X, Y, Z\}$ forms a vector space basis for the (continuous) set of all one-qubit errors, the nine-qubit code can correct an arbitrary single-qubit error.

Syndrome Decoding and the Need for Fresh Ancillas

A way to do the majority vote quantum-mechanically is to introduce two new qubits (also called ancillas) that are initialized in $|0\rangle$. Then, the results of the two parity checks for the repetition code of length three can be computed into these two ancillas. This syndrome computation for the repetition code can be done using the so-called controlled not (CNOT) gates [19] and Hadamard gates. After this, the qubits holding the syndrome will factor out (i.e., they have no influence on future superpositions or interferences of the computational qubits) and can be discarded. Quantum error correction demands a large supply of fresh qubits for the syndrome computations which have to be initialized in a state $|0\rangle$. The preparation of many such states is required to fuel active quantum error-correcting cycles, in which syndrome measurements have to be applied repeatedly. This poses great challenges to any concrete physical realization of quantum error-correcting codes.

Conditions for General Quantum Codes

Soon after the discovery of the first quantum code, general conditions required for the existence of codes, which protect quantum systems against noise, were sought after. Here the noise is modeled by a general quantum channel, given by a set of error operators E_i. The Knill-Laflamme conditions [13] yield such a characterization. Let C be the code subspace and let P_C be an orthogonal projector onto C. Then the existence of a recovery operation for the channel with error operators E_i is equivalent to the equation

$$P_C E_i^\dagger E_j P_C = \lambda_{i,j} P_C,$$

for all i and j, where $\lambda_{i,j}$ are some complex constants. This recently has been extended to the more general framework of subsystem codes (also called operator quantum error-correcting codes) [16].

Constructing Quantum Codes

The problem of deriving general constructions of quantum codes was addressed in a series of groundbreaking papers by several research groups in the mid-1990s. Techniques were developed which allow classical coding theory to be imported to an extent that is enough to provide many families of quantum codes with excellent error correction properties.

The IBM group [3] investigated quantum channels, placed bounds on the quantum channels' capacities, and showed that for some channels, it is possible to compute the capacity (such as for the quantum erasure channel). Furthermore, they showed the existence of a five-qubit quantum code that can correct an arbitrary error, thereby being much more efficient than Shor's code. Around the same time, Calderbank and Shor [4] and Steane [21] found a construction of quantum codes from any pair C_1, C_2 of classical linear codes satisfying $C_2^\perp \subseteq C_1$. Named after their inventors, these codes are known as CSS codes.

The AT&T group [5] found a general way of defining a quantum code. Whenever a classical code over the finite field $\mathbb{F}_4$ exists that is additively closed and self-orthogonal with respect to the Hermitian inner product, they were able to find even more examples of codes. Independently, D. Gottesman [8,9] developed the theory of stabilizer codes. These are defined as the simultaneous eigenspaces of an abelian subgroup of the group of tensor products of Pauli matrices on several qubits. Soon after this, it was realized that the two constructions are equivalent.

A stabilizer code which encodes k qubits into n qubits and has distance d is denoted by $[n, k, d]$. It can correct up to $\lfloor (d-1)/2 \rfloor$ errors of the n qubits. The rate of the code is defined as $r = k/n$. Similar to classical codes, bounds on quantum error-correcting codes are known, i.e., the Hamming, Singleton, and linear programming bounds.

Asymptotically Good Codes

Matching the developments in classical algebraic coding theory, an interesting question deals with the existence of asymptotically good codes, i.e., families of quantum codes with parameters $[[n_i, k_i, d_i]]$, where $i \geq 0$, which have asymptotically nonvanishing rate $\lim_{i\to\infty} k_i/n_i > 0$ and nonvanishing relative distance $\lim_{i\to\infty} d_i/n_i > 0$. In [4], the existence of asymptotically good codes was established using random codes. Using algebraic geometry (Goppa) codes, it was later shown by Ashikhmin, Litsyn, and Tsfasman that there are also explicit families of asymptotically good quantum codes [2]. Currently, most constructions of quantum codes are from the abovementioned stabilizer/additive code construction, with notable exception of a few nonadditive codes and some codes which do not fit into the framework of Pauli error bases.

Applications

Besides their canonical application to protect quantum information against noise, quantum error-correcting codes have been used for other purposes as well. The Preskill/Shor proof of the security of the quantum key distribution scheme BB84 relies on an entanglement purification protocol, which in turn uses CSS codes [19]. Furthermore, quantum codes have been used for quantum secret sharing, quantum message authentication, and secure multiparty quantum computations. Properties of stabilizer codes are also germane for the theory of fault-tolerant quantum computation.

Open Problems

The literature of quantum error correction is fast growing, and the list of open problems is certainly too vast to be surveyed here in detail. The following short list is highly influenced by the preference of the author.

1. It is desirable to find quantum codes for which all stabilizer generators have low weight and which at the same time allow for efficient fault-tolerant quantum computation with the encoded data. These requirements correspond to a quantum equivalent to low-density parity check (LDPC) codes. So far only a few constructions are known, but recent progress was made by Gottesman [10] who used quantum LDPC codes to show that universal fault-tolerant quantum computing with constant overhead is possible. See also [11,15] for recent progress on quantum LDPC codes.
2. It is an open problem to find new families of quantum codes that improve on the currently best known estimates for the threshold for fault-tolerant quantum computing, in particular for codes that can be implemented on a two-dimensional fabric of qubits. An advantage might be had by using subsystem codes since they allow for simple error correction circuits. For more information about noise thresholds, see also the entry on ▸ Fault-Tolerant Quantum Computation.
3. Many quantum codes are designed for the depolarizing channel, where – roughly speaking – the error probability is improved from p to $p^{d/2}$ for a distance d code. The independence assumption underlying this model might not always be justified, and therefore, it seems imperative to consider other channels, e.g., non-Markovian local error models. Under some assumptions on the decay of the interaction strengths, threshold results for such channels have been shown [1]. However, it remains open to find constructions of good codes for non-Markovian noise and in general for noise models that are more realistic than the depolarizing channel.

Experimental Results

Active quantum error-correcting codes, such as those codes which require syndrome measurements and correction operations, as well as passive codes (i.e., codes in which the system stays in a simultaneous invariant subspace of all error operators for certain types of noise), have been

demonstrated for various physical systems. First, this was shown in nuclear magnetic resonance (NMR) experiments [14]. The three-qubit repetition code, which protects one qubit against phase-flip error Z, was then demonstrated in an ion trap for beryllium ion qubits [6].

Subsequently, architectures have been proposed [18] that would in principle allow to construct scalable quantum computers based on ion traps and concatenated coding, e.g., based on the [1, 3, 7] Steane code. In superconducting qubit systems, using an architecture that supports nine physical qubits, high gate fidelities have been reported [12]. This suggests that it might be possible in this architecture to achieve error rates that are below the threshold for the surface code, which is known to be around 1 % [7].

Data Sets

Markus Grassl maintains http://www.codetables.de, which contains tables of the best known quantum codes, some entries of which extend ([5], Table III). It also contains bounds on the minimum distance of quantum codes for given lengths and dimensions and contains information about the construction of the codes. In principle, this can be used to get explicit generator matrices (see also the following section "URL to Code").

URL to Code

The computer algebra system Magma (http://magma.maths.usyd.edu.au/magma/) has functions and data structures for defining and analyzing quantum codes. Several quantum codes are already defined in a database of quantum codes. For instance, the command *BestKnownQuantumCode(F, n, k)* returns the best known quantum code (i.e., one of the highest known minimum weight) over the field F, of length n, and dimension k. It allows the user to define new quantum codes and to study its properties such as the weight distribution, automorphism, and several predefined methods for obtaining new codes from a set of given ones.

Cross-References

- Abelian Hidden Subgroup Problem
- Fault-Tolerant Quantum Computation
- Quantum Algorithm for Element Distinctness
- Quantum Algorithm for Factoring
- Quantum Algorithm for Finding Triangles
- Quantum Algorithm for Solving Pell's Equation
- Quantum Analogues of Markov Chains
- Quantum Key Distribution
- Teleportation of Quantum States

Recommended Reading

1. Aharonov D, Kitaev A, Preskill J (2006) Fault-tolerant quantum computation with long-range correlated noise. Phys Rev Lett 96:050504
2. Ashikhmin A, Litsyn S, Tsfasman MA (2001) Asymptotically good quantum codes. Phys Rev A 63:032311
3. Bennett CH, DiVincenzo DP, Smolin JA, Wootters WK (1996) Mixed-state entanglement and quantum error correction. Phys Rev A 54:3824–3851
4. Calderbank AR, Shor PW (1996) Good quantum error-correcting codes exist. Phys Rev A 54:1098–1105
5. Calderbank AR, Rains EM, Shor PW, Sloane NJA (1998) Quantum error correction via codes over GF(4). IEEE Trans Inf Theory 44:1369–1387
6. Chiaverini J, Leibfried D, Schaetz T, Barrett MD, Blakestad RB, Britton J, Itano WM, Jost JD, Knill E, Langer C, Ozeri R, Wineland DJ (2004) Realization of quantum error correction. Nature 432:602–605
7. Fowler AG, Mariantoni M, Martinis JM, Cleland AN (2012) Surface codes: towards practical large-scale quantum computation. Phys Rev A 86:032324
8. Gottesman D (1996) Class of quantum error-correcting codes saturating the quantum Hamming bound. Phys Rev A 54:1862–1868
9. Gottesman D (1997) Stabilizer codes and quantum error correction, Ph.D. thesis, Caltech. See also: arXiv preprint quant-ph/9705052
10. Gottesman D (2013) Fault-tolerant quantum computation with constant overhead. arXiv.org preprint, arXiv:1310.2984
11. Hastings MB (2014) Decoding in hyperbolic spaces: quantum LDPC codes with linear rate and efficient error correction. Quantum Inf Comput 14:1187–1202
12. Kelly J, Barends R, Fowler AG, Megrant A, Jeffrey E, White TC, Sank D, Mutus JY, Campbell B, Chen Y, Chen Z, Chiaro B, Dunsworth A, Hoi I-C, Neill C, O'Malley PJJ, Quintana C, Roushan P, Vainsencher A, Wenner J, Cleland AN, Martinis JM (2014) State preservation by repetitive error detection in a superconducting quantum circuit. Nature 519:66–69

13. Knill E, Laflamme R (1997) Theory of quantum error-correcting codes. Phys Rev A 55:900–911
14. Knill E, Laflamme R, Martinez R, Negrevergne C (2001) Benchmarking quantum computers: the five-qubit error correcting code. Phys Rev Lett 86:5811–5814
15. Kovalev AA, Pryadko LP (2013) Fault-tolerance of "bad" quantum low-density parity check codes. Phys Rev A 87:020304(R)
16. Kribs D, Laflamme R, Poulin D (2005) Unified and generalized approach to quantum error correction. Phys Rev Lett 94(4):180501
17. MacWilliams FJ, Sloane NJA (1977) The theory of error-correcting codes. North-Holland, Amsterdam
18. Monroe C, Raussendorf R, Ruthven A, Brown KR, Maunz P, Duan L-M, Kim J (2014) Large scale modular quantum computer architecture with atomic memory and photonic interconnects. Phys Rev A 89:022317
19. Nielsen M, Chuang I (2000) Quantum computation and quantum information. Cambridge University Press, Cambridge
20. Shor PW (1995) Scheme for reducing decoherence in quantum computer memory. Phys Rev A 52:R2493–R2496
21. Steane A (1996) Error correcting codes in quantum theory. Phys Rev Lett 77:793–797
22. Steane A (1996) Multiple-particle interference and quantum error correction. Proc R Soc Lond A 452:2551–2577

Quantum Key Distribution

Renato Renner
Institute for Theoretical Physics, Zurich, Switzerland

Keywords

Quantum cryptography

Years and Authors of Summarized Original Work

1984; CH Bennett, G Brassard
1991; A Ekert

Problem Definition

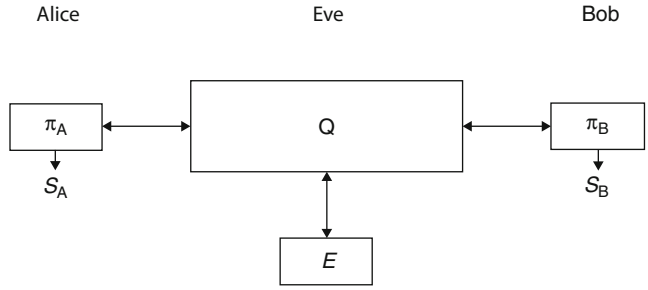
Secret keys, i.e., random bitstrings not known to an adversary, are a vital resource in cryptography (they can be used, e.g., for message encryption or authentication). The *distribution* of secret keys among distant parties, possibly only connected by insecure communication channels, is thus a fundamental cryptographic problem. *Quantum key distribution (QKD)* is a method to solve this problem using quantum communication. It relies on the fact that any attempt of an adversary to wiretap the communication would, by the laws of quantum mechanics, inevitably introduce disturbances which can be detected.

For the technical definition, consider a setting consisting of two honest parties, called *Alice* and *Bob*, as well as an adversary, *Eve*. Alice and Bob are connected by a quantum channel $\mathcal{Q}$ which might be coupled to a (quantum) system E controlled by Eve (see Fig. 1). In addition, it is assumed that Alice and Bob have some means to exchange classical messages *authentically*, that is, they can make sure that Eve is unable to (undetectably) alter classical messages during transmission. If only insecure communication channels are available, Alice and Bob can achieve this using an *authentication scheme* [19]. The scheme requires that Alice and Bob have a short initial key or at least some initial common randomness that is not entirely known to Eve [17]. This is why QKD is sometimes called *quantum key growing*.

A *QKD protocol* $\pi = (\pi_A, \pi_B)$ is a pair of algorithms for Alice and Bob, producing classical outputs S_A and S_B, respectively. S_A and S_B take values in $\mathcal{S} \cup \{\perp\}$ where $\mathcal{S}$ is called *key space* and $\perp$ is a symbol (not contained in $\mathcal{S}$) indicating that no key can be generated. A QKD protocol π with key space $\mathcal{S}$ is said to be *perfectly secure on a channel* $\mathcal{Q}$ if, after its execution using communication over $\mathcal{Q}$, the following holds:

- $S_A = S_B$;
- if $S_A \neq \perp$, then S_A and S_B are uniformly distributed on $\mathcal{S}$ and independent of the state of E.

More generally, π is said to be ε-*secure on* $\mathcal{Q}$ if it satisfies the above conditions except with probability (at most) ε. Furthermore, π is



Quantum Key Distribution, Fig. 1 A QKD protocol π consists of algorithms π_A and π_B for Alice and Bob, respectively. The algorithms communicate over a quantum channel $\mathcal{Q}$ that might be coupled to a system E controlled by an adversary. The goal is to generate identical keys S_A and S_B which are independent of E

said to be ε-*robust on* $\mathcal{Q}$ if the probability that $S_A = \perp$ is at most ε. These definitions may be extended to sets of channels $\mathcal{Q}$, i.e., one demands that the conditions hold for any member of the set.

In the standard literature on QKD, protocols are typically parametrized by some positive number k quantifying certain resources needed for its execution (e.g., the amount of communication). A protocol $\pi = (\pi_k)_{k\in\mathbb{N}}$ is said to be *secure* (*robust*) on a set of channels if there exists a sequence $(\varepsilon_k)_{k\in\mathbb{N}}$ which approaches zero exponentially fast such that π_k is ε_k-secure (ε_k-robust) on this set for any $k \in \mathbb{N}$. Moreover, if the key space of π_k is denoted by $\mathcal{S}_k$, the *key rate* of $\pi = (\pi_k)_{k\in\mathbb{N}}$ is defined by $r = \lim_{k\to\infty} \frac{l_k}{k}$ where $l_k := \log_2 |\mathcal{S}_k|$ is the key length.

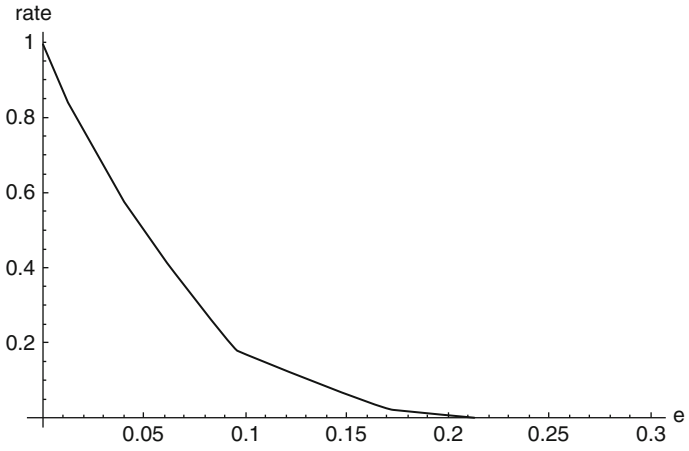
The ultimate goal is to construct QKD protocols π which are secure against general attacks, i.e., secure on the set of *all* possible channels $\mathcal{Q}$. This ensures that an adversary cannot get any information on the generated key even if she fully controls the communication between Alice and Bob. At the same time, a protocol π should be robust on a set of realistic channels, corresponding to a situation where the noise of the channel is below a given threshold and no adversary is present. Note that, in contrast to security, robustness cannot be guaranteed on the set of all possible channels. Indeed, an adversary could, for instance, interrupt the entire communication between Alice and Bob (in which case key generation is obviously impossible).

Key Results

Protocols

On the basis of the pioneering work of Wiesner [20], Bennett and Brassard, in 1984, invented QKD and proposed a first protocol, known today as the *BB84 protocol* [3]. In 1991, Ekert invented entanglement-based QKD. His protocol is commonly referred to as E91 [8] and provides an additional level of security, termed *device independence* [1, 9]. Later, in an attempt to increase the efficiency and practicability of QKD, various extensions to the BB84 and E91 protocols as well as alternative schemes have been proposed.

QKD protocols can generally be subdivided into (at least) two subprotocols. The purpose of the first, called *distribution protocol*, is to generate a *raw key pair*, i.e., a pair of correlated classical values X and Y known to Alice and Bob, respectively. In many protocols (including BB84), Alice chooses $X = (X_1, \ldots, X_k)$ at random, encodes each of the X_i into the state of a quantum particle, and then sends the k particles over the quantum channel to Bob. Upon receiving the particles, Bob applies a measurement to each of them, resulting in $Y = (Y_1, \ldots, Y_k)$. The crucial idea now is that, by virtue of the laws of quantum mechanics, the secrecy of the raw key is a function of the strength of the correlation between X and Y; in other words, the more information about the (raw) key an adversary tries to acquire, the more disturbances she introduces.



Quantum Key Distribution, Fig. 2 Key rate of an extended version of the BB84 QKD protocol depending on the maximum tolerated channel noise (measured in terms of the bit-flip probability e) [14]

This is exploited in the second subprotocol, called *distillation protocol*. Roughly speaking, Alice and Bob estimate the statistics of the raw key pair (X, Y). If the correlation between their respective parts is sufficiently strong, they use classical techniques such as *information reconciliation* (error correction) and *privacy amplification* (see [4] for the case of a classical adversary which is relevant for the analysis of security against individual attacks and [14, 16] for the quantum-mechanical case which is relevant in the context of collective and general attacks; cf. the characterization below) to turn (X, Y) into a pair (S_A, S_B) of identical and secret keys.

Key Rate as a Function of Robustness and Security

The performance (in terms of the key rate) of a QKD protocol strongly depends on the desired level of security and robustness, as illustrated in Fig. 2. (The robustness is typically measured in terms of the *maximum tolerated channel noise*, i.e., the maximum noise of a channel $\mathcal{Q}$ such that the protocol is still robust on $\mathcal{Q}$.) The results summarized below apply to protocols of the form described above where, for the analysis of robustness, it is assumed that the quantum channel $\mathcal{Q}$ connecting Alice and Bob is *memoryless* and *time invariant*, i.e., each transmission is subject to the same type of disturbances. Formally, such channels are denoted by $\mathcal{Q} = \bar{\mathcal{Q}}^{\otimes k}$ where $\bar{\mathcal{Q}}$ describes the action of the channel in a single transmission.

Security Against Individual Attacks

A QKD protocol π is said to be *secure against individual attacks* if it is secure on the set of channels $\mathcal{Q}$ of the form $\bar{\mathcal{Q}}^{\otimes k}$ under the constraint that the coupling to E is purely classical. Note that this notion of security is relatively weak. Essentially, it only captures attacks where the adversary applies identical and independent measurements to each of the particles sent over the channel.

The following statement can be derived from a classical argument due to Csiszár and Körner [6]. Let τ be a distribution subprotocol as described above, i.e., τ generates a raw key pair (X, Y). Moreover, let $\mathcal{S}$ be a set of quantum channels $\bar{\mathcal{Q}}$ suitable for τ. Then there exists a QKD protocol π (parametrized by k) consisting of k executions of the subprotocol τ followed by an appropriate distillation subprotocol such that the following holds: π is robust on $\mathcal{Q} = \bar{\mathcal{Q}}^{\otimes k}$ for any $\bar{\mathcal{Q}} \in \mathcal{S}$, is secure against individual attacks, and has key rate at least

$$r \geq \min_{\bar{\mathcal{Q}} \in \mathcal{S}} H(X|Z) - H(X|Y), \qquad (1)$$

where the conditional Shannon entropies on the r.h.s. are evaluated for the joint distribution $P^{\bar{\mathcal{Q}}}_{XYZ}$ of the raw key (X, Y) and the (classical) state Z of Eve's system E after one execution of τ on the channel $\mathcal{Q}$. Evaluating the right hand side for the BB84 protocol on a channel with bit-flip probability e shows that the rate is non-negative if $e \leq 14.6\,\%$ [10].

Security Against Collective Attacks

A QKD protocol π is said to be *secure against collective attacks* if it is secure on the set of channels $\mathcal{Q}$ of the form $\bar{\mathcal{Q}}^{\otimes k}$ with arbitrary coupling to E. This notion of security is strictly stronger than security against individual attacks, but it still relies on the assumption that an adversary does not apply joint operations to the particles sent over the channel.

As shown by Devetak and Winter [7], the above statement for individual attacks extends to collective attacks when replacing inequality (1) by

$$r \geq \min_{\bar{\mathcal{Q}} \in \mathcal{S}} S(X|E) - H(X|Y), \qquad (2)$$

where $S(X|E)$ is the conditional von Neumann entropy evaluated for the classical value X and the quantum state of E after one execution of τ on $\bar{\mathcal{Q}}$. For the standard BB84 protocol, the rate is positive as long as the bit-flip probability e of the channel satisfies $e \leq 11.0\,\%$ [18] (see Fig. 2 for a graph of the performance of an extended version of the protocol).

Security Against General Attacks

A QKD protocol π is said to be *secure against general attacks* if it is secure on the set of all channels $\mathcal{Q}$. This type of security is sometimes also called *full* or *unconditional security* as it does not rely on any assumptions on the type of attacks (as long as they are constrained to the communication channel) or the resources needed by an adversary.

The first QKD protocol to be proved secure against general attacks was the BB84 protocol. The original argument by Mayers [13] was followed by various alternative proofs. Most notably, based on a connection to the problem of entanglement purification [5] established by Lo and Chau [12], Shor and Preskill [18] presented a general argument which applies to various versions of the BB84 protocol.

Later it has been shown that, for virtually *any* QKD protocol, security against collective attacks implies security against general attacks [14, 15]. In particular, the above statement about the security of QKD protocols against collective attacks, including formula 2 for the key rate, extends to security against general attacks.

Applications

Because the notion of security described above is *composable* [16] (see [2, 14] for a general discussion of composability of QKD), the key generated by a secure QKD protocol can in principle be used within any application that requires a secret key (such as one-time pad encryption). More precisely, let $\mathcal{A}$ be a scheme which, when using a *perfect* key S (i.e., a uniformly distributed bitstring which is independent of the adversary's knowledge), has some failure probability δ (according to some arbitrary failure criterion). Then, if the perfect key S is replaced by the key generated by an ε-secure QKD protocol, the failure probability of $\mathcal{A}$ is bounded by $\delta + \varepsilon$ [14].

Experimental Results

Most known QKD protocols (including BB84 and E91) only require relatively simple quantum operations on Alice and Bob's side (e.g., preparing a two-level quantum system in a given state or measuring the state of such a system). This makes it possible to realize them with today's technology. Experimental implementations of QKD protocols usually use photons as carriers of quantum information, because they can easily be transmitted (e.g., through optical fibers or free space). A main limitation, however, is noise in the transmission, which, with increasing distance between Alice and Bob, reduces the performance of the protocol (see Fig. 2). We refer to [11] for an overview on quantum cryptography with a focus on experimental aspects.

Cross-References

▸ Quantum Error Correction
▸ Teleportation of Quantum States

Recommended Reading

1. Acín A et al (2007) Device-independent security of quantum cryptography against collective attacks. Phys Rev Lett 98:230501
2. Ben-Or M, Horodecki M, Leung DW, Mayers D, Oppenheim J (2005) The universal composable security of quantum key distribution. In: Second theory of cryptography conference TCC. Lecture notes in computer science, vol 3378. Springer, Berlin, pp 386–406. Also available at http://arxiv.org/abs/quant-ph/0409078
3. Bennett CH, Brassard G (1984) Quantum cryptography: public-key distribution and coin tossing. In: Proceedings of IEEE international conference on computers, systems and signal processing. IEEE Computer Society, Los Alamitos, pp 175–179
4. Bennett CH, Brassard G, Crépeau C, Maurer U (1995) Generalized privacy amplification. IEEE Trans Inf Theory 41(6):1915–1923
5. Bennett CH, Brassard G, Popescu S, Schumacher B, Smolin J, Wootters W (1996) Purification of noisy entanglement and faithful teleportation via noisy channels. Phys Rev Lett 76:722–726
6. Csiszár I, Körner J (1978) Broadcast channels with confidential messages. IEEE Trans Inf Theory 24:339–348
7. Devetak I, Winter A (2005) Distillation of secret key and entanglement from quantum states. Proc R Soc Lond A 461:207–235
8. Ekert AK (1991) Quantum cryptography based on Bell's theorem. Phys Rev Lett 67:661–663
9. Ekert A, Renner R (2014) The ultimate physical limits of privacy. Nature 507:443–447
10. Fuchs CA, Gisin N, Griffiths RB, Niu C, Peres A (1997) Optimal eavesdropping in quantum cryptography, I. Information bound and optimal strategy. Phys Rev A 56:1163–1172
11. Gisin N, Ribordy G, Tittel W, Zbinden H (2002) Quantum cryptography. Rev Mod Phys 74:145–195
12. Lo H-K, Chau HF (1999) Unconditional security of quantum key distribution over arbitrarily long distances. Science 283:2050–2056
13. Mayers D (1996) Quantum key distribution and string oblivious transfer in noisy channels. In: Advances in cryptology – CRYPTO '96. Lecture notes in computer science, vol 1109. Springer, Berlin, pp 343–357
14. Renner R (2005) Security of quantum key distribution. Ph.D. thesis, Swiss Federal Institute of Technology (ETH) Zurich. Also available at http://arxiv.org/abs/quant-ph/0512258
15. Renner R (2007) Symmetry of large physical systems implies independence of subsystems. Nat Phys 3:645–649
16. Renner R, König R (2005) Universally composable privacy amplification against quantum adversaries. In: Second theory of cryptography conference TCC. Lecture notes in computer science, vol 3378. Springer, Berlin, pp 407–425. Also available at http://arxiv.org/abs/quant-ph/0403133
17. Renner R, Wolf S (2003) Unconditional authenticity and privacy from an arbitrarily weak secret. In: Proceedings of Crypto 2003. Lecture notes in computers science, vol 2729. Springer, Berlin, pp 78–95
18. Shor PW, Preskill J (2000) Simple proof of security of the BB84 quantum key distribution protocol. Phys Rev Lett 85:441
19. Wegman MN, Carter JL (1981) New hash functions and their use in authentication and set equality. J Comput Syst Sci 22:265–279
20. Wiesner S (1983) Conjugate coding. Sigact News 15(1):78–88

Quantum Search

Apoorva D. Patel[1] and Lov K. Grover[2]
[1]Centre for High Energy Physics, Indian Institute of Science, Bangalore, India
[2]Bell Laboratories, Alcatel-Lucent, Murray Hill, NJ, USA

Keywords

Amplitude amplification; Grover diffusion; Quantum walk; Unsorted database; Unstructured search

Years and Authors of Summarized Original Work

1996; Grover

Problem Definition

Brief Description

The search problem can be described informally as finding an item possessing a specific property, in a given set of N items. Each item either does or does not possess the specified property, and that can be checked by a binary query. The complexity of the problem is the number of such queries required to find the desired item (also called the *target* item). The items are often collected in a

database and sorted to simplify the subsequent searches. When they are not sorted, there is no shortcut to the brute force method of checking each item one by one until the desired item is found. A familiar example of a database is a telephone directory. Its entries are sorted according to the names of persons but not according to the telephone numbers. Hence, it is easy to find the telephone number of a particular person, but difficult to find the name of a person to whom a particular telephone number belongs (i.e., a lookup is difficult when it is not in the same order in which the database is sorted).

The $\Omega(N)$ lower bound on search speed, based on inspection of one item at a time, is correct only for classical computers. Quantum computers can be in a superposition of multiple states, however, and so can inspect multiple items at the same time. There is no obvious lower bound on how fast a quantum search can be, nor is there an obvious technique faster than the brute force search. It turns out, though, that there is an efficient and optimal quantum search algorithm that requires only $O(\sqrt{N})$ queries [15].

This quantum algorithm is very different from the search on a classical computer [8]. The optimal classical strategy is to check the items one at a time in a random order, avoiding in later trials the items that have already been checked earlier. After η items have been checked from a uniform distribution, the probability that the search hasn't yet succeeded is $(1 - 1/N)(1 - 1/(N-1)) \cdots (1 - 1/(N - \eta + 1))$. For $\eta \ll N$, the success probability is therefore roughly $1 - (1 - 1/N)^{\eta} \approx \eta/N$. Increasing this success probability to $\Theta(1)$ requires the number of items checked, η, to be $\Theta(N)$.

In contrast to classical computation, quantum computation is formulated in terms of wavelike complex amplitudes, whose interference can be used to cancel undesirable components and boost the desired component. Quantum search is then analogous to the design of a multi-element antenna array, where a careful choice of phases can boost the radiation in a particular direction. The analysis of such structures is carried out using the algebra of unitary transformations, and absolute-value squares of the amplitudes give the observation probabilities. Unitary transformations include rotations and reflections (about various directions) in the space of amplitudes, as well as local phase shifts. Rotations and reflections redistribute the amplitudes and are similar to classical transformations. On the other hand, the phase shifts are uniquely quantum; they do not alter probabilities of individual components, but affect their subsequent interference pattern. The challenge of quantum computation is to find a sequence of elementary unitary operations (i.e., quantum logic gates) that solve the given computer science problem, while ensuring that the input and the output of the quantum algorithm have clear classical interpretations.

The quantum search algorithm steadily increases the amplitude of the desired item through a series of quantum operations. Starting with an initial amplitude $1/\sqrt{(N)}$, in η steps, the amplitude increases to roughly $\eta/\sqrt{N}$, and hence, the success probability (on observation of the state) increases to η^2/N. Boosting this to $\Theta(1)$ requires only $O(\sqrt{N})$ steps, approximately the square root of the number of steps required by the best classical algorithm.

The quantum search algorithm is of wide interest because of its versatility; it can be adapted to different settings in a variety of fields, giving a new class of quantum algorithms extending well beyond the search problems. Since its discovery, it has been incorporated in solutions of many quantum problems – several of them are mentioned later in this article. Even now, two decades after the algorithm's discovery, new applications and extensions keep on appearing regularly.

Formal Construction

Let the items in the set be labeled by an index $i = 1, 2, \ldots, N$. Let the binary query be represented by an oracle $f(i)$, such that $f(i) = 1$ when i represents a desired item and $f(i) = 0$ otherwise. The quantum algorithm works in an N-dimensional vector space with complex coordinates, known as the Hilbert space. We use Dirac's notation, which is standard in the literature of quantum mechanics and quantum computation. Then the items are mapped to the N orthogonal basis vectors $|i\rangle$ of the Hilbert space, and the bi-

nary query is mapped to the selective phase-shift operator defined by $U_f|i\rangle = (-1)^{f(i)}|i\rangle$. Given the binary query $f(i)$, it is easy to construct the operator U_f using an ancilla qubit.

The problem is to start from a specific initial state $|s\rangle$ and evolve to a target state satisfying $U_f|t\rangle = -|t\rangle$, by applying a sequence of unitary operations. The number of times U_f is used in the algorithm is its query complexity. This search problem is *unstructured* because nothing is known about the solution, except the information available from the oracle that can tell whether or not a specific state is the target state.

NP-complete problems can be represented as exhaustive search problems. For example, let ϕ be a 3-SAT formula on n Boolean variables. Then the search problem is to find an assignment for the variables, $i \in \{1, 2, \ldots, N = 2^n\}$, that satisfy ϕ. This example does not involve a database and so bypasses concerns regarding how the items are stored in a physical memory device and the spatial relationship among them.

Key Results

Grover [15] showed that there indeed exists a quantum search algorithm that provides a square-root speedup over the optimal classical randomized algorithm. The algorithm has its simplest form when there is only one target item. Then the algorithm starts with an unbiased uniform superposition state $|s\rangle = (1/\sqrt{N}) \sum_{i=1}^{N} |i\rangle$ and performs $Q = O(\sqrt{N})$ iterations to evolve to the state $(U_D U_f)^Q |s\rangle$. Each iteration consists of two reflection operations:

1. $U_f = 1 - 2|t\rangle\langle t|$ is a reflection along $|t\rangle$. It uses the binary query to flip the sign of the amplitude of the target state.
2. $U_D = 2|s\rangle\langle s| - 1$ is reflection about $|s\rangle$. It can be carried out without any information about the target state. Since the action of $|s\rangle\langle s|$ gives the average amplitude state, U_D amounts to inversion about the average or overrelaxation.

At the end, the final state is measured in the $\{|i\rangle\}$ basis that encodes the item labels, and i is output.

There are several ways of analyzing the algorithm, and the geometric picture is perhaps the simplest. We observe that throughout the evolution, the quantum state stays in the two-dimensional subspace (of the Hilbert space) spanned by $|s\rangle$ and $|t\rangle$. Initially, the amplitude of the state along $|t\rangle$ is $\langle t|s\rangle = 1/\sqrt{N}$, and the angle between $|t\rangle$ and $|s\rangle$ is $\pi/2 - \theta$ with $\sin\theta = 1/\sqrt{N}$. It is a general property of linear transformations in two dimensions that a pair of reflections about two distinct axes produces a rotation, and the amount of rotation is twice the angle between the two axes. The quantum search algorithm is an alternating sequence of reflections about two different axes. Each application of the operator $U_D U_f$ rotates the quantum state from $|s\rangle$ toward $|t\rangle$ by angle 2θ. The number of iterations $\tilde{Q}$ required to exactly reach the target state is therefore given by $(2\tilde{Q} + 1)\theta = \pi/2$. In practice, we have to truncate to integer $Q = \lfloor \tilde{Q} + 0.5 \rfloor$, introducing a small error. The success probability still remains at least $\cos^2\theta = 1 - 1/N$. Asymptotically, $Q = (\pi/4)\sqrt{N}$.

The reflection about the uniform superposition state, U_D, is known as the Grover diffusion operation. When the indices are represented in binary notation, with $N = 2^n$, we have $|s\rangle = H^{\otimes n}|0\rangle^{\otimes n}$ in terms of the Hadamard operator $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. Then $U_D = H^{\otimes n} U_0 H^{\otimes n}$, with $U_0 = 2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - 1$ being the reflection about the $|0\rangle^{\otimes n}$ state, and it can be implemented using $O(n)$ qubit-level operations. In this case, the full quantum search algorithm evolves the state $|0\rangle^{\otimes n}$ to the state $(H^{\otimes n} U_0 H^{\otimes n} U_f)^Q H^{\otimes n}|0\rangle^{\otimes n}$.

When there are M target items, instead of just one, all that is required is to replace $|t\rangle$ by $\sum_{j=1}^{M} |t_j\rangle/\sqrt{M}$ in the algorithm. The final measurement then yields one of the target items after $O\left(\sqrt{N/M}\right)$ queries. Thus, we have [15]:

Theorem 1 (Grover search) *There is a quantum black-box unstructured search algorithm*

with success probability $\Theta(1)$*, which finds any one of the* M *target items in a set of* N *items, using* $O\left(\sqrt{N/M}\right)$ *queries.*

This algorithm has several noteworthy properties:

- The algorithm is optimal. It saturates the $\Omega\left(\sqrt{N}\right)$ lower bound [5] on the number of queries required for an unstructured quantum search. The evolution from $|s\rangle$ to $|t\rangle$ follows the shortest geodesic route in the Hilbert space at constant speed. A variational analysis shows that the algorithm cannot be improved by even a single query [43].
- The best classical search algorithm has to walk randomly through all the items, while the quantum search algorithm performs a directed walk in the Hilbert space. The square-root speedup of the quantum search algorithm can therefore be understood as the well-known result that directed walk provides a square-root speedup over random walk while covering the same distance.
- The algorithm can be looked upon as evolution of the quantum state from $|s\rangle$ to $|t\rangle$, governed by a Hamiltonian containing two terms, $|t\rangle\langle t|$ and $|s\rangle\langle s|$. The former represents a potential energy attracting the state toward $|t\rangle$, and the latter represents a kinetic energy diffusing the state throughout the Hilbert space. The algorithm is then the discrete Trotter's formula, generated by exponentiating the two terms in the Hamiltonian [17].
- Grover search does not require the full power of quantum dynamics and can be implemented using any system that obeys the superposition principle. Explicit examples using classical waves in the form of coupled oscillators have been constructed [20, 29]. In these mechanical systems, the role of the uniform superposition state is played by the center-of-mass mode, and the search problem becomes the energy focusing problem. The classical wave implementation requires the same number of queries as the quantum algorithm. The difference is that to represent N items, we need N wave modes but only $n = \log_2 N$ qubits.
- Grover search finds with certainty a single target state out of four possibilities using a single binary query, i.e., $Q = 1$ for $N = 4$. The best classical Boolean algorithm can distinguish only two items with a single binary query, and so it needs two binary queries to carry out the same task. When the query can be factored into subqueries, e.g., the item label is searched for one digit at a time and not as a whole, the best quantum (or wave) arrangement for a database is a quaternary tree. Then every subquery reduces the search space by a factor of 4, which is a factor-of-2 advantage over the classically optimal binary tree. An additional advantage following from commutativity of superposition is that, unlike the classical case, the quantum tree does not require sorting of the database [28].
- The quantum search algorithm is robust against changes in the initial state and the operators, in sharp contrast to many other quantum processes that are highly sensitive to errors. The initial state and the diffusion operator are related in Grover search, but can be separated in a more general context [2, 40]. Let $\tilde{U}_D$ be the modified diffusion operator with $|s\rangle$ as an eigenstate with eigenvalue 1, i.e., the diffusion is translationally invariant. The algorithm then succeeds with $\Theta(1)$ probability, provided $\alpha = |\langle t|s\rangle|$ as well as the angular spectral gap of $\tilde{U}_D$ in the vicinity of identity (say $\tilde{\theta}$) are bounded away from zero. The number of queries required is $O(B^3/\alpha)$, where B^2 is related to the second moment of the eigenvalue distribution of $\tilde{U}_D$ and obeys $B^2 < 1 + (4/\tilde{\theta}^2)$, in contrast to the classical result $O(1/\alpha^2)$. Grover search, therefore, can be generalized to an entire class of algorithms that use different diffusion operators. This flexibility is one of the reasons why Grover search ideas appear frequently in quantum algorithms.
- Quantum search can be implemented so as to be robust also against faulty queries, a problem known as *bounded-error search*. When the query has a bounded coherent

error, say $\tilde{U}_f|i\rangle|1\rangle = \sqrt{p_i}(-1)^{f(i)}|i\rangle|1\rangle + \sqrt{1-p_i}|\tilde{i}\rangle|0\rangle$ with each $p_i \geq 0.9$ ($|\tilde{i}\rangle$ are arbitrary states and the last qubit is a witness for the fault), quantum search can still be implemented using $O\left(\sqrt{N}\right)$ queries [22]. The deterioration of the search algorithm depends on p_i, but that is only in the scaling constant for the number of queries.

A useful generalization of the quantum search algorithm is the *amplitude amplification* technique [10, 16], which can be applied on top of nearly any quantum algorithm for any problem. It says that given a quantum algorithm that solves a problem with a small success probability ϵ, the success probability can be increased to roughly $m^2\epsilon$ using $O(m)$ calls to that algorithm. (Classically, the success probability can be increased to only about $m\epsilon$.) For the standard search problem, the simple algorithm that picks a random item has success probability $\epsilon = 1/N$, which the quantum search algorithm increases to $\Theta(1)$.

More formally, let V be the unitary operator corresponding to an algorithm that evolves the initial state $|s\rangle$ to $V|s\rangle$. Its success probability is $\epsilon = |\langle t|V|s\rangle|^2 = |V_{ts}|^2$. The algorithm obtained by replacing $|s\rangle$ by $V|s\rangle$ in Grover search, i.e., replacing U_D by $VU_DV^\dagger$, then increases the success probability to $\Theta(1)$ in $O\left(1/\sqrt{|V_{ts}|}\right)$ iterations. In particular, this algorithm evolves the quantum state in the two-dimensional subspace spanned by $|t\rangle$ and $V|s\rangle$, rotating it by angle $2\sin^{-1}|V_{ts}|$ at every iteration. In order to implement U_f, the algorithm needs a witness for the correctness of the output. Thus, we have [10, 16]:

Theorem 2 (Amplitude amplification) *Let $\mathcal{A}$ be a quantum algorithm that outputs a correct answer with witness, with known probability $\epsilon = \sin^2\theta$. Furthermore, let $m = \lfloor \pi/(4\theta) \rfloor$. Then there is an algorithm $\mathcal{A}'$ that uses $2m+1$ calls to $\mathcal{A}$ and $\mathcal{A}^{-1}$ and outputs a correct answer with probability $\epsilon' \geq 1-\epsilon$.*

Depending on the actual implementation, it is possible to vary the quantum search algorithm somewhat from the preceding deterministic and optimal approach and obtain small improvements:

- The algorithm needs $O\left(\sqrt{N}\log N\right)$ qubit-level operations in order to implement $H^{\otimes n}$ and U_0. The $\log N$ factor in this count can be suppressed by adding a small number of queries to the algorithm. A simple scheme divides the n qubits into k sets of n/k qubits each and uses the Grover diffusion operators ${U_D}^{(i)} = H^{\otimes(n/k)}{U_0}^{(i)}H^{\otimes(n/k)}$ that act only on one set at a time leaving the other sets unchanged [18]. Sequentially going through all the sets generates the transformation $V = \prod_{i=1}^{k}(U_D^{(i)}U_f)H^{\otimes n}$, which is then used for amplitude amplification with the initial state $|0\rangle^{\otimes n}$. Overall, the number of qubit-level operations reduce by a factor $\Theta(k)$, while the number of queries go up by a factor $1+\Theta(kN^{-1/k})$, provided $kN^{-1/k} = o(1)$. The choice $k = \Theta(n/\log n)$ reduces the qubit-level operations to $O\left(\sqrt{N}\log n\right)$, at the cost of increasing the queries by a factor $1+\Theta(1/\log n)$.
- Consider the partial search problem where the items are separated into N/b blocks of size b each, and only the block containing the desired item is to be located using the same U_f. In that case, the number of queries can be reduced by $0.34\sqrt{b}$ for large b [25]. The procedure first uses Grover search to make the amplitudes of nontarget blocks sufficiently small, then applies Grover search in parallel within each block to make the amplitudes of the target block sufficiently negative (amplitudes of nontarget blocks remain unchanged in this step), and then executes a final U_D operation to reduce the amplitudes of nontarget blocks to zero.
- Though Grover search proceeds from $|s\rangle$ to $|t\rangle$ with uniform speed in the Hilbert space, it slows down in terms of the success probability as it nears the target state. So one can reduce the expected number of queries, by stopping the algorithm before reaching the target state and then looking for the desired

item probabilistically. That amounts to minimizing $(Q+1)/p$, with the success probability $p = \sin^2\left((2Q+1)\sin^{-1}(1/\sqrt{N})\right)$. This probabilistic search reduces the required number of queries asymptotically to $0.6900\sqrt{N}$ [14].

- Consider the search problem where the times required for querying different items are not the same. This can happen when the query is an algorithm $\mathcal{A}$ acting on different input states $|i\rangle$. When the query for the i^{th} item takes time t_i to execute, unstructured quantum search can be accomplished in time $O\left(T = \left(\sum_{i=1}^{N} t_i^2\right)^{1/2}\right)$ when t_i are known apriori [3]. The strategy is to divide the items into multiple groups so that items in every group have query times within a constant factor, apply Grover search within each group, and then query the groups sequentially. The number of groups needed is $O(\log N)$, and the result improves upon the global Grover search bound $O\left(\sqrt{N}t_{\max}\right)$. Amplitude amplification can be used when $\mathcal{A}$ is probabilistic, and a polylogarithmic overhead in T is required when t_i are not known in advance.

Applications

NP-Complete Problems

Even though NP-complete problems have some structure, there are few known algorithms that exploit this structure to solve them, and often the only recourse left is to solve them as exhaustive search problems. Since quantum search does not assume any structure or pattern in the input data, it provides a square-root speedup in such cases.

Quantum Counting

The counting problem is to find the number of items in a set that satisfy the given query. Its quantum solution is based on the fact that the iterative evolution in Grover search is periodic, with angular frequency $\omega = 2\sin^{-1}\left(\sqrt{M/N}\right)$. The phase estimation procedure (based on quantum Fourier transform) [24] can therefore determine M approximately, up to error $\sqrt{M}$, using $O\left(\sqrt{N}\right)$ queries. Then, using the property that ω differs by $1/\sqrt{M(N-M)}$ between adjacent values of M, M can be determined exactly using $O/\sqrt{M(N-M)}$ queries [27]. For $M = o(N)$, this quantum result is a power-law improvement over the classical result of $\Theta(N)$ queries, although not as good as a square-root speedup.

Element Distinctness

An early application of Grover search was to find collisions, i.e., given oracle access to a 2-to-1 function f, find distinct arguments x, y such that $f(x) = f(y)$. The quantum *collision problem* has an $O(N^{1/3})$ algorithm [9]. The more general *element distinctness problem* is to find distinct x, y such that $f(x) = f(y)$, for an unknown function f that can be accessed only by an oracle. Ambainis discovered an optimal $O(N^{2/3})$ quantum algorithm for this problem [2]. It searches a suitably constructed graph, with the Grover diffusion operation replaced by a certain quantum walk. The vertices of the graph correspond to various subsets of items $S_j \subseteq \{1, 2, \ldots, N\}$, each of size $N^{2/3}$, two vertices are connected by an edge when the corresponding subsets differ by only one item, and the target vertices are the subsets S_j that solve the element distinctness problem.

Distributed Search

Grover search is also useful in improving communication complexity. For example, a straightforward distributed implementation of the quantum search algorithm solves the *set intersection problem* or the *appointment problem*. The result is, when A and B have respective data strings $x, y \in \{0,1\}^N$, and they want to find an index i such that $x_i = y_i = 1$, only $O\left(\sqrt{N}\log N\right)$ qubits of communication is necessary [11]. This result has led to an exponential classical/quantum separation in the memory required to evaluate a certain total function with a streaming input [26].

Fixed-Point Search

The iterative evolution in Grover's search algorithm is cyclic, and knowledge of N is necessary

to stop it at the right time to find the target item. In contrast, fixed-point search algorithms converge monotonically to the target state. For a long time, a fixed-point quantum algorithm was considered unlikely, since any iterative unitary evolution is periodic. Surprisingly, a way out was provided by recursive unitary evolution. When the reflection operations in the amplitude amplification algorithm are replaced by selective phase shifts of $\pi/3$ (e.g., $R_i = 1 + (e^{i\pi/3} - 1)|i\rangle\langle i|$ for the state $|i\rangle$), $|\langle t|V|s\rangle|^2 = 1 - \epsilon$ implies $|\langle t|VR_sV^\dagger R_tV|s\rangle|^2 = 1 - \epsilon^3$ [19]. So each recursive substitution of the operator V by the operator $VR_sV^\dagger R_tV$ reduces the deviation of the final state from the target state to the cube of what it was before. (The corresponding best classical reduction is $O(\epsilon^2)$, e.g., by majority rule selection after three trials.) This technique does not give a square-root speedup for search, but it is useful when ϵ is small, for instance, in error correction. It has been used to design composite pulse sequences for reducing systematic errors [35]. An iterative quantum search algorithm with similar properties has been obtained combining reflection operations with non-unitary projective measurements [41]. Another recent construction is a bounded-error quantum search algorithm (i.e., success probability $p \geq 1 - \delta$) that varies the phase shifts between π and $\pi/3$ as a function of the iteration number [42]. It exhibits square-root speedup as well as convergence to the target state, provided that both δ and $|\langle t|s\rangle|$ are bounded away from zero.

Spatial Search

This is the search problem where the items belonging to a database are spread over distinct physical locations, say a d-dimensional lattice, and there is a restriction that one can proceed from any location to only its neighbors while searching for the target item. Its quantum solution replaces the global Grover diffusion operator by a local quantum walk, and Grover search becomes the $d \to \infty$ limit. The required number of queries has to obey the double lower bound $\Omega\left(dN^{1/d}, \sqrt{N}\right)$; the former arises from the finite speed of movement on the lattice and the latter from the optimality of Grover search. The best algorithms are found in the framework of relativistic quantum mechanics. They use $O\left(\sqrt{N}\right)$ queries for $d > 2$, with the scaling constant approaching $\pi/4$ from above as $d \to \infty$ [1, 33]. In the critical dimension $d = 2$, the algorithms are slowed down by logarithmic factors arising from the infrared divergence, and the best known algorithm requires $O\left(\sqrt{N \log N}\right)$ queries [39]. For non-integer values of d, the scaling behavior of the algorithm has been verified using numerical simulations on fractal lattices [32].

Markov Chain Evolution

Generic stationary stochastic processes (e.g., random walks) are defined in terms of transition matrices that encode the possible evolutionary changes at each step. Many properties of the resulting evolution (e.g., hitting time, detection, mixing, escape time) scale as negative powers of the spectral gap of the transition matrix. For Markovian evolution on bipartite graphs, the transition matrix can be separated into two disjoint parts, say $\{x\} \to \{y\}$ and $\{y\} \to \{x\}$. Szegedy constructed two reflection operators from these parts and defined a quantum evolution operator as their product [38] (classical Markov chain evolution does not allow such reflection operators). The spectral gap of this quantum evolution operator scales as the square root of the spectral gap of the original transition matrix and so speeds up the evolution the same way as Grover search does.

Recursive Search

Game-tree evaluation, which is a recursive search problem, is an extension of unstructured search. Classically, using the alpha-beta pruning technique, the value of a balanced binary AND-OR tree can be computed with $o(1)$ error in expected time $O\left(N^{\log_2[(1+\sqrt{33})/4]}\right) = O(N^{0.754})$ [36]. This is optimal even for bounded-error algorithms [37]. By applying quantum search recursively, a depth-d regular AND-OR tree can be evaluated with constant error in time $\sqrt{N} \cdot O(\log N)^{d-1}$. The log factors come from amplifying the success probability of inner searches to

be close to one. Bounded-error quantum search eliminates these log factors, reducing the time to $O\left(\sqrt{N}\cdot c^{d}\right)$ for some constant c. Recently, an $O(N^{0.5+o(1)})$ time algorithm has been discovered for evaluating an arbitrary AND-OR tree on N variables [4, 13].

Open Problems

In several applications of the quantum search algorithm, only the leading asymptotic behavior of the query complexity is known, and attempts to suppress logarithmic corrections (when they appear) and reduce the scaling constants continue [34]. In this section, we point out some other offbeat applications.

Hamiltonian Evolution

Many conventional algorithms for simulations of quantum systems with sparse Hamiltonians use the Trotter formula with a small step size. They have a power-law dependence of the computational complexity on the simulation error and hence are not efficient. In contrast, Grover search amounts to a Trotter formula with the largest possible step size, given the projection operator nature of the terms in the Hamiltonian. A recent exciting realization is that this feature leads to only logarithmic dependence of the computational complexity on the simulation error, which is an exponential improvement. The general strategy is to decompose the sparse Hamiltonian as a sum of projection operators, formulate the evolution problem as a multi-query search problem, and then use a large step size Trotter formula to simulate it [7, 31]. This framework can also readily benefit classical simulations of quantum systems.

Molecular Biology

Many molecular processes of metabolism occur at scales, nanometer and picosecond, where quantum dynamics is relevant. They frequently involve unstructured search and transport, in the sense that correct ingredients for the processes have to be found from the mixture of molecules floating around. Evolution over billions of years has certainly produced complex machinery to carry out these searches efficiently, although we do not fully comprehend their optimization criteria. Attempts to understand some of these processes suggest that Grover search may have played a role in their design, quite likely exploiting coherent coupled vibrational modes and not quantum superposition. An intriguing example is that the universal genetic language uses an alphabet of four letters, while a binary alphabet would be sufficient and simpler to construct during evolution [30]. Coherent vibrational dynamics of molecules also contributes to efficient energy transport during photosynthesis and to the detection of smell [23].

Ordered Search

A sequentially ordered database can be easily searched by factoring $f(i)$ into subqueries for individual digits of i. An alternative is to use a different oracle $g(i)$, such that $g(i) = 0$ when i represents items before the desired item and $g(i) = 1$ otherwise. Classically, binary search is the optimal algorithm given either $f(i)$ or $g(i)$ and requires $\lceil \log_2 N \rceil$ queries. The optimal quantum algorithm for $f(i)$ is quaternary search with $0.5\lceil \log_2 N \rceil$ queries, but surprisingly a quantum algorithm using $g(i)$ can do better. In case of $g(i)$, though the optimal solution is unknown, the query complexity for an exact algorithm has a lower bound of $0.221 \log_2 N$ [21] and a known solution of $0.433 \log_2 N$ [12] (there also exists a quantum stochastic Las Vegas algorithm with $0.32 \log_2 N$ expected queries and $o(1)$ error [6]).

Search with Additional Structure

It may be possible to speed up a search process beyond the square-root speedup of Grover search, when the problem has extra structure beyond the minimal information provided by the oracle $f(i)$. The details of the algorithm and the extent of speedup would then depend on the extra structure, and the possibilities are open to explorations. Some examples are symmetries among the items, associative memory recall with connections, and patterns in the Boolean function

to be evaluated. Another problem of interest is determination of the complete path (with certain properties) from the initial to the target state instead of locations of just the end points.

Perspective

In a lecture at the Bell Labs in 1985, Richard Feynman made an interesting observation. In the 1940s when airplanes were being developed, aeronautical engineers had proved bounds and theorems about why planes would never be able to fly faster than the speed of sound. For several years, this speed was regarded as fundamentally a bound for flights as the speed of light is for communications. However, gradually just by using intelligent design, it was discovered that airplanes could indeed fly faster than the speed of sound – only the rules of design in the new regime were very different. The question is whether the bounds on quantum computation (specifically the $\Omega\left(\sqrt{N}\right)$ bound for search) will continue to hold, or by making the rules of design very different, just as in the case of supersonic airplanes, someone will find a way around these bounds. No one has found any loophole in the arguments in the 20 years since the lower bound for quantum search was discovered, despite numerous scientists from different fields having tried their hand at it. On the other hand, even though this bound has been derived over and over again using different methods, no one has come up with a simple and short physical explanation for it, which would give one the assurance that one really understood it.

Cross-References

- ▸ Quantum Algorithm for Element Distinctness
- ▸ Routing

Recommended Reading

1. Aaronson S, Ambainis A (2005) Quantum search of spatial regions. Theory Comput 1:47–79
2. Ambainis A (2007) Quantum walk algorithm for element distinctness. SIAM J Comput 37(1):210–239
3. Ambainis A (2010) Quantum search with variable times. Theory Comput Syst 47(3): 786–807
4. Ambainis A, Childs AM, Reichardt BW, Špalek R, Zhang S (2010) Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. SIAM J Comput 39(6):2513–2530
5. Bennett CH, Bernstein E, Brassard G, Vazirani U (1997) Strengths and weaknesses of quantum computing. SIAM J Comput 26(5):1510–1523
6. Ben-Or M, Hassidim A (2007) Quantum search in an ordered list via adaptive learning. arXiv:quant-ph/0703231
7. Berry DW, Childs AM, Cleve R, Kothari R, Somma RD (2014) Exponential improvement in precision for simulating sparse Hamiltonians. In: Proceedings of the 46th ACM symposium on theory of computing (STOC'14), New York, 31 May–3 June 2014, pp 283–292
8. Brassard G (1997) Searching a quantum phone book. Science 275(5300):627–628
9. Brassard G, Høyer P, Tapp A (1998) Quantum cryptanalysis of hash and claw-free functions. In: Proceedings of the 3rd Latin American theoretical informatics symposium (LATIN'98). Lecture notes in computer science, vol 1380. Springer, Berlin/Heidelberg, pp 163–169
10. Brassard G, Høyer P, Tapp A (1998) Quantum Counting. In: Proceedings of the 25th international colloquium on automata, languages and programming (ICALP'98). Lecture notes in computer science, vol 1443. Springer, Berlin/Heidelberg, pp 820–831
11. Buhrman H, Cleve R, Wigderson A (1998) Quantum vs. classical communication and computation. In: Proceedings of the 30th ACM symposium on theory of computing (STOC'98), Dallas, 24–26 May 1998, pp 63–68
12. Childs AM, Landahl AJ, Parrilo PA (2007) Improved quantum algorithms for the ordered search problem via semidefinite programming. Phys Rev A 75(3):032335
13. Farhi E, Goldstone J, Gutmann S (2008) A quantum algorithm for the Hamiltonian NAND tree. Theory Comput 4:169–190
14. Gingrich RM, Williams CP, Cerf NJ (2000) Generalized quantum search with parallelism. Phys Rev A 61(5):052313
15. Grover LK (1996) A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th ACM symposium on theory of computing (STOC'96), Philadelphia, 22–24 May 1996, pp 212–219
16. Grover LK (1998) A framework for fast quantum mechanical algorithms. In: Proceedings of the 30th ACM symposium on theory of computing (STOC'98), Dallas, 24–26 May 1998, pp 53–62
17. Grover LK (2001) From Schrödinger's equation to the quantum search algorithm. Pramana 56:333–348
18. Grover LK (2002) Trade-offs in the quantum search algorithm. Phys Rev A 66(5):052314

19. Grover LK (2005) Fixed-point quantum search. Phys Rev Lett 95:150501
20. Grover LK, Sengupta AM (2002) From coupled pendulums to quantum search. In: Brylinski RK, Chen G (eds) Mathematics of quantum computation. CRC, Boca Raton, pp 119–134
21. Høyer P, Neerbak J, Shi Y (2002) Quantum complexities of ordered searching, sorting and element distinctness. Algorithmica 34(4):429–448
22. Høyer P, Mosca M, de Wolf R (2003) Quantum search on bounded-error inputs. In: Proceedings of the 30th international colloquium on automata, languages and programming (ICALP 2003). Lecture notes in computer science, vol 2719. Springer, Berlin/Heidelberg, pp 291–299
23. Huelga SF, Plenio MB (2013) Vibrations, quanta and biology. Contemp Phys 54(4):181–207
24. Kitaev AYu, Shen AH, Vyalyi MN (2002) Classical and quantum computation. Graduate studies in mathematics, vol 47. American Mathematical Society, Providence. Section 13.5
25. Korepin VE, Grover LK (2006) Simple algorithm for partial quantum search. Quantum Inf Process 5(1):5–10
26. Le Gall F (2006) Exponential separation of quantum and classical online space complexity. In: Proceedings of the 18th annual ACM symposium on parallelism in algorithms and architectures (SPAA 2006), Cambridge, 30 July–2 Aug 2006, pp 67–73
27. Mosca M (2001) Counting by quantum eigenvalue estimation. Theor Comput Sci 264:139–153
28. Patel A (2001) Quantum database search can do without sorting. Phys Rev A 64(3):034303
29. Patel A (2006) Optimal database search: waves and catalysis. Int J Quantum Inf 4(5):815–825; Erratum: ibid. 5(3):437 (2007)
30. Patel A (2008) Towards understanding the origin of genetic languages. In: Abbott D, Davies PCW, Pati AK (eds) Quantum aspects of life. Imperial College Press, London, pp 187–219
31. Patel A (2014) Optimisation of quantum Hamiltonian evolution. In: Proceedings of the 32nd international symposium on lattice field theory, New York, 23–28 June 2014. PoS(LATTICE2014)324
32. Patel A, Raghunathan KS (2012) Search on a fractal lattice using a quantum random walk. Phys Rev A 86(1):012332
33. Patel A, Rahaman MdA (2010) Search on a hypercubic lattice using a quantum random walk: I. d>2. Phys Rev A 82(3):032330
34. Portugal R (2013) Quantum walks and search algorithms. Springer, New York
35. Reichardt BW, Grover LK (2005) Quantum error correction of systematic errors using a quantum search framework. Phys Rev A 72:042326
36. Saks M, Wigderson A (1986) Probabilistic Boolean decision trees and the complexity of evaluating game trees. In: Proceedings of the 27th annual IEEE symposium on foundations of computer science (FOCS), Toronto, 27–29 Oct 1986, pp 29–38
37. Santha M (1995) On the Monte Carlo decision tree complexity of read-once formulae. Random Struct Algorithms 6(1):75–87
38. Szegedy M (2004) Quantum speed-up of Markov chain based algorithms. In: Proceedings of the 45th annual IEEE symposium on foundations of computer science (FOCS'04), Rome, 17–19 Oct 2004, pp 32–41
39. Tulsi A (2008) Faster quantum-walk algorithm for the two-dimensional spatial search. Phys Rev A 78(1):012310
40. Tulsi A (2012) General framework for quantum search algorithms. Phys Rev A 86(4):042331
41. Tulsi T, Grover LK, Patel A (2006) A new algorithm for fixed point quantum search. Quantum Inf Comput 6(6):483–494
42. Yoder TJ, Low GH, Chuang IL (2014) Optimal fixed-point quantum amplitude amplification using Chebyshev polynomials. Phys Rev Lett 113:210501
43. Zalka C (1999) Grover's quantum searching algorithm is optimal. Phys Rev A 60(4):2746–2751

Query Release via Online Learning

Jonathan Ullman
Department of Computer Science, Columbia University, New York, NY, USA

Keywords

Computational learning theory; Differential privacy; Statistical query model

Years and Authors of Summarized Original Work

2010; Hardt, Rothblum
2012; Hardt, Rothblum, Servedio
2012; Thaler, Ullman, Vadhan
2013; Ullman
2014; Chandrasekaran, Thaler, Ullman, Wan
2014; Bun, Ullman, Vadhan

Problem Definition

Our goal is to design differentially private algorithms to answer *statistical queries* on a sensitive database. We model the database $D = (x_1, \ldots, x_n) \in (\{0,1\}^d)^n$ as a collection of n records – one per individual – each consisting

of d binary attributes. A differentially private algorithm is a randomized algorithm whose output distribution does not depend "significantly" on any one record of the database. The formal definition is as follows:

Definition 1 ([8]) An algorithm $\mathcal{A}:(\{0,1\}^d)^n \to \mathcal{R}$ is (ε, δ)-*differentially private* if for every pair of databases $D, D' \in (\{0,1\}^d)^n$ that differ on at most one row and every $S \subseteq \mathcal{R}$,

$$\mathbb{P}[\mathcal{A}(D) \in S] \leq e^{\varepsilon}\mathbb{P}[\mathcal{A}(D') \in S] + \delta.$$

Henceforth, we will say $\mathcal{A}$ is *differentially private* if it satisfies $(1, 1/n^2)$-differential privacy. (The choice of $\varepsilon = 1$ and $\delta = 1/n^2$ is arbitrary and can be replaced with $\varepsilon = c$ and $\delta = 1/n^{1+c'}$ for any constants $c, c' > 0$ without affecting any stated results).

A *statistical query* (henceforth, simply *query*) is specified by a Boolean predicate $q : \{0,1\}^d \to \{0,1\}$. The answer to a query is the expected value of the predicate over records in the database. Abusing notation, we write

$$q(D) = \frac{1}{n}\sum_{i=1}^{n} q(x_i).$$

We wish to design a differentially private algorithm $\mathcal{A}$ that takes a database and a set of statistical queries and outputs an approximate answer to each query.

Definition 2 An algorithm $\mathcal{A}$ is α-*accurate* for a query q if $\mathcal{A}(D)$ outputs $a \in [0,1]$ such that $|a - q(D)| \leq \alpha$, with probability at least 99/100. An algorithm $\mathcal{A}$ is α-*accurate* for a set of queries $Q = \{q_1, q_2, \ldots\}$ if $\mathcal{A}(D)$ outputs $(a_q)_{q\in Q}$ such that for every $q \in Q$, $|a_q - q(D)| \leq \alpha$ with probability at least 99/100.

The goal is to design differentially private algorithms that are α-accurate for sets of queries Q as large as possible. As privacy is easier to achieve when the number of records n is large, we will seek to obtain privacy and accuracy for n as small as possible. Lastly, we seek to make the algorithms as computationally efficient as possible.

Key Results

As a baseline, we will consider simple *additive perturbation* [2, 6–8], which answers each query by independently perturbing the answer with noise from a suitable distribution.

Theorem 1 *There is a differentially private algorithm $\mathcal{A}$ that takes a database $D \in (\{0,1\}^d)^n$ and a set of queries $Q = \{q_1, \ldots, q_k\}$ as input, runs in time* $\mathrm{poly}(n, d, |q_1| + \cdots + |q_k|)$, *and is α-accurate for Q so long as $n \geq \tilde{O}(|Q|^{1/2}/\alpha)$.*

Here, $|q|$ represents the time complexity of evaluating the predicate on a single row of the database. Typically, this it is assumed to be $\mathrm{poly}(d)$.

Additive perturbation is differentially private and computationally efficient, but requires that the size of the database be polynomial in the number of queries, and thus is restricted to answering at most about n^2 queries. As we will see, it is possible to accurately answer *exponentially* more queries under differential privacy.

Answering Many Queries via No-Regret Learning

The first algorithm that improved on additive perturbation for answering arbitrary queries was given by Blum, Ligett, and Roth [3]. Surprisingly, they showed for the first time that it was possible to answer *exponentially many* queries under differential privacy. Subsequent to their work, there were several improvements in the computational efficiency, functionality, and quantitative guarantees of their algorithm. This work led to the *private multiplicative weights algorithm* of Hardt and Rothblum [10]. We summarize the capabilities of this algorithm in the following theorem.

Theorem 2 ([10]) *There is a differentially private algorithm $\mathcal{A}$ that takes a database $D \in (\{0,1\}^d)^n$ and a set of queries $Q = \{q_1, \ldots, q_k\}$ as input, runs in time* $\mathrm{poly}(n, 2^d, |q_1| + \cdots + |q_k|)$, *and is α-accurate for $|Q|$ so long as $n \geq \tilde{O}(\sqrt{d}\log|Q|/\alpha^2)$.*

The private multiplicative weights algorithm is based on the following surprisingly simple

framework: Begin with a "crude approximation" of the database D^1. Then, for $t = 1, \ldots, T$, find (in a differentially private manner) a query $q^t \in Q$ such that the approximation D^t does not give an accurate answer. That is, $|q^t(D) - q^t(D^t)| > \alpha$. Use q^t to "update" D^t into a better approximation D^{t+1}. Finally, output the answers to Q given by D^T.

Remarkably, it is possible to find a query $q^t \in Q$ such that D^t is inaccurate (or conclude that none exists) using much less data than would be required to simply answer all the queries in Q using additive perturbation. Perhaps even more surprisingly, it can be shown that if the updates are performed using the *multiplicative weights update rule*, then after $T = O(d/\alpha^2)$ iterations (independent of $|Q|$!), the database D^T will give an accurate answer to *every* $q \in Q$. This argument makes use of the guarantee that multiplicative weights update rule is a "no-regret learning algorithm" (cf. the survey of Arora, Hazan, and Kale [1] for more information about the multiplicative weights update rule). This fast convergence makes it possible to argue that the algorithm can give accurate and differentially private answers with much less data than would be required by simple additive perturbation.

Computational Complexity and Optimality

When $|Q|$ is large, the private multiplicative weights algorithm requires many fewer records n than additive perturbation (when $|Q| \gg d/\alpha^2$). One might ask whether even fewer records suffices. Bun, Ullman, and Vadhan [4] gave a negative answer to this question, and showed that the private multiplicative weights algorithm uses essentially the fewer records possible.

Theorem 3 ([4]) *There is no (even computationally inefficient) differentially private algorithm $\mathcal{A}$ that takes an arbitrary set of queries $Q = \{q_1, \ldots, q_k\}$ with $k \gg d/\alpha^2$ and a database $D \in (\{0,1\}^d)^n$ with $n \leq \tilde{\Omega}(\sqrt{d} \log |Q|/\alpha^2)$ as input and is α-accurate for the set of queries Q.*

A drawback of the private multiplicative weights algorithm (and all known algorithms with similar properties), when compared to additive perturbation, is computational complexity. Even when answering a polynomial number of efficiently computable queries, the running time of private multiplicative weights is dominated by the factor of 2^d, which is exponential in the number of attributes in the database. Ullman [13] showed that this is inherent, and (under a widely believed cryptographic assumption) improving on additive perturbation requires exponential running time.

Theorem 4 ([13]) *Assuming the existence of one-way functions, there is no differentially private algorithm $\mathcal{A}$ that takes an arbitrary set of queries $Q = \{q_1, \ldots, q_k\}$ database $D \in (\{0,1\}^d)^n$ with $n \leq \tilde{\Omega}(|Q|^{1/2})$ as input, runs in time* $\mathrm{poly}(n, d, |q_1| + |q_k|)$*, and is* $1/3$*-accurate for the set of queries Q.*

Together, these negative results show the private multiplicative weights algorithm is nearly optimal for answering large sets of arbitrary statistical queries under differential privacy.

Faster Algorithms for Marginal Queries via Efficient Learning

Given the hardness of answering arbitrary queries, there has been a significant effort to design faster differentially private algorithms that improve on additive noise for natural restricted set of queries. One such set of queries is *k-way marginals*. These queries are specified by a subset of attributes $S \subseteq [d]$ of size at most k and a pattern $t \in \{0,1\}^{|S|}$ and asks for the fraction of records in D that have each attribute $j \in S$ set to t_j. Note that there are $\mathrm{poly}(d^k)$ such queries, and thus, additive perturbation would require running time $\mathrm{poly}(d^k)$ and $n \geq \mathrm{poly}(d^k)$. On the other hand, private multiplicative weights would require running time $\mathrm{poly}(2^d)$, but $n \geq \tilde{O}(k\sqrt{d})$ would suffice.

Most of the more effective algorithms for answering k-way marginal queries are based on the following technique, introduced by Gupta et al. [9]: View the database D as specifying a function $f_D(q) = q(D)$ that maps a query to its answer on D, and then attempt to "learn" a differentially private approximation $g_D \approx f_D$. Intuitively, the value of this approach is that learning algorithms see the evaluation of f_D on a small

number of queries and then are able to predict the value of f_D on new queries. Since the learning algorithm only needs a small number of examples, it is easier to ensure differential privacy. If the queries q are "simple," then good learning algorithms may exist for the function f_D. In the case of k-way marginal queries, it turns out that f is in fact (an average of) *conjunctions*, and there are learning algorithms for this class of functions that satisfy various interesting parameter trade-offs. This technique underlies the following two results:

Theorem 5 ([12], building on [11]) *For every $k \in \mathbb{N}$, there is a differentially private algorithm $\mathcal{A}$ that takes a database $D \in (\{0,1\}^d)^n$ as input and runs in time $\mathrm{poly}(n, d^{\sqrt{k}})$, and if $n \geq \mathrm{poly}(d^{\sqrt{k}})$, $\mathcal{A}$ outputs a summary of the database that yields $1/100$-accurate answers to every k-way marginal query. That is, for every k-way marginal query q, one can obtain a $1/100$-accurate answer to q in time $\mathrm{poly}(n, d^{\sqrt{k}})$.*

Theorem 6 ([5]) *For every $k \in \mathbb{N}$, there is a differentially private algorithm $\mathcal{A}$ that takes a database $D \in (\{0,1\}^d)^n$ as input and runs in time $\mathrm{poly}(n, 2^{d^{1-1/100\sqrt{k}}})$, and if $n \geq kd^{.51}$, $\mathcal{A}$ outputs $1/100$-accurate answers to every k-way marginal query.*

We remark that there are many other algorithms for answering k-way marginal queries based on this learning approach, each achieving different parameter trade-offs and guarantees of accuracy. At the time of writing, improving these algorithms and extending these techniques to richer classes of queries remains an active area of research.

Recommended Reading

1. Arora S, Hazan E, Kale S (2012) The multiplicative weights update method: a meta-algorithm and applications. Theory Comput 8(1):121–164
2. Blum A, Dwork C, McSherry F, Nissim K (2005) Practical privacy: the SuLQ framework. In: PODS. ACM, Baltimore MD, pp 128–138
3. Blum A, Ligett K, Roth A (2013) A learning theory approach to noninteractive database privacy. J ACM 60(2):12
4. Bun M, Ullman J, Vadhan SP (2014) Fingerprinting codes and the price of approximate differential privacy. In: STOC. ACM, New York, NY, pp 1–10
5. Chandrasekaran K, Thaler J, Ullman J, Wan A (2014) Faster private release of marginals on small databases. In: ITCS. ACM, Princeton, NJ, pp 387–402
6. Dinur I, Nissim K (2003) Revealing information while preserving privacy. In: PODS. ACM, San Diego, CA, pp 202–210
7. Dwork C, Nissim K (2004) Privacy-preserving datamining on vertically partitioned databases. In: CRYPTO, Santa Barbara, CA, pp 528–544
8. Dwork C, McSherry F, Nissim K, Smith A (2006) Calibrating noise to sensitivity in private data analysis. In: Halevi S, Rabin T (eds) TCC. Lecture notes in computer science, vol 3876. Springer, New York, NY, pp 265–284
9. Gupta A, Hardt M, Roth A, Ullman J (2013) Privately releasing conjunctions and the statistical query barrier. SIAM J Comput 42(4):1494–1520
10. Hardt M, Rothblum G (2010) A multiplicative weights mechanism for privacy-preserving data analysis. In: Proceedings of the 51st foundations of computer science (FOCS). IEEE, Las Vegas, NV, pp 61–70
11. Hardt M, Rothblum GN, Servedio RA (2012) Private data release via learning thresholds. In: SODA. SIAM, Kyoto, Japan, pp 168–187
12. Thaler J, Ullman J, Vadhan SP (2012) Faster algorithms for privately releasing marginals. In: ICALP (1). Springer, Warwick, UK, pp 810–821
13. Ullman J (2013) Answering $n^{2+o(1)}$ counting queries with differential privacy is hard. In: STOC. ACM, Palo Alto, CA, pp 361–370

Quorums

Dahlia Malkhi
Microsoft, Silicon Valley Campus, Mountain View, CA, USA

Keywords

Coteries; Quorum systems; Voting systems

Years and Authors of Summarized Original Work

1985; Garcia-Molina, Barbara

Problem Definition

Quorum systems are tools for increasing the availability and efficiency of replicated services. A *quorum system* for a universe of servers is a collection of subsets of servers, each pair of which intersect. Intuitively, each quorum can operate on behalf of the system, thus increasing its availability and performance, while the intersection property guarantees that operations done on distinct quorums preserve consistency.

The motivation for quorum systems stems from the need to make critical missions performed by machines that are reliable. The only way to increase the reliability of a service, aside from using intrinsically more robust hardware, is via replication. To make a service robust, it can be installed on multiple identical servers, each one of which holds a copy of the service state and performs read/write operations on it. This allows the system to provide information and perform operations even if some machines fail or communication links go down. Unfortunately, replication incurs a cost in the need to maintain the servers consistent. To enhance the availability and performance of a replicated service, Gifford and Thomas introduced in 1979 [3, 14] the usage of *votes* assigned to each server, such that a majority of the sum of votes is sufficient to perform operations. More generally, quorum systems are defined formally as follows:

Quorum system: Assume a *universe* U of servers, $|U| = n$, and an arbitrary number of clients. A *quorum system* $\mathcal{Q} \subseteq 2^U$ is a set of subsets of U, every pair of which intersect. Each $Q \in \mathcal{Q}$ is called a *quorum*.

Access Protocol

To demonstrate the usability of quorum systems in constructing replicated services, quorums are used here to implement a multi-writer multi-reader atomic shared variable. Quorums have also been used in various *mutual exclusion* protocols, to achieve Consensus, and in commit protocols.

In the application, clients perform read and write operations on a variable x that is replicated at each server in the universe U. A copy of the variable x is stored at each server, along with a timestamp value t. Timestamps are assigned by a client to each replica of the variable when the client writes the replica. Different clients choose distinct timestamps, e.g., by choosing integers appended with the name of c in the low-order bits. The read and write operations are implemented as follows.

Write: For a client c to write the value v, it queries each server in some quorum Q to obtain a set of value/timestamp pairs $A = \{\langle v_u, t_u\rangle\}_{u\in Q}$; chooses a timestamp $t \in T_c$ greater than the highest timestamp value in A; and updates x and the associated timestamp at each server in Q to v and t, respectively.

Read: For a client to read x, it queries each server in some quorum Q to obtain a set of value/timestamp pairs $A = \{\langle v_u, t_u\rangle\}_{u\in Q}$. The client then chooses the pair $\langle v, t\rangle$ with the highest timestamp in A to obtain the result of the read operation. It writes back $\langle v, t\rangle$ to each server in some quorum Q'.

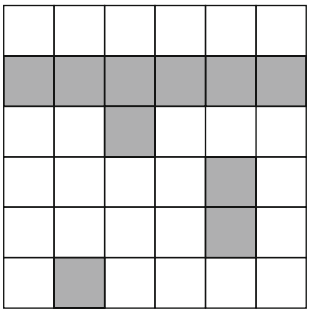
In both read and write operations, each server updates its local variable and timestamp to the received values $\langle v, t\rangle$ only if t is greater than the timestamp currently associated with the variable. The above protocol correctly implements the semantics of a multi-writer multi-reader atomic variable (see ▸ Linearizability).

Key Results

Perhaps the two most obvious quorum systems are the singleton, and the set of majorities, or more generally, weighted majorities suggested by Gifford [3].

Singleton: The set system $\mathcal{Q} = \{\{u\}\}$ for some $u \in U$ is the singleton quorum system.

Weighted Majorities: Assume that every server s in the universe U is assigned a number of votes w_s. Then, the set system $\mathcal{Q} = \{Q \subseteq U : \sum_{q\in Q} w_q > (\sum_{q\in U} w_q)/2\}$ is a quorum system called Weighted Majorities. When all the weights are the same, simply call this the system of Majorities.



Quorums, Fig. 1 The Grid quorum system of 6×6, with one quorum shaded

An example of a quorum system that cannot be defined by voting is the following Grid construction:

Grid: Suppose that the universe of servers is of size $n = k^2$ for some integer k. Arrange the universe into a $\sqrt{n} \times \sqrt{n}$ grid, as shown in Fig. 1. A quorum is the union of a full row and one element from each row below the full row. This yields the Grid quorum system, whose quorums are of size $O(\sqrt{n})$.

Maekawa suggests in [6] a quorum system that has several desirable symmetry properties, and in particular, that every pair of quorums intersect in exactly one element:

FPP: Suppose that the universe of servers is of size $n = q^2 + q + 1$, where $q = p^r$ for a prime p. It is known that a finite projective plane exists for n, with $q + 1$ pairwise intersecting subsets, each subset of size $q + 1$, and where each element is contained in $q + 1$ subsets. Then the set of finite projective plane subsets forms a quorum system.

Voting and Related Notions

Since generally it would be senseless to access a large quorum if a subset of it is a quorum, a good definition may avoid such anomalies. Garcia-Molina and Barbara [2] call such well-formed systems *coteries*, defined as follows:

Coterie: A *coterie* $\mathcal{Q} \subseteq 2^U$ is a quorum system such that for any $Q, Q' \in \mathcal{Q} : Q \not\subseteq Q'$.

Of special interest are quorum systems that cannot be reduced in size (i.e., that no quorum in the system can be reduced in size). Garcia-Molina and Barbara [2] use the term "dominates" to mean that one quorum system is always superior to another, as follows:

Domination: Suppose that $\mathcal{Q}, \mathcal{Q}'$ are two coteries, $\mathcal{Q} \neq \mathcal{Q}'$, such that for every $Q' \in \mathcal{Q}'$, there exists a $Q \in \mathcal{Q}$ such that $Q \subseteq Q'$. Then $\mathcal{Q}$ *dominates* $\mathcal{Q}'$.$\mathcal{Q}'$ is *dominated* if there exists a coterie $\mathcal{Q}$ that dominates it, and is *non-dominated* if no such coterie exists.

Voting was mentioned above as an intuitive way of thinking about quorum techniques. As it turns out, vote assignments and quorums are not equivalent. Garcia-Molina and Barbara [2] show that quorum systems are strictly more general than voting, i.e., each vote assignment has some corresponding quorum system but not the other way around. In fact, for a system with n servers, there is a double-exponential ($2^{2^{cn}}$) number of non-dominated coteries, and only $O(2^{n^2})$ different vote assignments, though for $n \leq 5$, voting and non-dominated coteries are identical.

Measures

Several measures of quality have been identified to address the question of which quorum system works best for a given set of servers; among these, *load* and *availability* are elaborated on here.

Load

A measure of the inherent performance of a quorum system is its *load*. Naor and Wool define in [10] the load of a quorum system as the probability of accessing the busiest server in the *best* case. More precisely, given a quorum system $\mathcal{Q}$, an *access strategy* w is a probability distribution on the elements of $\mathcal{Q}$; i.e., $\sum_{Q \in \mathcal{Q}} w(Q) = 1$. $w(Q)$ is the probability that quorum Q will be chosen when the service is accessed. Load is then defined as follows:

Load: Let a strategy w be given for a quorum system $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ over a universe U. For an element $u \in U$, the load induced by w on u is $l_w(u) = \sum_{Q_i \ni u} w(Q_i)$. The load induced by a strategy w on a quorum system $\mathcal{Q}$ is

$$L_w(\mathcal{Q}) = \max_{u \in U} \{l_w(u)\}.$$

The *system load* (or just *load*) on a quorum system $\mathcal{Q}$ is

$$L(\mathcal{Q}) = \min_{w}\{L_w(\mathcal{Q})\},$$

where the minimum is taken over all strategies.

The load is a best-case definition, and will be achieved only if an optimal access strategy is used, and only in the case that no failures occur. A strength of this definition is that load is a property of a quorum system, and not of the protocol using it.

The following theorem was proved in [10] for all quorum systems.

Theorem 1 *Let $\mathcal{Q}$ be a quorum system over a universe of n elements. Denote by $c(\mathcal{Q})$ the size of the smallest quorum of $\mathcal{Q}$. Then $L(\mathcal{Q}) \geq \max\{\frac{1}{c(\mathcal{Q})}, \frac{c(\mathcal{Q})}{n}\}$. Consequently, $L(\mathcal{Q}) \geq \frac{1}{\sqrt{n}}$.*

Availability

The resilience f of a quorum system provides one measure of how many crash failures a quorum system is *guaranteed* to survive.

Resilience: The *resilience* f of a quorum system $\mathcal{Q}$ is the largest k such that for every set $K \subseteq U$, $|K| = k$, there exists $Q \in \mathcal{Q}$ such that $K \cap Q = \emptyset$.

Note that, the resilience f is at most $c(\mathcal{Q}) - 1$, since by disabling the members of the smallest quorum every quorum is hit. It is possible, however, that an f-resilient quorum system, though vulnerable to a few failure configurations of $f + 1$ failures, can survive many configurations of more than f failures. One way to measure this property of a quorum system is to assume that each server crashes independently with probability p and then to determine the probability F_p that no quorum remains completely alive. This is known as *failure probability* and is formally defined as follows:

Failure probability: Assume that each server in the system crashes independently with probability p. For every quorum $Q \in \mathcal{Q}$ let $\mathcal{E}_Q$ be the event that Q is *hit*, i.e., at least one element $i \in Q$ has crashed. Let crash $(\mathcal{Q})$ be the event that all the quorums $Q \in \mathcal{Q}$ were hit, i.e., crash $(\mathcal{Q}) = \bigwedge_{Q \in \mathcal{Q}} \mathcal{E}_Q$. Then the system failure probability is $F_p(\mathcal{Q}) = \Pr(\text{crash } (\mathcal{Q}))$.

Peleg and Wool study the availability of quorum systems in [11]. A good failure probability $F_p(\mathcal{Q})$ for a quorum system $\mathcal{Q}$ has $\lim_{n\to\infty} F_p(\mathcal{Q}) = 0$ when $p < \frac{1}{2}$. Note that, the failure probability of any quorum system whose resilience is f is at least $e^{-\Omega(f)}$. Majorities has the best availability when $p < \frac{1}{2}$; for $p = \frac{1}{2}$, there exist quorum constructions with $F_p(\mathcal{Q}) = \frac{1}{2}$; for $p > \frac{1}{2}$, the singleton has the best failure probability $F_p(\mathcal{Q}) = p$, but for most quorum systems, $F_p(\mathcal{Q})$ tends to 1.

The Load and Availability of Quorum Systems

Quorum constructions can be compared by analyzing their behavior according to the above measures. The singleton has a load of 1, resilience 0, and failure probability $F_p = p$. This system has the best failure probability when $p > \frac{1}{2}$, but otherwise performs poorly in both availability and load.

The system of Majorities has a load of $\lceil \frac{n+1}{2n} \rceil \approx \frac{1}{2}$. It is resilient to $\lfloor \frac{n-1}{2} \rfloor$ failures, and its failure probability is $e^{-\Omega(n)}$. This system has the highest possible resilience and asymptotically optimal failure probability, but poor load.

Grid's load is $O(\frac{1}{\sqrt{n}})$, which is within a constant factor from optimal. However, its resilience is only $\sqrt{n} - 1$ and it has poor failure probability which tends to 1 as n grows.

The resilience of a FPP quorum system is $q \approx \sqrt{n}$. The load of FPP was analyzed in [10] and shown to be $L(\text{FPP}) = \frac{q+1}{n} \approx 1/\sqrt{n}$, which is optimal. However, its failure probability tends to 1 as n grows.

As demonstrated by these systems, there is a tradeoff between load and fault tolerance in quorum systems, where the resilience f of a quorum system $\mathcal{Q}$ satisfies $f \leq nL(\mathcal{Q})$. Thus, improving one must come at the expense of the other, and it is in fact impossible to simultaneously achieve both optimally. One might conclude that good load conflicts with low failure probability, which is not necessarily the case. In fact, there exist quorum systems such as the

Paths system of Naor and Wool [10] and the Triangle Lattice of Bazzi [1] that achieve asymptotically optimal load of $O(1/\sqrt{n})$ and have close to optimal failure probability for their quorum sizes. Another construction is the CWlog system of Peleg and Wool [12], which has unusually small quorum sizes of $\log n - \log\log n$, and for systems with quorums of this size, has optimal load, $L(\text{CWlog}) = O(1/\log n)$, and optimal failure probability.

Byzantine Quorum Systems

For the most part, quorum systems were studied in environments where failures may simply cause servers to become unavailable (benign failures). But what if a server may exhibit arbitrary, possibly malicious behavior? Malkhi and Reiter [7] carried out a study of quorum systems in environments prone to arbitrary (Byzantine) behavior of servers. Intuitively, a quorum system tolerant of Byzantine failures is a collection of subsets of servers, each pair of which intersect in a set containing sufficiently many *correct* servers to mask out the behavior of faulty servers. More precisely, Byzantine quorum systems are defined as follows:

Masking quorum system

A quorum system $\mathcal{Q}$ is a *b-masking quorum system* if it has resilience $f \geq b$, and each pair of quorums intersect in at least $2b + 1$ elements.

The masking quorum system requirements enable a client to obtain the correct answer from the service despite up to b Byzantine server failures. More precisely, a write operation remains as before; to obtain the correct value of x from a read operation, the client reads a set of value/timestamp pairs from a quorum Q and sorts them into clusters of identical pairs. It then chooses a value/timestamp pair that is returned from at least $b + 1$ servers, and therefore must contain at least one correct server. The properties of masking quorum systems guarantee that at least one such cluster exists. If more than one such cluster exists, the client chooses the one with the highest timestamp. It is easy to see that any value so obtained was written before, and moreover, that the most recently written value is obtained. Thus, the semantics of a multi-writer multi-reader safe variable are obtained (see ▸ Linearizability) in a Byzantine environment.

For a b-masking quorum system, the following lower bound on the load holds:

Theorem 2 *Let $\mathcal{Q}$ be a b-masking quorum system. Then $L(\mathcal{Q}) \geq \max\{\frac{2b+1}{c(\mathcal{Q})}, \frac{c(\mathcal{Q})}{n}\}$, and consequently $L(\mathcal{Q}) \geq \sqrt{\frac{2b+1}{n}}$.*

This bound is tight, and masking quorum constructions meeting it were shown.

Malkhi and Reiter explore in [7] two variations of masking quorum systems. The first, called *dissemination quorum systems*, is suited for services that receive and distribute *self-verifying* information from correct clients (e.g., digitally signed values) that faulty servers can fail to redistribute but cannot undetectably alter. The second variation, called *opaque masking quorum systems*, is similar to regular masking quorums in that it makes no assumption of self-verifying data, but it differs in that clients do not need to know the failure scenarios for which the service was designed. This somewhat simplifies the client protocol and, in the case that the failures are maliciously induced, reveals less information to clients that could guide an attack attempting to compromise the system. It is also shown in [7] how to deal with faulty clients in addition to faulty servers.

Probabilistic Quorum Systems

The resilience of any quorum system is bounded by half of the number of servers. Moreover, as mentioned above, there is an inherent tradeoff between low load and good resilience, so that it is in fact impossible to simultaneously achieve both optimally. In particular, quorum systems over n servers that achieve the optimal load of $\frac{1}{\sqrt{n}}$ can tolerate at most $\sqrt{n}$ faults.

To break these limitations, Malkhi et al. propose in [8] to relax the intersection property of a quorum system so that "quorums" chosen according to a specified strategy intersect only with very high probability. They accordingly name

these *probabilistic quorum systems*. These systems admit the possibility, albeit small, that two operations will be performed at non-intersecting quorums, in which case consistency of the system may suffer. However, even a small relaxation of consistency can yield dramatic improvements in the resilience and failure probability of the system, while the load remains essentially unchanged. Probabilistic quorum systems are thus most suitable for use when availability of operations despite the presence of faults is more important than certain consistency. This might be the case if the cost of inconsistent operations is high but not irrecoverable, or if obtaining the most up-to-date information is desirable but not critical, while having no information may have heavier penalties.

The family of constructions suggested in [8] is as follows:

W(n, ℓ) Let U be a universe of size n. $W(n, \ell)$, $\ell \geq 1$, is the system $\langle \mathcal{Q}, w \rangle$ where $\mathcal{Q}$ is the set system $\mathcal{Q} = \{Q \subseteq U : |Q| = \ell\sqrt{n}\}$; w is an access strategy w defined by $\forall Q \in \mathcal{Q}$, $w(Q) = \frac{1}{|\mathcal{Q}|}$.

The probability of choosing according to w two quorums that do not intersect is less than $e^{-\ell^2}$, and can be made sufficiently small by appropriate choice of ℓ. Since every element is in $\binom{n-1}{\ell\sqrt{n}-1}$ quorums, the load $L(W(n, \ell))$ is $\frac{\ell}{\sqrt{n}} = O(\frac{1}{\sqrt{n}})$. Because only $\ell\sqrt{n}$ servers need be available in order for some quorum to be available, $W(n, \ell)$ is resilient to $n - \ell\sqrt{n}$ crashes. The failure probability of $W(n, \ell)$ is less than $e^{-\Omega(n)}$ for all $p \leq 1 - \frac{\ell}{\sqrt{n}}$, which is asymptotically optimal. Moreover, if $\frac{1}{2} \leq p \leq 1 - \frac{\ell}{\sqrt{n}}$, this probability is provably better than any (non-probabilistic) quorum system.

Relaxing consistency can also provide dramatic improvements in environments that may experience Byzantine failures. More details can be found in [8].

Applications

Just about any fault tolerant distributed protocol, such as Paxos [5] or consensus [1] implicitly builds on quorums, typically majorities. More concretely, scalable data repositories were built, such as Fleet [9], Rambo [4], and Rosebud [13].

Cross-References

▸ Concurrent Programming, Mutual Exclusion

Recommended Reading

1. Dwork C, Lynch N, Stockmeyer L (1988) Consensus in the presence of partial synchrony. J Assoc Comput Mach 35:288–323
2. Garcia-Molina H, Barbara D (1985) How to assign votes in a distributed system. J ACM 32:841–860
3. Gifford DK (1979) Weighted voting for replicated data. In: Proceedings of the 7th ACM symposium on operating systems principles, pp 150–162
4. Gilbert S, Lynch N, Shvartsman A (2003) Rambo ii: rapidly reconfigurable atomic memory for dynamic networks. In: Proceedings if the IEEE 2003 international conference on dependable systems and networks (DNS), San Francisco, pp 259–268
5. Lamport L (1998) The part-time parliament. ACM Trans Comput Syst 16:133–169
6. Maekawa M (1985) A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. ACM Trans Comput Syst 3(2):145–159
7. Malkhi D, Reiter M (1998) Byzantine quorum systems. Distr Comput 11:203–213
8. Malkhi D, Reiter M, Wool A, Wright R (2001) Probabilistic quorum systems. Inf Comput J 170:184–206
9. Malkhi D, Reiter MK (2000) An architecture for survivable coordination in large-scale systems. IEEE Trans Knowl Data Eng 12:187–202
10. Naor M, Wool A (1998) The load, capacity and availability of quorum systems. SIAM J Comput 27:423–447
11. Peleg D, Wool A (1995) The availability of quorum systems. Inf Comput 123:210–223
12. Peleg D, Wool A (1997) Crumbling walls: a class of practical and efficient quorumsystems. Distrib Comput 10:87–98
13. Rodrigues R, Liskov B (2003) Rosebud: a scalable Byzantine-fault tolerant storage architecture. In: Proceedings of the 18th ACM symposium on operating system principles, San Francisco
14. Thomas RH (1979) A majority consensus approach to concurrency control for multiple copy databases. ACM Trans Database Syst 4:180–209

R

Radiocoloring in Planar Graphs

Vicky Papadopoulou
Department of Computer Science, University of Cyprus, Nicosia, Cyprus

Keywords

λ-coloring; k-coloring; Coloring the square of the graph; Distance-2 coloring

Years and Authors of Summarized Original Work

2005; Fotakis, Nikoletseas, Papadopoulou, Spirakis

Problem Definition

Consider a graph $G(V, E)$. For any two vertices $u, v \in V$, $d(u, v)$ denotes the distance of u, v in G. The general problem concerns a coloring of the graph G and it is defined as follows:

Definition 1 (k-coloring problem)
INPUT: A graph $G(V, E)$.
OUTPUT: A function $\phi : V \to \{1, \ldots, \infty\}$, called k *-coloring of* G such that $\forall u, v \in V$, $x \in \{0, 1, \ldots, k\}$: if $d(u, v) \geq k - x + 1$ then $|\phi(u) - \phi(v)| = x$.

OBJECTIVE: Let $|\phi(V)| = \lambda_\phi$. Then λ_ϕ is the *number of colors* that φ actually uses (it is usually called *order* of G under φ). The number $v_\phi = max_{v \in V}\phi(v) - min_{u \in V}\phi(u) + 1$ is usually called the *span* of G under φ. The function φ satisfies one of the following objectives:

- minimum span: λ_ϕ is the minimum possible over all possible functions φ of G;
- minimum order: v_ϕ is the minimum possible over all possible functions φ of G;
- Min span order: obtains a minimum span and moreover, from all minimum span assignments, φ obtains a minimum order.
- Min order span: obtains a minimum order and moreover, from all minimum order assignments, φ obtains a minimum span.

Note that the case $k = 1$ corresponds to the well known problem of *vertex graph coloring*. Thus, k-coloring problem (with k as an input) is $\mathcal{NP}$-complete [4]. The case of k-coloring problem where $k = 2$, is called the *Radiocoloring problem*.

Definition 2 (Radiocoloring Problem (RCP) [7])
INPUT: A graph $G(V, E)$.
OUTPUT: A function $\Phi : V \to N^*$ such that $|\Phi(u) - \Phi(v)| \geq 2$ if $d(u, v) = 1$ and $|\Phi(u) - \Phi(v)| \geq 1$ if $d(u, v) = 2$.
OBJECTIVE: The least possible number (order) needed to radiocolor G is denoted

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

by $X_{\text{order}}(G)$. The least possible number $\max_{v \in V} \Phi(v) - \min_{u \in V} \Phi(u) + 1$ (span) needed for the radiocoloring of G is denoted as $X_{\text{span}}(G)$. Function Φ satisfies one of the followings:

- Min span RCP: Φ obtains a minimum span, i.e., $\lambda_\Phi = X_{span}(G)$;
- Min order RCP: Φ obtains a minimum order $\nu_\Phi = X_{\text{order}}(G)$;
- Min span order RCP: obtains a minimum span and moreover, from all minimum span assignments, Φ obtains a minimum order.
- Min order span RCP: obtains a minimum order and moreover, from all minimum order assignments, Φ obtains a minimum span.

A related to the RCP problem concerns to the square of a graph G, which is defined as follows:

Definition 3 Given a graph $G(V, E)$, G^2 is the graph having the same vertex set V and an edge set $E' : \{u, v\} \in E'$ iff $d(u, v) \leq 2$ in G.

The related problem is to color the square of a graph G, G^2 so that no two neighbor vertices (in G^2) get the same color. The objective is to use a minimum number of colors, denoted as $\chi(G^2)$ and called *chromatic number of the square of the graph G*. Fotakis et al. [5, 6] first observed that for any graph G, $X_{\text{order}}(G)$ is the same as the (vertex) chromatic number of G^2, i.e., $X_{\text{order}}(G) = \chi(G^2)$.

Key Results

Fotakis et al. [5, 6] studied *min span order*, *min order* and *min span* RCP in *planar* graph G. A planar graph, is a graph for which its edges can be embedded in the plane without crossings. The following results are obtained:

- It is first shown that the number of colors used in the *min span order RCP* of graph G is different from the chromatic number of the square of the graph, $\chi(G^2)$. In particular, it may be greater than $\chi(G^2)$.
- It is then proved that the radiocoloring problem for general graphs is hard to approximate (unless $\mathcal{NP} = ZPP$, the class of problems with polynomial time zero-error randomized algorithms) within a factor of $n^{1/2-\epsilon}$ (for any $\epsilon > 0$), where n is the number of vertices of the graph. However, when restricted to some special cases of graphs, the problem becomes easier.

  It is shown that the *min span RCP*and *min span order RCP* are $\mathcal{NP}$ *-complete* for planar graphs. Note that few combinatorial problems remain hard for *planar* graphs and their proofs of hardness are not easy since they have to use planar gadgets which are difficult to find and understand.
- It presents a $O(n\Delta(G))$ time algorithm that *approximates* the min order of RCP, X_{order}, of a planar graph G *by a constant ratio which tends to 2* as the maximum degree $\Delta(G)$ of G increases.

  The algorithm presented is motivated by a constructive coloring theorem of Heuvel and McGuiness [9]. The construction of [9] can lead (as shown) to an $O(n^2)$ technique assuming that a planar embedding of G is given. Fotakis et al. [5, 6] improves the time complexity of the approximation, and presents a much more simple algorithm to verify and implement. The algorithm does not need any planar embedding as input.
- Finally, the work considers the problem of *estimating the number of different radiocolorings* of a planar graph G. This is a $\#\mathcal{P}$-complete problem (as can be easily seen from the completeness reduction presented there that can be done parsimonious). They authors employ here standard techniques of rapidly mixing Markov Chains and the *new method of coupling* for purposes of proving *rapid convergence* (see e.g., [10]) and present *a fully polynomial randomized approximation scheme* for estimating the number of radiocolorings with λ colors for a planar graph G, when $\lambda \geq 4\Delta(G) + 50$.

In [8] and [7] it has been proved that the problem of min span RCP is $\mathcal{NP}$-complete, even

for graphs of diameter 2. The reductions use highly non-planar graphs. In [11] it is proved that the problem of coloring the square of a general graph is $\mathcal{NP}$-complete.

Another variation of RCP for planar graphs, called *distance-2-coloring* is studied in [12]. This is the problem of coloring a given graph G with the minimum number of colors so that the vertices of distance *at most* two get different colors. Note that this problem is equivalent to coloring the square of the graph G, G^2. In [12] it is proved that the distance-2-coloring problem for planar graphs is $\mathcal{NP}$-complete. As it is shown in [5, 6], this problem is different from the min span order RCP. Thus, the $\mathcal{NP}$-completeness proof in [12] certainly does not imply the $\mathcal{NP}$-completeness of min span order RCP proved in [5, 6]. In [12] a 9-approximation algorithm for the distance-2-coloring of planar graphs is also provided.

Independently and in parallel, Agnarsson and Halldórsson in [1] presented approximations for the chromatic number of square and power graphs (G^k). In particular they presented an 1.8-approximation algorithm for coloring the square of a planar graph of large degree ($\Delta(G) \geq 749$). Their method utilizes the notion of *inductiveness* of the square of a planar graph.

Bodlaender et al. in [2] proved also independently and and in parallel that the min span RCP, called λ-labeling there, is $\mathcal{NP}$-complete for planar graphs, using a similar to the approach used in [5, 6]. In the same work the authors presented approximations for the problem for some interesting families of graphs: outerplanar graphs, graphs of bounded treewidth, permutation and split graphs.

Applications

The Frequency Assignment Problem (FAP) in radio networks is a well-studied, interesting problem, aiming at assigning frequencies to transmitters exploiting frequency reuse while keeping signal interference to acceptable levels. The interference between transmitters are modeled by an interference graph $G(V, E)$, where V ($|V| = n$) corresponds to the set of transmitters and E represents distance constraints (e.g., if two neighbor nodes in G get the same or close frequencies then this causes unacceptable levels of interference). In most real life cases the network topology formed has some special properties, e.g., G is a lattice network or a planar graph. Planar graphs are mainly the object of study in [5, 6].

The FAP is usually modeled by variations of the graph coloring problem. The set of colors represents the available frequencies. In addition, each color in a particular assignment gets an integer value which has to satisfy certain inequalities compared to the values of colors of nearby nodes in G (frequency-distance constraints). A discrete version of FAP is the k-coloring problem, of which a particular instance, for $k = 2$, is investigated in [5, 6].

Real networks reserve bandwidth (range of frequencies) rather than distinct frequencies. In this case, an assignment seeks to use as small range of frequencies as possible. It is sometimes desirable to use as few distinct frequencies of a given bandwidth (span) as possible, since the unused frequencies are available for other use. However, there are cases where the primary objective is to minimize the number of frequencies used and the span is a secondary objective, since we wish to avoid reserving unnecessary large span. These realistic scenaria directed researchers to consider optimization versions of the RCP, where one aims in minimizing the span (bandwidth) or the order (distinct frequencies used) of the assignment. Such optimization problems are investigated in [5, 6].

Cross-References

- Channel Assignment and Routing in Multi-radio Wireless Mesh Networks
- Graph Coloring

Recommended Reading

1. Agnarsson G, Halldórsson MM (2000) Coloring powers of planar graphs. In: Proceedings of the 11th annual ACM-SIAM symposium on discrete algorithms, pp 654–662

2. Bodlaender HL, Kloks T, Tan RB, van Leeuwen J (2000) Approximations for λ-coloring of graphs. In: Proceedings of the 17th annual symposium on theoretical aspects of computer science. Lecture notes in computer science, vol 1770. Springer, pp 395–406
3. Hale WK (1980) Frequency assignment: theory and applications. Proc IEEE 68(12):1497–1514
4. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman
5. Fotakis D, Nikoletseas S, Papadopoulou V, Spirakis P (2000) NP completeness results and efficient approximations for radiocoloring in planar graphs. In: Proceedings of the 25th international symposium on mathematical foundations of computer science. Lecture notes of computer science, vol 1893. Springer, pp 363–372
6. Fotakis D, Nikoletseas S, Papadopoulou VG, Spirakis PG (2005) Radiocoloring in planar graphs: complexity and approximations. Theor Comput Sci Elsevier 340:514–538
7. Fotakis D, Pantziou G, Pentaris G, Spirakis P (1999) Frequency assignment in mobile and radio networks. In: Networks in distributed computing. DIMACS series in discrete mathematics and theoretical computer science, vol 45, pp 73–90
8. Griggs J, Liu D (1998) Minimum span channel assignments. In: Recent advances in radio channel assignments. Invited Minisymposium, Discrete Mathematics
9. van d Heuvel J, McGuiness S (1999) Colouring the square of a planar graph. CDAM research report series, July 1999
10. Jerrum M (1994) A very simple algorithm for estimating the number of k-colourings of a low degree graph. Random Struct Algorithm 7:157–165
11. Lin YL, Skiena S (1995) Algorithms for square roots of graphs. SIAM J Discret Math 8:99–118
12. Ramanathan S, Loyd ER (1992) The complexity of distance 2- coloring. In: Proceedings of the 4th international conference of computing and information, pp 71–74

Random Planted 3-SAT

Abraham Flaxman
Theory Group, Microsoft Research, Redmond, WA, USA

Keywords

Constraint satisfaction

Years and Authors of Summarized Original Work

2003; Flaxman

Problem Definition

This classic problem in complexity theory is concerned with efficiently finding a satisfying assignment to a propositional formula. The input is a formula with n Boolean variables which is expressed as an AND of ORs with 3 variables in each OR clause (a *3-CNF formula*). The goal is to (1) find an assignment of variables to TRUE and FALSE so that the formula has value TRUE or (2) prove that no such assignment exists. Historically, recognizing satisfiable 3-CNF formulas was the first "natural" example of an NP-complete problem, and, because it is NP-complete, no polynomial-time algorithm can succeed on all 3-CNF formulas unless P = NP [4, 10]. Because of the numerous practical applications of 3-SAT, and also due to its position as the canonical NP-complete problem, many heuristic algorithms have been developed for solving3-SAT, and some of these algorithms have been analyzed rigorously on random instances.

Notation

A 3-CNF formula over variables $x_1, x_2, \ldots, x_n$ is the conjunction of m clauses $C_1 \wedge C_2 \wedge \ldots \wedge C_m$, where each clause is the disjunction of 3 literals, $C_i = \ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}$, and each literal ℓ_{i_j} is either a variable or the negation of a variable (the negation of the variable x is denoted by $\bar{x}$). A 3-CNF formula is *satisfiable* if and only if there is an assignment of variables to truth values so that every clause contains at least one true literal. Here, all asymptotic analysis is in terms of n, the number of variables in the 3-CNF formula, and a sequence of events $\{\mathcal{E}_n\}$ is said to hold *with high probability* (abbreviated **whp**) if $\lim_{n\to\infty} \Pr[\mathcal{E}_n] = 1$.

Distributions

There are many distributions over 3-CNF formulas which are interesting to consider, and this

chapter focuses on dense satisfiable instances. Dense satisfiable instances can be formed by conditioning on the event $\{I_{n,m}$ is satisfiable$\}$, but this conditional distribution is difficult to sample from and to analyze. This has led to research in "planted" random instances of 3-SAT, which are formed by first choosing a truth assignment φ uniformly at random and then selecting each clause independently from the triples of literals where at least one literal is set to TRUE by the assignment φ. The clauses can be included with equal probabilities in analogy to the $\mathbb{I}_{n,p}$ or $\mathbb{I}_{n,m}$ distributions above [8, 9], or different probabilities can be assigned to the clauses with one, two, or three literals set to TRUE by φ, in an effort to better hide the satisfying assignment [2, 7].

Problem 1 (3-SAT)

INPUT: *3-CNF Boolean formula* $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$*, where each clause* C_i *is of the form* $C_i = \ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}$ *and each literal* ℓ_{i_j} *is either a variable or the negation of a variable.*

OUTPUT: *A truth assignment of variables to Boolean values which makes at least one literal in each clause TRUE or a certificate that no such assignment exists.*

Key Results

A line of basic research dedicated to identifying hard search and decision problems, as well as the potential cryptographic applications of planted instances of 3-SAT, has motivated the development of algorithms for 3-SAT which are known to work on planted random instances.

Majority Vote Heuristic: If every clause consistent with the planted assignment is included with the same probability, then there is a bias towards including the literal satisfied by the planted assignment more frequently than its negation. This is the motivation behind the majority vote heuristic, which assigns each variable to the truth value which will satisfy the majority of the clauses in which it appears. Despite its simplicity, this heuristic has been proven successful **whp** for sufficiently dense planted instances [8].

Theorem 1 *When* c *is a sufficiently large constant and* $I \sim \mathbb{I}^{\phi}_{n,cn\log n}$*,* ***whp*** *the majority vote heuristic finds the planted assignment* φ*.*

When the density of the planted random instance is lower than $c \log n$*, then the majority vote heuristic will fail, and if the relative probability of the clauses satisfied by one, two, and three literals is adjusted appropriately, then it will fail miserably. But there are alternative approaches.*

For planted instances where the density is a sufficiently large constant, the majority vote heuristic provides a good starting assignment, and then the k-OPT heuristic can finish the job. The k-OPT heuristic of [6] is defined as follows: Initialize the assignment by majority vote. Initialize k to 1. While there exists a set of k variables for which flipping the values of the assignment will (1) make false clauses true and (2) will not make true clauses false, flip the values of the assignment on these variables. If this reaches a local optimum that is not a satisfying assignment, increase k and continue.

Theorem 2 *When* c *is a sufficiently large constant and* $I \sim I^{\phi}_{n,cn}$*, the k-OPT heuristic finds a satisfying assignment in polynomial time* ***whp****. The same is true even in the semi-random case, where an adversary is allowed to add clauses to* I *that have all three literals set to TRUE by* φ *before giving the instance to the k-OPT heuristic.*

A related algorithm has been shown to run in expected polynomial time in [9], and a rigorous analysis of *warning propagation (WP)*, a message passing algorithm related to survey propagation, has shown that WP is successful **whp** on planted satisfying assignments, provided that the clause density exceeds a sufficiently large constant [5].

When the relative probabilities of clauses containing one, two, and three literals are adjusted carefully, it is possible to make the majority vote assignment very different from the planted assignment. A way of setting these relative probabilities that is predicted to be difficult is discussed in [2]. If the density of these instances is high

enough (and the relative probabilities are anything besides the case of "Gaussian elimination with noise"), then a spectral heuristic provides a starting assignment close to the planted assignment and local reassignment operations are sufficient to recover a satisfying assignment [7].

More formally, consider instance $I = I_{n,p1,p2,p3}$, formed by choosing a truth assignment φ on n variables uniformly at random and including in I each clause with exactly i literals satisfied by φ independently with probability p_i. By setting $p_1 = p_2 = p_3$, this reduces to the distribution mentioned above.

Setting $p_1 = p_2$ and $p_3 = 0$ yields a natural distribution on 3CNFs with a planted not-all-equal assignment, a situation where the greedy variable assignment rule generates a random assignment. Setting $p_2 = p_3 = 0$ gives 3CNFs with a planted exactly-one-true assignment (which succumb to the greedy algorithm followed by the nonspectral steps below). Also, correctly adjusting the ratios of p_1, p_2, and p_3 can obtain a variety of (slightly less natural) instance distributions which thwart the greedy algorithm. Carefully selected values of p_1, p_2, and p_3 are considered in [2], where it is conjectured that no algorithm running in polynomial time can solve $I_{n,p1,p2,p3}$ **whp** when $p_i = c_i\alpha/n^2$ and

$$0.007 < c_3 < 0.25 \qquad c_2 = (1-4c_3)/6$$
$$c_1 = (1+2c_3)/6 \quad \alpha > \frac{4.25}{7}.$$

The *spectral heuristic* modeled after the coloring algorithms of [1, 3] was developed for such planted distributions in [7]. This polynomial time algorithm which returns a satisfying assignment to $I_{n,p1,p2,p3}$ **whp** when $p_1 = d/n^2$, $p_2 = \eta_2 d/n^2$, and $p_3 = \eta_3 d/n^2$, for $0 \le \eta_2, \eta_3 \le 1$, and $d \ge d_{\min}$, where $d_{\min}$ is a function of η_2, η_3. The algorithm is structured as follows:

1. Construct a graph G from the 3CNF.
2. Find the most negative eigenvalue of a matrix related to the adjacency matrix of G.
3. Assign a value to each variable based on the signs of the eigenvector corresponding to the most negative eigenvalue.
4. Iteratively improve the assignment.
5. Perfect the assignment by exhaustive search over a small set containing all the incorrect variables.

A more elaborate description of each step is the following:

Step (1): Given 3CNF $I = I_{n,p1,p2,p3}$, where $p_1 = d/n^2$, $p_2 = \eta_2 d/n^2$, and $p_3 = \eta_3 d/n^2$, the graph in step (1), $G = (V, E)$, has $2n$ vertices, corresponding to the literals in I, and labeled $\{x_1, \bar{x}_1, \ldots x_n, \bar{x}_n\}$. G has an edge between vertices ℓ_i and ℓ_j if I includes a clause with both ℓ_i and ℓ_j (and G does not have multiple edges).

Step (2): Consider $G' = (V, E')$, formed by deleting all the edges incident to vertices with degree greater than $180d$. Let A be the adjacency matrix of G'. Let λ be the most negative eigenvalue of A and $\mathbf{v}$ be the corresponding eigenvector.

Step (3): There are two assignments to consider, π_+, which is defined by

$$\pi_+(x_i) = \begin{cases} T, \text{ if } \mathbf{v}_i \ge 0; \\ F, \text{ otherwise;} \end{cases}$$

and π_-, which is defined by

$$\pi_-(x) = \neg\pi_+(x).$$

Let π_0 be the better of π_+ and π_- (i.e., the assignment which satisfies more clauses). It can be shown that π_0 agrees with φ on at least $(1 - C/d)n$ variables for some absolute constant C.

Step (4): For $i = 1, \ldots, \log n$, do the following: for each variable x, if x appears in $5\varepsilon d$ clauses unsatisfied by π_{i-1}, then set $\pi_i(x) = \neg\pi_{i-1}(x)$, where ε is an appropriately chosen constant (taking $\varepsilon = 0.1$ works); otherwise set $\pi_i(x) = \pi_{i-1}(x)$.

Step (5): Let $\pi'_0 = \pi_{\log n}$ denote the final assignment generated in step (4). Let $\mathcal{A}_4^{\pi'_0}$ be

the set of variables which do not appear in $(3 \pm 4\varepsilon)d$ clauses as the only true literal with respect to assignment π'_0, and let $\mathcal{B}$ be the set of variables which do not appear in $(\mu_D \pm \varepsilon)d$ clauses, where $\mu_D d = (3+6)d + (6+3)\eta_2 d + 3\eta_3 d + \mathcal{O}(1/n)$ is the expected number of clauses containing variable x. Form partial assignment π'_1 by unassigning all variables in $\mathcal{A}_4^{\pi'_0}$ and $\mathcal{B}$. Now, for $i \geq 1$, if there is a variable x_i which appears in less than $(\mu_D - 2\varepsilon)d$ clauses consisting of variables that are all assigned by π'_i, then let π'_{i+1} be the partial assignment formed by unassigning x_i in π'_i. Let π' be the partial assignment when this process terminates. Consider the graph Γ with a vertex for each variable that is unassigned in π' and an edge between two variables if they appear in a clause together. If any connected component in Γ is larger than $\log n$, then fail. Otherwise, find a satisfying assignment for I by performing an exhaustive search on the variables in each connected component of Γ.

Theorem 3 *For any constants* $0 \leq \eta_2, \eta_3 \leq 1$, *except* $(\eta_2, \eta_3) = (0, 1)$, *there exists a constant* $d_{\min}$ *such that for any* $d \geq d_{\min}$, *if* $p_1 = d/n^2$, $p_2 = \eta_2 d/n^2$, *and* $p_3 = \eta_3 d/n^2$, *then this polynomial-time algorithm produces a satisfying assignment for random instances drawn from* $I_{n,p1,p2,p3}$ ***whp***.

Applications

3-SAT is a universal problem, and due to its simplicity, it has potential applications in many areas, including proof theory and program checking, planning, cryptanalysis, machine learning, and modeling biological networks.

Open Problems

An important direction is to develop alternative models of random distributions which more accurately reflect the type of instances that occur in the real world.

Data Sets

Sample instances of satisfiability and 3-SAT are available on the web at http://www.satlib.org/.

URL to Code

Solvers and information on the annual satisfiability solving competition are available on the web at http://www.satlive.org/.

Recommended Reading

1. Alon N, Kahale N (1997) A spectral technique for coloring random 3-colorable graphs. SIAM J Comput 26(6):1733–1748
2. Barthel W, Hartmann AK, Leone M, Ricci-Tersenghi F, Weigt M, Zecchina R (2002) Hiding solutions in random satisfiability problems: a statistical mechanics approach. Phys Rev Lett 88:188701
3. Chen H, Frieze AM (1996) Coloring bipartite hypergraphs. In: Cunningham HC, McCormick ST, Queyranne M (eds) Integer programming and combinatorial optimization, 5th international IPCO conference, Vancouver, 3–5 June 1996. Lecture notes in computer science, vol 1084. Springer, pp 345–358
4. Cook S (1971) The complexity of theorem-proving procedures. In: Proceedings of the 3rd annual symposium on theory of computing, Shaker Heights, 3–5 May, pp 151–158
5. Feige U, Mossel E, Vilenchik D (2006) Complete convergence of message passing algorithms for some satisfiability problems. In: Díaz J, Jansen K, Rolim JDP, Zwick U (eds) Approximation, randomization, and combinatorial optimization. Algorithms and techniques, 9th international workshop on approximation algorithms for combinatorial optimization problems, APPROX 2006 and 10th International Workshop on Randomization and Computation, RANDOM 2006, Barcelona, 28–30 Aug 2006. Lecture notes in computer science, vol 4110. Springer, pp 339–350
6. Feige U, Vilenchik D (2004) A local search algorithm for 3-SAT. Technical report, The Weizmann Institute, Rehovat
7. Flaxman AD (2003) A spectral technique for random satisfiable 3CNF formulas. In: Proceedings of the fourteenth annual ACM-SIAM symposium on discrete algorithms, Baltimore. ACM, New York, pp 357–363
8. Koutsoupias E, Papadimitriou CH (1992) On the greedy algorithm for satisfiability. Inf Process Lett 43(1):53–55

9. Krivelevich M, Vilenchik D (2006) Solving random satisfiable 3CNF formulas in expected polynomial time. In: Proceedings of the 17th annual ACM-SIAM symposium on discrete algorithm (SODA '06), Miami. ACM
10. Levin LA (1973) Universal enumeration problems. Probl Pereda Inf 9(3):115–116

Randomization in Distributed Computing

Tushar Deepak Chandra
IBM Watson Research Center, Yorktown Heights, NY, USA

Keywords

Agreement; Byzantine agreement

Years and Authors of Summarized Original Work

1996; Chandra

Problem Definition

This problem is concerned with using the multi-writer multi-reader register primitive in the shared memory model to design a fast, wait-free implementation of consensus. Below are detailed descriptions of each of these terms.

Consensus Problems

There are n processors and the goal is to design distributed algorithms to solve the following two consensus problems for these processors.

Problem 1 (Binary consensus)
INPUT: Processor i has input bit b_i.
OUTPUT: Each processor i has output bit b'_i such that: (1) all the output bits b'_i equal the same value v; and (2) $v = b_i$ for some processor i.

Problem 2 (Id consensus)
INPUT: Processor i has a unique id u_i.
OUTPUT: Each processor i has output value u'_i such that: (1) all the output values u'_i equal the same value u; and (2) $u = u_i$ for some processor i.

Wait-Free

This result builds on extensive previous work on the shared memory model of parallel computing. Shared object types include data structures such as read/write registers and synchronization primitives such as "test and set". A shared object is said to be *wait-free* if it ensures that every invocation on the object is guaranteed a response in finite time even if some or all of the other processors in the system crash. In this problem, the existence of wait-free registers is assumed and the goal is to create a fast wait-free algorithm to solve the consensus problem. In the rest of this summary, "wait-free implementations" will be referred to simply as "implementations" i.e., the term wait-free will be omitted.

Multi-writer Multi-reader Register

Many past results on solving consensus in the shared memory model assume the existence of a single writer multi-reader register. For such a register, there is a single writer client and multiple reader clients. Unfortunately, it is easy to show that the per processor step complexity of any implementation of consensus from single writer multi-reader registers will be at least linear in the number of processors. Thus, to achieve a time efficient implementation of consensus, the more powerful primitive of a multi-writer multi-reader register must be assumed. A multi-writer multi-reader register assumes the clients of the register are multiple writers and multiple readers. It is well known that it is possible to implement such a register in the shared memory model.

The Adversary

Solving the above problems is complicated by the fact that the programmer has little control over the rate at which individual processors execute. To model this fact, it is assumed that the schedule at which processors run is picked by an adversary.

It is well-known that there is no deterministic algorithm that can solve either Binary consensus or ID consensus in this adversarial model if the number of processors is greater than 1 [6, 7]. Thus, researchers have turned to the use of randomized algorithms to solve this problem [1]. These algorithms have access to random coin flips. Three types of adversaries are considered for randomized algorithms. The *strong adversary* is assumed to know the outcome of a coin flip immediately after the coin is flipped and to be able to modify its schedule accordingly. The *oblivious adversary* has to fix the schedule before any of the coins are flipped. The *intermediate adversary* is not permitted to see the outcome of a coin flip until some process makes a choice based on that coin flip. In particular, a process can flip a coin and write the result in a global register, but the intermediate adversary does not know the outcome of the coin flip until some process reads the value written in the register.

Key Results

Theorem 1 *Assuming the existence of multi-writer multi-reader registers, there exists a randomized algorithm to solve binary consensus against an intermediate adversary with O(1) expected steps per processor.*

Theorem 2 *Assuming the existence of multi-writer multi-reader registers, there exists a randomized algorithm to solve id-consensus against an intermediate adversary with* $O(\log^2 n)$ *expected steps per processor.*

Both of these results assume that every processor has a unique identifier. Prior to this result, the fastest known randomized algorithm for binary consensus made use of single writer multiple reader registers, was robust against a strong adversary, and required $O(n \log^2 n)$ steps per processor [2]. Thus, the above improvements are obtained at the cost of weakening the adversary and strengthening the system model when compared to [2].

Applications

Binary consensus is one of the most fundamental problems in distributed computing. An example of its importance is the following result shown by Herlihy [8]: If an abstract data type X together with shared memory is powerful enough to implement wait-free consensus, then X together with shared memory is powerful enough to implement in a wait-free manner any other data structure Y. Thus, using this result, a wait-free version of any data structure can be created using only wait-free multi-writer multi-reader registers as a building block.

Binary consensus has practical applications in many areas including: database management, multiprocessor computation, fault diagnosis, and mission-critical systems such as flight control. Lynch contains an extensive discussion of some of these application areas [9].

Open Problems

This result leaves open several problems. First, it leaves open a gap on the number of steps per process required to perform randomized consensus using multi-writer multi-reader registers against the *strong* adversary. A recent result by Attiya and Censor shows an $\Omega(n^2)$ lower bound on the total number of steps for all processors with multi-writer multi-reader registers (implying $\Omega(n)$ steps per process) [3]. They also show a matching upper bound of $O(n^2)$ on the total number of steps. However, closing the gap on the per-process number of steps is still open.

Another open problem is whether there is a randomized implementation of id consensus using multi-reader multi-writer registers that is robust to the intermediate adversary and whose expected number of steps per processor is better than $O(\log^2 n)$. In particular, is a constant run time possible? Aumann in follow up work to this result was able to improve the expected run time per process to $O(\log n)$ [4]. However, to the best of the reviewer's knowledge, there have been no further improvements.

A third open problem is to close the gap on the time required to solve binary consensus against the strong adversary with a single writer multiple reader register. The fastest known randomized algorithm in this scenario requires $O(n \log^2 n)$ steps per processor [2]. A trivial lower bound on the number of steps per processor when single-writer registers are used is $\Omega(n)$. However, to the best of this reviewers knowledge, a $O(\log^2 n)$ gap still remains open.

A final open problem is to close the gap on the total work required to solve consensus with single-reader single-writer registers against an oblivious adversary. Aumann and Kapah-Levy describe algorithms for this scenario that require $O(n \log n \exp(2\sqrt{\ln n \ln(c \log n \log^* n)})$ expected total work for some constant c [5]. In particular, the total work is less than $O(n^{1+\epsilon})$ for any $\epsilon > 0$. A trivial lower bound on total work is $\Omega(n)$, but a gap remains open.

Cross-References

- Asynchronous Consensus Impossibility
- Atomic Broadcast
- Byzantine Agreement
- Implementing Shared Registers in Asynchronous Message-Passing Systems
- Optimal Probabilistic Synchronous Byzantine Agreement
- Registers
- Set Agreement
- Snapshots in Shared Memory
- Wait-Free Synchronization

Recommended Reading

1. Aspnes J (2003) Randomized protocols for asynchronous consensus. Distrib Comput 16(2–3):165–175
2. Aspnes J, Waarts O (1992) Randomized consensus in expected $o(n \log^2 n)$ operations per processor. In: Proceedings of the 33rd symposium on foundations of computer science. IEEE Computer Society, Pittsburgh, 24–26 Oct 1992. pp 137–146
3. Attiya H, Censor K (2007) Tight bounds for asynchronous randomized consensus. In: Proceedings of the symposium on the theory of computation. ACM special interest group on algorithms and computation theory (SIGACT), San Diego, 11–13 June 2007
4. Aumann Y (1997) Efficient asynchronous consensus with the weak adversary scheduler. In: Symposium on principles of distributed computing (PODC). ACM special interest group on algorithms and computation theory (SIGACT), Santa Barbara, 21–24 Aug 1997, pp 209–218
5. Aumann Y, Kapach-Levy A (1999) Cooperative sharing and asynchronous consensus using single-reader/single-writer registers. In: Proceedings of 10th annual ACM-SIAM symposium of discrete algorithms (SODA). Society for Industrial and Applied Mathematics (SIAM), Baltimore, 17–19 Jan 1999, pp 61–70
6. Dolev D, Dwork C, Stockmeyer L (1987) On the minimal synchronism needed for distributed consensus. J ACM (JACM) 34(1):77–97
7. Fischer MJ, Lynch NA, Paterson M (1983) Impossibility of distributed consensus with one faulty process. In: Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on principles of database system (PODS). Association for Computational Machinery (ACM), Atlante, 21–23 Mar 1983, pp 1–7
8. Herlihy M (1991) Wait-free synchronization. ACM Trans Program Lang Syst 13(1):124–149
9. Lynch N (1996) Distributed algorithms. Morgan Kaufmann, San Mateo

Randomized Broadcasting in Radio Networks

Alon Itai
Technion, Haifa, Israel

Keywords

Ad hoc networks; Multi-hop radio networks

Years and Authors of Summarized Original Work

1992; Reuven Bar-Yehuda, Goldreich, Itai

Problem Definition

This entry investigates deterministic and randomized protocols for achieving broadcast (dis-

tributing a message from a source to all other nodes) in arbitrary multi-hop synchronous radio networks.

The model consists of an arbitrary (undirected) network, with processors communicating in synchronous time-slots subject to the following rules. In each time-slot, each processor acts either as a *transmitter* or as a *receiver*. A processor acting as a receiver is said to receive a message in time-slot t if exactly one of its neighbors transmits in that time-slot. The message received is the one transmitted. If more than one neighbor transmits in that time-slot, a *conflict* occurs. In this case, the receiver may either get a message from one of the transmitting neighbors or get no message. It is assumed that conflicts (or "collisions") are not detected, hence a processor cannot distinguish the case in which no neighbor transmits from the case in which two or more of its neighbors transmits during that time-slot. The processors are not required to have IDs nor do they know their neighbors; in particular, the processors do not know the topology of the network.

The only inputs required by the protocol are the number of processors in the network – n, Δ – an a priori known upper bound on the maximum degree in the network, and the error bound $-\epsilon$. (All bounds are a priori known to the algorithm.)

Broadcast is a task initiated by a single processor, called the *source*, transmitting a single *message*. The goal is to have the message reach all processors in the network.

Key Results

The main result is a randomized protocol that achieves broadcast in time which is optimal up to a logarithmic factor. In particular, with probability $1-\epsilon$, the protocol achieves broadcast within $O((D+\log n/\epsilon)\cdot\log n)$ time-slots.

On the other hand, a linear lower bound on the deterministic time-complexity of broadcast is proved. Namely, any deterministic broadcast protocol requires $\Omega(n)$ time-slots, even if the network has diameter 3, and n is known to all processors. These two results demonstrate an exponential gap in complexity between randomization and determinism.

Randomized Protocols

The Procedure *Decay*

The basic idea used in the protocol is to resolve potential conflicts by randomly eliminating half of the transmitters. This process of "cutting by half" is repeated each time-slot with the hope that there will exist a time-slot with a single active transmitter. The "cutting by half" process is easily implemented distributively by letting each processor decide randomly whether to eliminate itself. It will be shown that if all neighbors of a receiver follow the elimination procedure, then, with positive probability, there exists a time slot in which exactly one neighbor transmits.

What follows is a description of the procedure for sending a message m, that is executed by each processor after receiving m:

procedure *Decay*(k, m);
repeat at most k times (but at least once!)
send m to all neighbors;
set *coin*←0 or 1 with equal probability.
until *coin* = 0.

By using elementary probabilistic arguments, one can prove:

Theorem 1 *Let y be a vertex of G. Also let $d \geq 2$ neighbors of y execute Decay during the time interval [0,k) and assume that they all start the execution at Time = 0. Then $P(k,d)$, the probability that y receives a message by Time = k, satisfies:*

1. $\lim_{k\to\infty} P(k,d) \geq \frac{2}{3}$;
2. *For* $k \geq 2\lceil \log d \rceil$, $P(k,d) > \frac{1}{2}$.

(All logarithms are to base 2.)

The expected termination time of the algorithm depends on the probability that coin = 0. Here, this probability is set to be one half. An analysis of the merits of using other probabilities was carried out by Hofri [4].

The Broadcast Protocol

The broadcast protocol makes several calls to *Decay* (k, m). By Theorem 1 (2), to ensure that the probability of a processor y receiving the message be at least 1/2, the parameter k should be at least $2\log d$ (where d is the number of neighbors sending a message to y). Since d is not known, the parameter was chosen as $k = 2\lceil \log \Delta \rceil$ (recall that Δ was defined to be an upper bound on the in-degree). Theorem 1 also requires that all participants start executing *Decay* at the same time-slot. Therefore, *Decay* is initiated only at integer multiples of $2\lceil \log \Delta \rceil$.

procedure *Broadcast*;
$k = 2\lceil \log \Delta \rceil$;
$t = 2\lceil \log(N/\epsilon) \rceil$;
Wait until receiving a message, say m;
do t times {
Wait until $(Time \bmod k) = 0$;
Decay(k, m) ;
}

A network is said to execute the *Broadcast_scheme* if some processor, denoted s, transmits an initial message and each processor executes the abovementioned *Broadcast* procedure.

Theorem 2 *Let* $T = 2D + 5\max\{\sqrt{D}, \sqrt{\log(n/\epsilon)} \cdot \sqrt{\log(n/\epsilon)}$. *Assume that Broadcast_scheme starts at Time* $= 0$. *Then, with probability* $\geq 1 - 2\epsilon$, *by time* $2\lceil \log \Delta \rceil \cdot T$ *all nodes will receive the message. Furthermore, with probability* $\geq 1 - 2\epsilon$, *all the nodes will terminate by time* $2\lceil \log \Delta \rceil \cdot (T + \lceil \log(N/\epsilon) \rceil)$.

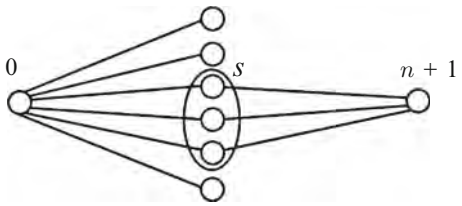
The bound provided by Theorem 2 contains two additive terms: the first represents the diameter of the network, and the second represents delays caused by conflicts (which are rare, yet they exist).

Additional Properties of the Broadcast Protocol

- **Processor IDs** – The protocol does not use processor IDs, and thus does not require that the processors have distinct IDs (or that they know the identity of their neighbors). Furthermore, a processor is not even required to know the number of its neighbors. This property makes the protocol adaptive to changes in topology which occur throughout the execution and resilient to non-malicious faults.
- **Knowing the size of the network** – The protocol performs almost as well when given instead of the actual number of processors (i.e., n), a "good" upper bound on this number (denoted N). An upper bound polynomial in n yields the same time-complexity, up to a constant factor (since complexity is logarithmic in N).
- **Conflict detection** – The algorithm and its complexity remain valid even if no messages can be received when a conflict occurs.
- **Simplicity and fast local computation** – In each time slot, each processor performs a constant amount of local computation.
- **Message complexity** – Each processor is active for $\lceil \log(N/\epsilon) \rceil$ consecutive phases, and the average number of transmissions per phase is at most 2. Thus, the expected number of transmissions of the entire network is bounded by $2n \cdot \lceil \log(N/\epsilon) \rceil$.
- **Adaptiveness to changing topology and fault resilience** – The protocol is resilient to some changes in the topology of the network. For example, edges may be added or deleted at any time, provided that the network of unchanged edges remains connected. This corresponds to fail/stop failure of edges, thus demonstrating the resilience to some non-malicious failures.
- **Directed networks** – The protocol does not use acknowledgments. Thus it may be applied even when the communication links are not symmetric, i.e., the fact that processor v can transmit to u does not imply that u can transmit to v. (The appropriate network model is, therefore, a directed graph.) In real life this situation occurs, for instance, when v has a stronger transmitter than u.

A Lower Bound on Deterministic Algorithms

For deterministic algorithms, one can show a lower bound: for every n, there exist a family of n-node networks such that every



Randomized Broadcasting in Radio Networks, Fig. 1 The network used for the lower bound

deterministic broadcast scheme requires $\Omega(n)$ time. For every **non-empty** subset $S \subseteq \{1, 2, \ldots, n\}$, consider the following network G_S (Fig. 1).

Node 0 is the *source* and node $n + 1$ the *sink*. The source initiates the message and the problem of broadcast in G_S is to reach the sink. The difficulty stems from the fact that the partition of the middle layer (i.e., S) is not known a priori. The following theorem can be proved by a series of reductions to a certain "hitting game":

Theorem 3 *Every deterministic broadcast protocol that is correct for all n-node networks requires time* $\Omega(n)$.

The result of [2] depends crucially on the assumption that the nodes do not know the number and IDs of their neighbors. If this restriction is lifted, Kowalski and Pelc [5] showed how to broadcast in logarithmic time on all networks of type G_S. Moreover, they show how to broadcast in sublinear time on all n-node graphs of diameter o (log log n).

Kowalski and Pelc also constructed a class of graphs of diameter 4, such that every broadcasting algorithm requires time $\boldsymbol{\Omega}\left(\sqrt[4]{n}\right)$ on one of these graphs. Thus they showed an exponential gap for their model too.

Applications

The procedure *Decay* has been used to resolve contention in radio and cellular phone networks.

Cross-References

- ▸ Broadcasting in Geometric Radio Networks
- ▸ Deterministic Broadcasting in Radio Networks
- ▸ Randomized Gossiping in Radio Networks

Recommended Reading

Subsequent papers showed the optimality of the randomized algorithm:

- Alon et al. [1] showed the existence of a family of radius-2 networks on n vertices for which any broadcast schedule requires at least $\Omega(\log^2 n)$ time slots.
- Kushilevitz and Mansour [7] showed that for any randomized broadcast protocol, there exists a network in which the expected time to broadcast a message is $\Omega(D \log(N/D))$.
- Bruschi and Del Pinto [3] showed that for any deterministic distributed broadcast algorithm, any n and $D \leq n/2$ there exists a network with n nodes and diameter D such that the time needed for broadcast is $\Omega(D \log n)$.
- Kowalski and Pelc [6] discussed networks in which collisions are indistinguishable from the absence of transmission. They showed an $\Omega(n \log n / \log(n/D))$ lower bound and an $O(n \log n)$ upper bound. For this model, they also showed an $O(D \log n + \log^2 n)$ randomized algorithm, thus matching the lower bound of [1] and improving the bound of [2] for graphs for which $D = \theta(n/\log n)$.

1. Alon N, Bar-Noy A, Linial N, Peleg D (1991) A lower bound for radio broadcast. J Comput Syst Sci 43(2):290–298
2. Bar-Yehuda R, Goldreich O, Itai A (1992) On the time-complexity of broadcast in multi-hop radio networks: an exponential gap between determinism and randomization. J Comput Syst Sci 45(1): 104–126
3. Bruschi D, Del Pinto M (1997) Lower bounds for the broadcast problem in mobile radio networks. Distrib Comput 10(3):129–135
4. Hofri M (1987) A feedback-less distributed broadcast algorithm for multihop radio networks with time-varying structure. In: Computer performance and reliability: 2nd international models of communication sys-

tem, Rome, 25-29 May 1987. North Holland (1988), pp 353–368
5. Kowalski DR, Pelc A (2002) Deterministic broadcasting time in radio networks of unknown topology. In: FOCS'02: proceedings of the 43rd symposium on foundations of computer science, Washington, DC. IEEE Computer Society, pp 63–72
6. Kowalski DR, Pelc A (2005) Broadcasting in undirected ad hoc radio networks. Distrib Comput 18(1):43–57
7. Kushilevitz E, Mansour Y (1993) An $\Omega(d \log(n/d))$ lower bound for broadcast in radio networks. In: PODC'93, Los Angeles, California, pp 65–74

Randomized Contraction

Marek Cygan
Institute of Informatics, University of Warsaw, Warsaw, Poland

Keywords

Cuts; Graphs; Parameterized complexity; Randomization

Years and Authors of Summarized Original Work

2012; Chitnis, Cygan, Hajiaghayi, Pilipczuk, Pilipczuk

Problem Definition

The randomized contractions framework is often useful when designing fixed-parameter-tractable (FPT) algorithms for graph cut problems. Let us assume that we are given an undirected graph G with n vertices and m edges together with an integer k. The goal is to remove at most k edges or at most k vertices, in the edge- and vertex-deletion variants of a problem, respectively, to satisfy some problem-specific constraints. In this entry, for the sake of simplicity, we restrict our attention to edge-deletion variants only.

Examples of problems that fit in the above graph cut problem class include:

Multiway Cut
Input: an undirected graph G, a set of terminals $T \subseteq V(G)$, and an integer k.
Question: is there a set $X \subseteq E(G)$ of at most k edges of G, so that in $G \setminus X$, no connected component contains more than one terminal from T?

Steiner Cut
Input: an undirected graph G, a set of terminals $T \subseteq V(G)$, and integers k, s.
Question: is there a set $X \subseteq E(G)$ of at most k edges of G, so that in $G \setminus X$, at least s connected components contain at least one terminal from T?

Multiway Cut-Uncut
Input: an undirected graph G, a set of terminals $T \subseteq V(G)$, an equivalence relation $\mathcal{R}$ on the set T, and an integer k.
Question: is there a set $X \subseteq E(G)$ of at most k edges of G, so that for any $u, v \in T$, vertices u, v are in the same connected component of $G \setminus X$ iff $\mathcal{R}(u, v)$?

Unique Label Cover
Input: an undirected graph G, a finite alphabet Σ of size s, an integer k, for each vertex $v \in V(G)$ a set $\phi_v \subseteq \Sigma$, and for each edge $e \in E(G)$ and each its endpoint v, a partial permutation $\psi_{e,v}$ of Σ, such that if $e = uv$ then $\psi_{e,u} = \psi_{e,v}^{-1}$.
Question: is there a set $X \subseteq E(G)$ of at most k edges of G and a function $\Psi : V(G) \to \Sigma$ such that for any $v \in V(G)$ we have $\Psi(v) \in \phi_v$ and for any $uv \in E(G) \setminus X$, we have $(\Psi(u), \Psi(v)) \in \psi_{uv,u}$?

Key Results

The randomized contractions framework was obtained by Chitnis et al. [2]; however, it was inspired by an earlier work of Kawarabayashi and Thorup [4], who have shown that the k-way cut problem is fixed parameter tractable. Randomized contractions were used to obtain the first FPT algorithm for unique label cover parameterized by both the cut size and the alphabet size, as well as to improve the dependency on k in the FPT algorithms for Steiner cut and multiway cut-uncut.

To exemplify usage of randomized contractions, we use the multiway cut problem. Multiway cut is known to be FPT for a long time [5] and it admits efficient FPT algorithms with $f(k) = 4^k$ dependency on k by using important separators [1] as well as $f(k) = 2^k$ by LP-branching [3]. We use multiway cut as an illustration of usage of randomized contractions to simplify the description and magnify the most important parts of the technique.

High-Level Intuition

From now on, we assume that the given undirected graph G is connected, as otherwise one can solve the problem independently for each connected component of G. Observe that this guarantees that after removing k edges, the graph contains at most $k + 1$ connected components.

On a high level, the technique works in two phases. In the first phase, as long as the graph admits a certain type of a good edge separation, we proceed recursively and simplify the instance.

On the other hand, if the graph is well connected and does not contain a cut we are looking for, then in the second phase, we solve the problem directly, by exploiting the high connectivity of G.

Recursive Understanding

Assume that we have a set of vertices $V_1 \subseteq V(G)$, such that $G[V_1]$ is connected, V_1 contains at least $k \cdot k! + 2$ vertices, and there are at most k edges between V_1 and $V_2 = V(G) \setminus V_1$ in G. Let $B \subseteq V_1$ be the set of vertices in V_1 having at least one neighbor in V_2. In such a setting, one can show that by looking at $G[V_1]$ only (in particular without looking at $G[V_2]$), one can find an edge of $G[V_1]$ which can be safely contracted, i.e., which is not part of some solution for the whole graph G. The reason is that any solution $X \subseteq E(G)$ gives some partition of B by looking at the set of connected components o $G[V_2 \cup B] \setminus X$. There are at most $k!$ partitions of B as $|B| \leq k$. Imagine that for any such partition, we mark a set of at most k edges, which would extend the partial solution under consideration, i.e., extend $X \cap E(G[B \cup V_2])$. In total, this marking procedure would select $k \cdot k!$ edges, leaving at least one edge unmarked, as $E(G[V_1]) \geq |V_1| - 1 \geq k \cdot k! + 1$. Such an unmarked edge can be safely contracted. The intuition behind this reasoning leads to the following definition:

Definition 1 Let G be a connected graph. A partition (V_1, V_2) of $V(G)$ is called a (q, k)-*good edge separation*, if

- $|V_1|, |V_2| > q$;
- $|E(V_1, V_2)| \leq k$;
- $G[V_1]$ and $G[V_2]$ are connected.

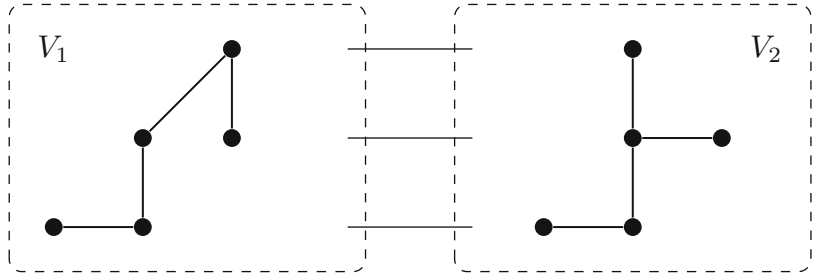
For the multiway cut problem, we would set $q = k \cdot k! + 1$. The following lemma states that we can find a (q, k)-good edge separation, if it exists:

Lemma 1 *There exists a deterministic algorithm that, given an undirected, connected graph G on n vertices along with integers q and k, in time $\mathcal{O}(2^{\mathcal{O}(\min(q,k)\log(q+k))} n^3 \log n)$ either finds a (q, k)-good edge separation or correctly concludes that no such separation exists.*

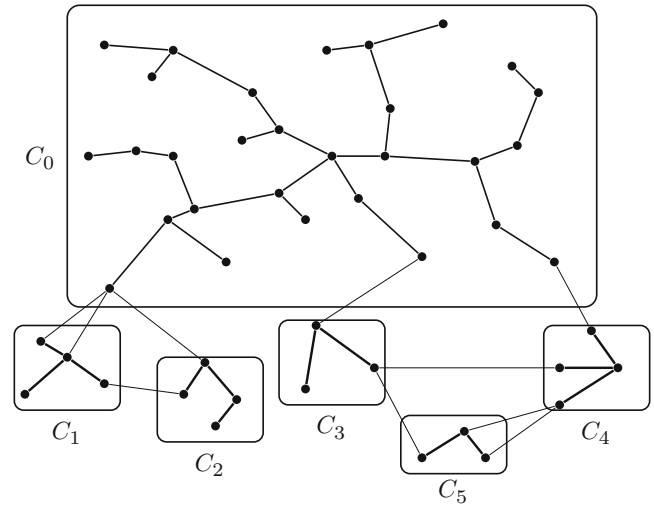
A rough sketch of the proof follows. Assume that a (q, k)-good edge separation (V_1, V_2) exists. Let E_1 be the set of edges of some subtree of $G[V_1]$ with exactly q edges; similarly let E_2 be the set of edges of some subtree of $G[V_2]$ with exactly q edges. By the definition of a (q, k)-good separation, such sets E_1, E_2 exist. Contract each edge of the graph with probability $1/2$ independently from other edges. With probability at least $1/f(k, q) = 2^{-(2q+k)}$, the following event happens (see Fig. 1):

(i) No edge between V_1 and V_2 is contracted,
(ii) All edges of $E_1 \cup E_2$ are contracted.

If we are lucky and such an event occurs, then by looking for a minimum cut between each two vertices onto which at least $q + 1$ vertices of G were contracted, we can find a (q, k)-good separation. By a better choice of contraction probability, we can improve the probability of success, whereas by using splitters [6], we can derandomize the procedure.



Randomized Contraction, Fig. 1 A (q, k)-good separation. In the randomized routine, we hope the thick edges to be contracted and the thin edges not to be contracted



Randomized Contraction, Fig. 2 Structure of a connected graph G that does not admit a (q, k)-good separation. After removing at most k edges, only one big connected component, C_0, remains

Summarizing this phase of the algorithm, we look for a (q, k)-good edge separation. If it does not exist, then we proceed to the second – high connectivity – phase of the algorithm. However, if a (q, k)-good edge separation exists, then we proceed recursively. Clearly, we are omitting some important details in this description. The most important of them is that when recursing, some vertices play a special role, as they are *border terminals* – vertices which have neighbors outside of the part of the graph under consideration. For this reason, to make the induction work, we need a stronger definition of a problem, called its border version, which for multiway cut is as follows:

Border Multiway Cut

Input: a connected, undirected graph G, a set of terminals $T \subseteq V(G)$, an integer k, and a set $T_b \subseteq V(G)$ of at most $2k$ terminals.

Output: for each partition $\mathcal{P}$ of T_b output, a set $X_{\mathcal{P}}$ of size at most k (if it exists), such that in the graph $G_{\mathcal{P}} \setminus X$, no two terminals from T are in the same connected component, where $G_{\mathcal{P}} = (V(G), E(G) \cup E_{\mathcal{P}})$ and $E_{\mathcal{P}}$ contains pairs of vertices which are in the same block of $\mathcal{P}$.

High-Connectivity Phase

The second phase of the approach is usually problem specific; however, its main idea is the following. Since we know that G does not admit a (q, k)-good edge separation, if we remove any set X of at most k edges, there is at most one connected component of $G \setminus X$ containing more than q vertices (see Fig. 2). Therefore, if we independently contract each edge at random, then with good enough probability, no solution edge

will be contracted and all connected components of $G \setminus X$ except possibly one will be contracted onto single vertices (again, see Fig. 2). In such a case, one can show that we can solve a cut problem under consideration either greedily or by dynamic programming.

Related Work

The currently best-known parameterized algorithm for unique label cover is due to Wahlström [7] and works in time $s^{2k} n^{\mathcal{O}(1)}$.

Recommended Reading

1. Chen J, Liu Y, Lu S (2009) An improved parameterized algorithm for the minimum node multiway cut problem. Algorithmica 55(1):1–13. doi:10.1007/s00453-007-9130-6
2. Chitnis RH, Cygan M, Hajiaghayi M, Pilipczuk M, Pilipczuk M (2012) Designing FPT algorithms for cut problems using randomized contractions. In: 53rd annual IEEE symposium on foundations of computer science (FOCS 2012), New Brunswick, 20–23 Oct 2012. IEEE Computer Society, pp 460–469. doi:10.1109/FOCS.2012.29
3. Cygan M, Pilipczuk M, Pilipczuk M, Wojtaszczyk JO (2013) On multiway cut parameterized above lower bounds. TOCT 5(1):3. doi:10.1145/2462896.2462899
4. Kawarabayashi K, Thorup M (2011) The minimum k-way cut of bounded size is fixed-parameter tractable. In: Ostrovsky R (ed) IEEE 52nd annual symposium on foundations of computer science (FOCS 2011), Palm Springs, 22–25 Oct 2011. IEEE Computer Society, pp 160–169. doi:10.1109/FOCS.2011.53
5. Marx D (2006) Parameterized graph separation problems. Theor Comput Sci 351(3):394–406. doi:10.1016/j.tcs.2005.10.007
6. Naor M, Schulman LJ, Srinivasan A (1995) Splitters and near-optimal derandomization. In: 36th annual symposium on foundations of computer science, Milwaukee, 23–25 Oct 1995. IEEE Computer Society, pp 182–191. doi:10.1109/SFCS.1995.492475
7. Wahlström M (2014) Half-integrality, LP-branching and FPT algorithms. In: Chekuri C (ed) Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms (SODA 2014), Portland, 5–7 Jan 2014. SIAM, pp 1762–1781. doi:10.1137/1.9781611973402.128

Randomized Energy Balance Algorithms in Sensor Networks

Pierre Leone[1], Sotiris Nikoletseas[2,3], and José Rolim[1]
[1]Informatics Department, University of Geneva, Geneva, Switzerland
[2]Computer Engineering and Informatics Department, University of Patras, Patras, Greece
[3]Computer Technology Institute and Press "Diophantus", Patras, Greece

Keywords

Power conservation

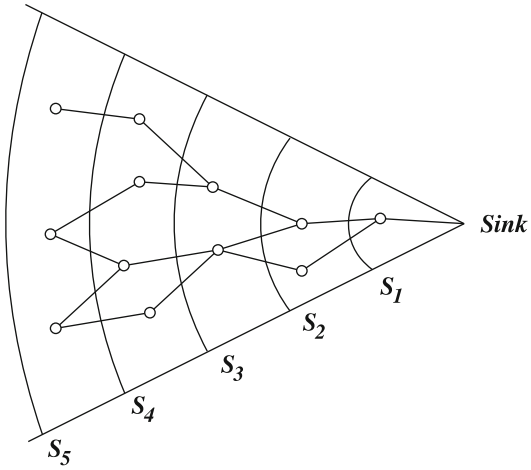
Years and Authors of Summarized Original Work

2005; Leone, Nikoletseas, Rolim

Problem Definition

Recent developments in wireless communications and digital electronics have led to the development of extremely small in size, low-power, low-cost sensor devices (often called smart dust). Such tiny devices integrate sensing, data processing and wireless communication capabilities. Examining each such resource constraint device individually might appear to have small utility; however, the distributed self-collaboration of large numbers of such devices into an ad hoc network may lead to the efficient accomplishment of large sensing tasks i.e., reporting data about the realization of a local event happening in the network area to a faraway control center.

The problem considered is the development of a randomized algorithm to balance energy among sensors whose aim is to detect events in the network area and report them to a sink. The network is sliced by the algorithm into layers composed of sensors at approximately equal distances from the



Randomized Energy Balance Algorithms in Sensor Networks, Fig. 1 The sink and five slices $S_1, \ldots, S_5$

sink [1, 2, 8] (Fig. 1). The slicing of the network depends on the communication distance. The sink initiates the process by sending a control message containing a counter, the value of which is initially 1. Sensors receiving the message assign themselves to a slice number corresponding to the counter, increment the counter and propagate the message in the network. A sensor already assigned to a slice ignores subsequent received control messages.

The strategy suggested to balance the energy among sensors consists in allowing a sensor to probabilistically choose between either sending data to a sensor in the next layer towards the sink or sending the data directly to the sink. The difference between the two choices is the energy consumption, which is much higher if the sensor decides to report to the sink directly. The energy consumption is modeled as a function of the transmission distance by assuming that the energy necessary to send data up to a distance d is proportional to d^2. Actually, more accurate models can be considered, in which the dependence is of the form d^α, with $2 \leq \alpha \leq 5$ depending on the particular environmental conditions. Although the model chosen determines the parameters of the algorithm, the particular shape of the function describing the relationship between the distance of transmission and energy consumption is not relevant except that it might increase with distance. The distance between two successive slices is normalized to be 1. Hence, a sensor sending data to one of its neighbors consumes one unit of energy and a sensor located in slice i consumes i^2 units of energy to report to the sink directly. Small hop transmissions are cheap (with respect to energy consumption) but pass through the critical region around the sink and might strain sensors in that region, while expensive direct transmissions bypass that critical area.

Energy balance is defined as follows:

Definition 1 The network is energy-balanced if the average per sensor energy dissipation is the same for all sectors, i.e., when

$$\frac{E[\mathcal{E}_i]}{S_i} = \frac{E[\mathcal{E}_j]}{S_j}, \quad i, j = 1, \ldots, n \qquad (1)$$

where $\mathcal{E}_i$ is the total energy available and S_i is the number of nodes in slice number i.

The dynamics of the network is modeled by assigning probabilities $\lambda_i, i = 1, \ldots, N$, $\sum \lambda_i = 1$, of the occurrence of an event in slice i. The protocol consists in transmitting the data to a neighbor slice with probability p_i and with probability $1 - p_i$ to the sink, for a sensor belonging to slice i. Hence, the mean energy consumption per data unit is $p_i + (1 - p_i)i^2$. A central assumption in the following is that the events are evenly generated in a given slice. Then, denoting by e_i the energy available per node in slice i (i.e., $e_i = \mathcal{E}_i / S_i$), the problem of energy-balanced data propagation can be formally stated as follows:

Given $\lambda_i, e_i, S_i, i = 1, \ldots, N$, find p_i, λ such that

$$\underbrace{\left(\lambda_i + \lambda_{i+1} p_{i+1} + \ldots + \lambda_n p_n p_{n-1} \cdots p_{i+1}\right)}_{=:x_i} \cdot \left(p_i \frac{1}{S_i} + (1 - p_i)\frac{i^2}{S_i}\right) = \lambda e_i, \quad i = 1, \ldots, N. \qquad (2)$$

```
Initialize x̃_0 = λ, ..., x̃_n
Initialize NbrLoop=1
repeat forever
   Send x̃_i and λ values to the stations which compute
                                        their p_i probability
    wait for a data
   for i=0 to n
      if the data passed through slice i then
         X ← 1
      else
         X ← 0
      end if
      Generate R a x̃_i-Bernoulli random variable
      x̃_i    x̃_i + 1/NbrLoop (X − R)
      Increment NbrLoop by one.
   end for
end repeat
```

Randomized Energy Balance Algorithms in Sensor Networks, Fig. 2 Pseudo-code for estimation of the x_i value by the sink

Equation (2) amounts to ensuring that the mean energy dissipation for all sensors is proportional to the available energy. In turn, this ensures that sensors might, on average, run out of energy all at the same time. Notice that (2) contains the definitions of the x_i. They are the ones estimated in the pseudo-code in Fig. 2, the successive estimations being denoted as $\tilde{x}_i$. These variables are proportional to the number of messages handled by slice i.

Key Results

In [1, 2] recursive equations similar to (2) were suggested and solved in closed form under adequate hypotheses. The need for a priori knowledge of the probability of occurrence of the events, the λ_i parameters, was considered in [7], in which these parameters were estimated by the sink on the basis of the observations of the various paths the data follow. The algorithm suggested is based on recursive estimation, is computationally not expensive and converges with rate $\mathcal{O}(1/\sqrt{n})$. One might argue that the rate of convergence is slow; however, it is numerically observed that relatively quickly compared with the convergence time, the algorithm finds an estimation close enough to the final value. The estimation algorithm run by the sink (which has no energy constraints) is given in Fig. 2.

Results taken from [1, 2, 7] all assume the existence of an energy-balance solution. However, particular distributions of the events might prevent the existence of such a solution and the relevant question is no longer the computation of an energy-balance algorithm. For instance, assuming that $\lambda_N = 0$, sensors in slice N have no way of balancing energy. In [9] the problem was reformulated as finding the probability distribution $\{p_i\}_{i=1,\ldots,N}$ which leads to the maximal functional lifetime of the networks. It was proved that if an energy-balance strategy exists, then it maximizes the lifetime of the network establishing formally the intuitive reasoning which was the motivation to consider energy-balance strategies. A centralized algorithm was presented to compute the optimal parameters. Moreover, it was observed numerically that the interslice energy consumption is prone to be uneven and a spreading technique was suggested and numerically validated as being efficient to overcome this limitation of the probabilistic algorithm.

The communication graph considered is a restrictive subset of the complete communication graph and it is legitimate to wonder whether one can improve the situation by extending it. For instance, by allowing data to be sent two hops or more away. In [3, 6] it was proved that the topology in which sensors communicate only to neighbor slices and the sink is the one which maximizes the flow of data in the network. Moreover, the communication graph in which sensors send data only to their neighbors and the sink leads to a completely distributed algorithm balancing energy [6]. Indeed, as a sensor sends data to a neighbor slice, the neighbor must in turn send the data and can attach information concerning its own energy level. This information might be captured by the initial sensor since it belongs to the communication range of its neighbor (this does not hold any longer if multiple hops are allowed). Hence, a distributed strategy consists in

sending data to a particular neighbor only if its energy level consumption is lower, otherwise the data are sent directly to the sink.

Applications

Among the several constraints sensor networks designers have to face, energy management is central since sensors are usually battery powered, making the lifetime of the networks highly sensitive to the energy management. Besides the traditional strategy consisting in minimizing the energy consumption at sensor nodes, energy-balance schemes aim at balancing the energy consumption among sensors. The intuitive function of such schemes is to avoid energy depletion holes appearing as some sensors that run out of their available energy resources and are no longer able to participate in the global function of the networks. For instance, routing might be no longer possible if a small number of sensors run out of energy, leading to a disconnected network. This was pointed out in [5] as well as the need to develop application-specific protocols. Energy balancing is suggested as a solution in order to make the global functional lifetime of the network longer. The earliest development of dedicated protocols ensuring energy balance can be found in [4, 10, 11].

A key application is to maximize the lifetime of the network while gathering data to a sink. Besides increasing the lifetime of the networks, other criteria have to be taken into account. Indeed, the distributed algorithm might be as simple as possible owing to limited computational resources, might avoid collisions or limit the total number of transmissions, and might ensure a large enough flow of data from the sensors toward the sink. Actually, maximizing the flow of data is equivalent to maximizing the lifetime of sensor networks if some particular realizable conditions are fulfilled. Besides the simplicity of the distributed algorithm, the network deployment and the self-realization of the network structure might be possible in realistic conditions.

Cross-References

- Obstacle Avoidance Algorithms in Wireless Sensor Networks
- Probabilistic Data Forwarding in Wireless Sensor Networks

Recommended Reading

1. Efthymiou C, Nikoletseas S, Rolim J (2006) Energy balanced data propagation in wireless sensor networks. In: 4th international workshop on algorithms for wireless, mobile, ad-hoc and sensor networks (WMAN'04) IPDPS 2004. Wirel Netw J (WINET) 12(6):691–707
2. Efthymiou C, Nikoletseas S, Rolim J (2006) Energy balanced data propagation in wireless sensor networks. In: Wireless networks (WINET) Journal, Special Issue on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks. Springer
3. Giridhar A, Kumar PR (2005) Maximizing the functional lifetime of sensor networks. In: Proceedings of the fourth international conference on information processing in sensor networks (IPSN'05). UCLA, Los Angeles, 25–27 Apr 2005
4. Guo W, Liu Z, Wu G (2003) An energy-balanced transmission scheme for sensor networks. In: 1st ACM international conference on embedded networked sensor systems (ACM SenSys 2003), Poster Session, Los Angeles, Nov 2003
5. Heinzelman W, Chandrakasan A, Balakrishnan H (2000) Energy efficient communication protocol for wireless microsensor networks. In: Proceedings of the 33rd IEEE Hawaii international conference on system sciences (HICSS 2000)
6. Jarry A, Leone P, Powell O, Rolim J (2006) An optimal data propagation algorithm for maximizing the lifespan of sensor networks. In: Second international conference, DCOSS 2006, San Francisco, June 2006. Lecture notes in computer science, vol 4026. Springer, Berlin, pp 405–421
7. Leone P, Nikoletseas S, Rolim J (2005) An adaptive blind algorithm for energy balanced data propagation in wireless sensor networks. In: First international conference on distributed computing in sensor systems (DCOSS), Marina del Rey, June/July 2005. Lecture notes in computer science, vol 3560. Springer, Berlin, pp 35–48
8. Olariu S, Stojmenovic I (2006) Design guidelines for maximizing lifetime and avoiding energy holes in sensor networks with uniform distribution and uniform reporting. In: IEEE INFOCOM, Barcelona, 24–25 Apr 2006
9. Powell O, Leone P, Rolim J (2007) Energy optimal data propagation in sensor networks. J Parallel

Distrib Comput 67(3):302–317. http://arxiv.org/abs/cs/0508052
10. Singh M, Prasanna V (2003) Energy-optimal and energy-balanced sorting in a single-hop wireless sensor network. In: Proceedings of the first IEEE international conference on pervasive computing and communications (PerCom'03), Fort Worth, 23–26 Mar 2003, pp 302–317
11. Yu Y, Prasanna VK (2003) Energy-balanced task allocation for collaborative processing in networked embedded system. In: Proceedings of the 2003 conference on language, compilers, and tools for embedded systems (LCTES'03), San Diego, 11–13 June 2003, pp 265–274

Randomized Gossiping in Radio Networks

Leszek Gąsieniec
University of Liverpool, Liverpool, UK

Keywords

All-to-all communication; Broadcast; Gossip; Total exchange of information; Wireless networks

Years and Authors of Summarized Original Work

2001; Chrobak, Gąsieniec, Rytter

Problem Definition

The two classical problems of disseminating information in computer networks are *broadcasting* and *gossiping*. In broadcasting, the goal is to distribute a message from a distinguished *source* node to all other nodes in the networks. In gossiping, each node v in the network initially contains a message m_v, and the task is to distribute each message m_v to all nodes in the network.

The radio network abstraction captures the features of distributed communication networks with multi-access channels, with minimal assumptions on the channel model and processors' knowledge. Directed edges model unidirectional links, including situations in which one of two adjacent transmitters is more powerful than the other. In particular, there is no feedback mechanism (see, for example, [6]). In some applications, collisions may be difficult to distinguish from the noise that is normally present in the channel, justifying the need for protocols that do not depend on the reliability of the collision detection mechanism (see [3, 4]). Some network configurations are subject to frequent changes. In other networks, a network topology could be unstable or dynamic, for example, when mobile users are present. In such situations, algorithms that do not assume any specific topology are more desirable.

More formally a radio network is a directed graph $G = (V, E)$, where by $|V| = n$, we denote the number of nodes in this graph. Individual nodes in V are denoted by letters $u, v, \ldots$. If there is an edge from u to v, i.e., $(u, v) \in E$, then we say that v is an *out-neighbor* of u and u is an *in-neighbor* of v. Messages are denoted by letter m, possibly with indices. In particular, the message originating from node v is denoted by m_v . The whole set of initial messages is $M = \{m_v : v \in V\}$. During the computation, each node v holds a set of messages M_v that have been received by v so far. Initially, each node v does not possess any information apart from $M_v = \{m_v\}$. Without loss of generality, whenever a node is in the transmitting mode, one can assume that it transmits the whole content of M_v.

The time is divided into discrete time steps. All nodes start simultaneously, have access to a common clock, and work synchronously. A gossiping algorithm is a protocol that for each node u, given all past messages received by u, specifies, for each time step t, whether u will transmit a message at time t, and if so, it also specifies the message. A message M transmitted at time t from a node u is sent instantly to all its out-neighbors. An out-neighbor v of u receives M at time step t only if no collision occurred, that is, if the other in-neighbors of v do not transmit

at time t at all. Further, collisions cannot be distinguished from background noise. If v does not receive any message at time t, it knows that either none of its in-neighbors transmitted at time t or that at least two did, but it does not know which of these two events occurred. The *running time* of a gossiping algorithm is the smallest t such that for any network topology, and any assignment of identifiers to the nodes, all nodes receive messages originating in every other node no later than at step t.

Limited Broadcast $_v$ (k) Given an integer k and a node v, the goal of *limited broadcasting* is to deliver the message m_v (originating in v) to at least k other nodes in the network.

Distributed Coupon Collection The set of network nodes V can be interpreted as a set of n bins and the set of messages M as a set of n coupons. Each coupon has at least k copies, each copy belonging to a different bin. M_v is the set of coupons in bin v. Consider the following process. At each step, one opens every bin at random, independently, with probability $1/n$. If no bin is opened, or if two or more bins are opened, a failure occurs and no coupons are collected. If exactly one bin, say v, is opened, all coupons from M_v are collected. The task is to establish how many steps are needed to collect (a copy of) each coupon.

Key Results

Theorem 1 ([1]) *There exists a deterministic $O(k \log^2 n)$-time algorithm for limited broadcasting from any node in radio networks with an arbitrary topology.*

Theorem 2 ([1]) *Let δ be a given constant, $0 < \delta < 1$, and $s = (4n/k)\ln(n/\delta)$. After s steps of the distributed coupon collection process, with probability at least $1 - \delta$, all coupons will be collected.*

Theorem 3 ([1]) *Let ϵ be a given constant, where $0 < \epsilon < 1$. There exists a randomized $O(n \log^3 n \log(n/\epsilon))$-time Monte Carlo-type algorithm that completes radio gossiping with probability at least $1 - \epsilon$.*

Theorem 4 ([1]) *There exists a randomized Las Vegas-type algorithm that completes radio gossiping with expected running time $O(n \log^4 n)$.*

Applications

Further work on efficient randomized radio gossiping include the $O(n \log^3 n)$-time algorithm by Liu and Prabhakaran; see [5], where the deterministic procedure for limited broadcasting is replaced by its $O(k \log n)$-time randomized counterpart. This bound was later reduced to $O(n \log^2 n)$ by Czumaj and Rytter in [2], where a new randomized limited broadcasting procedure with an expected running time $O(k)$ is proposed.

Open Problems

The exact complexity of randomized radio gossiping remains an open problem. All three gossiping algorithms [1, 2, 5] are based on the concepts of limited broadcast and distributed coupon collection. The two improvements [2, 5] refer solely to limited broadcasting. Thus, very likely further reduction of the time complexity must coincide with more accurate analysis of the distributed coupon collection process or with development of a new gossiping procedure.

Recommended Reading

1. Chrobak M, Gąsieniec L, Rytter W (2004) A randomized algorithm for gossiping in radio networks. In: Proceedings of 8th annual international computing combinatorics conference, Guilin, 2001, pp 483–492; Full version in Networks 43(2):119–124 (2004)
2. Czumaj A, Rytter W (2006) Broadcasting algorithms in radio networks with unknown topology. J Algorithms 60(2):115–143
3. Ephremides A, Hajek B (1998) Information theory and communication networks: an unconsummated union. IEEE Trans Inf Theory 44:2416–2434
4. Gallager R (1985) A perspective on multiaccess communications. IEEE Trans Inf Theory 31:124–142

5. Liu D, Prabhakaran M (2002) On randomized broadcasting and gossiping in radio networks. In: Proceedings of 8th annual international computing combinatorics conference, Singapore, pp 340–349
6. Massey JL, Mathys P (1985) The collision channel without feedback. IEEE Trans Inf Theory 31: 192–204

Randomized Minimum Spanning Tree

Vijaya Ramachandran
Computer Science, University of Texas, Austin, TX, USA

Keywords

Linear time; Minimum spanning tree; Minimum spanning tree verification; Randomized algorithm

Years and Authors of Summarized Original Work

1995; Karger, Klein, Tarjan

Problem Definition

The input to the problem is a connected undirected graph $G = (V, E)$ with a weight $w(e)$ on each edge $e \in E$. The goal is to find a spanning tree of minimum weight, where for any subset of edges $E' \subseteq E$, the *weight of* E' is defined to be $w(E') = \sum_{e \in E'} w(e)$.

If the graph G is not connected, the goal of the problem is to find a *minimum spanning forest*, which is defined to be a minimum spanning tree in each connected component of G. Both problems will be referred to as the *MST* problem.

The randomized MST algorithm by Karger, Klein, and Tarjan [9] which is considered here will be called the *KKT algorithm*. Also it will be assumed that the input graph $G = (V, E)$ has n vertices and m edges and that the edge weights are distinct.

The MST problem has been studied extensively prior to the KKT result, and several very efficient, deterministic algorithms are available from these studies. All of these are deterministic and are based on a method that greedily adds an edge to a forest that is a subgraph of the minimum spanning tree at all times. The early algorithms in this class are already efficient with a running time of $O(m \log n)$. These include the algorithms of Borůvka [1], Jarník [8] (later rediscovered by Dijkstra and Prim [5]), and Kruskal [5].

The fastest algorithm known for MST prior to the KKT algorithm runs in time $O(m \log \beta(m, n))$ [7], where $\beta(m, n) = \min\{i \mid \log^{(i)} n \leq m/n\}$ [7]; here $\log^{(i)} n$ is defined as $\log n$ if $i = 1$ and as $\log \log^{(i-1)} n$ if $i > 1$. Although this running time is close to linear, it is not linear time if the graph is very sparse.

The problem of finding the minimum spanning tree efficiently is an important and fundamental problem in graph algorithms and combinatorial optimization.

Background

Some relevant background is summarized here.

- The basic step in Borůvka's algorithm [1] is the *Borůvka step*, which picks the minimum edge-weight incident on each vertex, adds it to the minimum spanning tree, and then contracts these edges. This step runs in linear time and also very efficiently in parallel. It is the backbone of the most efficient parallel algorithms for minimum spanning tree and is also used in the KKT algorithm.
- A related and simpler problem is that of *minimum spanning tree verification*. Here, given a spanning tree T of the input edge-weighted graph, one needs to determine if T is its minimum spanning tree. An algorithm that solves this problem with a linear number of edge-weight comparisons was shown by Kom-

lós [13], and later a deterministic linear-time algorithm was given in [6] (see also [12] for a simpler algorithm).

Key Results

The main result in [9] is a randomized algorithm for the minimum spanning tree problem that runs in expected linear time. The only operations performed on the edge weights are pairwise comparisons. The algorithm does not assume any particular representation of the edge weights (i.e., integer or real values) and only assumes that any comparison between a pair of edge weights can be performed in unit time. The entry also shows that the algorithm runs in $O(m+n)$ time with the exponentially high probability $1 - \exp(-\Omega(m))$ and that its worst-case running time is $O(n + m \log n)$.

The simple and elegant *MST sampling lemma* given in Lemma 1 below is the key tool used to derive and analyze the KKT algorithm. This lemma needs a couple of definitions and facts:

1. The well-known *cycle property* for the minimum spanning tree states that the heaviest edge in any cycle in the input graph G *cannot* be in the minimum spanning tree.
2. Let F be a forest of G (i.e., an acyclic subgraph of G). An edge $e \in E$ is *F-light* if $F \cup \{e\}$ either continues to be a forest of G, or the heaviest edge in the cycle containing e is *not* e. An edge in G that is not F-light is *F-heavy*. Note that by the cycle property, an F-heavy edge cannot be in the minimum spanning tree of G, *no matter what forest F is used*. Given a forest F of G, the set of F-heavy edges can be determined in linear time by a simple modification to existing linear-time minimum spanning tree verification algorithms [6, 12].

Lemma 1 (MST Sampling Lemma) *Let $H = (V, E_H)$ be formed from the input edge-weighted graph $G = (V, E)$ by including each edge with probability p independent of the other edges. Let F be the minimum spanning forest of H. Then, the expected number of F-light edges in G is $\leq n/p$.*

The KKT algorithm identifies edges in the minimum spanning tree of G only using Borůvka steps. However, after every two Borůvka steps, it removes F-heavy edges using the minimum spanning forest F of a subgraph obtained through sampling edges with probability $p = 1/2$. As mentioned earlier, these F-heavy edges can be identified in linear time. The minimum spanning forest of the sampled graph is computed recursively.

The correctness of the KKT algorithm is immediate since every F-heavy edge it removes cannot be in the MST of G since F is a forest of G, and every edge it adds to the minimum spanning tree is in the MST since it is added through a Borůvka step.

The expected running time analysis as well as the exponentially high probability bound for the running time are surprisingly simple to derive using the MST Sampling Lemma (Lemma 1).

In summary, the entry [9] proves the following results.

Theorem 1 *The KKT algorithm is a randomized algorithm that finds a minimum spanning tree of an edge-weighted undirected graph on n nodes and m edges in $O(n + m)$ time with probability at least $1 - \exp(-\Omega(m))$. The expected running time is $O(n+m)$ and the worst-case running time is $O(n + m \log n)$.*

The model of computation used in [9] is the unit-cost RAM model since the known MST verification algorithms were for this model and not the more restrictive *pointer machine* model. More recently the MST verification result and hence the KKT algorithm have been shown to work on the pointer machine as well [2].

Lemma 1 is proved in [9] through a simulation of Kruskal's algorithm along with an analysis of the probability with which an F-light edge is not sampled. Another proof that uses a backward analysis is given in [3].

Further Comments

- Recently (and since the appearance of the KKT algorithm in 1995), two new deterministic algorithms for MST have appeared, due to Chazelle [4] and Pettie and Ramachandran [14]. The former [4] runs in $O(n + m\alpha(m, n))$ time, where α is an inverse of the Ackermann's function, whose growth rate is even smaller than the β function mentioned earlier for the best result that was known prior to the KKT algorithm [7]. The latter algorithm [14] provably runs in time that is within a constant factor of the decision-tree complexity of the MST problem and hence is optimal; its time bound is $O(n + m\alpha(m, n))$ and $\Omega(n + m)$, and the exact bound remains to be determined.
- Although the KKT algorithm runs in expected linear time (and with exponentially high probability), it is not the last word on randomized MST algorithms. A randomized MST algorithm that runs in expected linear time and uses only $O(\log^* n)$ random bits is given in [16, 17]. In contrast, the KKT algorithm uses a linear number of random bits.

Applications

The minimum spanning tree problems has a large number of applications, which are discussed in minimum spanning trees.

Open Problems

Some open problems that remain are the following:

1. Can randomness be removed in the KKT algorithm? A hybrid algorithm that uses the KKT algorithm within a modified version of the Pettie-Ramachandran algorithm [14] is given in [16, 17] that achieves expected linear time while reducing the number of random bits used to only $O(\log^* n)$. Can this tiny amount of randomness be removed as well? If all randomness can be removed from the KKT algorithm, that will establish a linear time bound for the Pettie-Ramachandran algorithm [14] and also provide another optimal deterministic MST algorithm, this one based on the KKT approach.
2. Can randomness be removed from the work-optimal *parallel algorithms* [10] for MST? A linear-work, expected logarithmic-time parallel MST algorithm for the EREW PRAM is given in [15]. This parallel algorithm is both work and time optimal. However, it uses a linear number of random bits. Another work-optimal parallel algorithm is given in [16, 17] that runs in expected polylog time using only polylog random bits. This leads to the following open questions regarding parallel algorithms for the MST problem:
   - To what extent can dependence on random bits be reduced (from the current linear bound) in a time- and work-optimal parallel algorithm for MST?
   - To what extent can the dependence on random bits be reduced (from the current polylog bound) in a work-optimal parallel algorithm with reasonable parallelism (say polylog parallel time)?

Experimental Results

Katriel, Sanders, and Träff [11] performed an experimental evaluation of the KKT algorithm and showed that it has good performance on moderately dense graphs.

Cross-References

▸ Minimum Spanning Trees

Acknowledgments This work was supported in part by NSF grant CFF-0514876.

Recommended Reading

1. Borůvka O (1926) O jistém problému minimálním. Práce Moravské Přírodovědecké Společnosti 3:37–58. (In Czech)
2. Buchsbaum A, Kaplan H, Rogers A, Westbrook JR (1998) Linear-time pointer-machine algorithms for least common ancestors, MST verification and dominators. In: Proceedings of the ACM symposium on theory of computing (STOC), Dallas, pp 279–288
3. Chan TM (1998) Backward analysis of the Karger-Klein-Tarjan algorithm for minimum spanning trees. Inf Process Lett 67:303–304
4. Chazelle B (2000) A minimum spanning tree algorithm with inverse-Ackermann type complexity. J ACM 47(6):1028–1047
5. Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms. MIT, Cambridge
6. Dixon B, Rauch M, Tarjan RE (1992) Verification and sensitivity analysis of minimum spanning trees in linear time. SIAM J Comput 21(6):1184–1192
7. Gabow HN, Galil Z, Spencer TH, Tarjan RE (1986) Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica 6:109–122
8. Graham RL, Hell P (1985) On the history of the minimum spanning tree problem. Ann Hist Comput 7(1):43–57
9. Karger DR, Klein PN, Tarjan RE (1995) A randomized linear-time algorithm for finding minimum spanning trees. J ACM 42(2):321–329
10. Karp RM, Ramachandran V (1990) Parallel algorithms for shared-memory machines. In: van Leeuwen J (ed) Handbook of theoretical computer science. Elsevier Science, Amsterdam, pp 869–941
11. Katriel I, Sanders P, Träff JL (2003) A practical minimum spanning tree algorithm using the cycle property. In: Proceedings of the 11th annual European symposium on algorithms, Budapest. LNCS, vol 2832. Springer, Berlin, pp 679–690
12. King V (1997) A simpler minimum spanning tree verification algorithm. Algorithmica 18(2):263–270
13. Komlós J (1985) Linear verification for spanning trees. Combinatorica 5(1):57–65
14. Pettie S, Ramachandran V (2002) An optimal minimum spanning tree algorithm. J ACM 49(1):16–34
15. Pettie S, Ramachandran V (2002) A randomized time-work optimal parallel algorithm for finding a minimum spanning forest. SIAM J Comput 31(6):1879–1895
16. Pettie S, Ramachandran V (2002) Minimizing randomness in minimum spanning tree, parallel connectivity, and set maxima algorithms. In: Proceedings of the ACM-SIAM symposium on discrete algorithms (SODA), San Francisco, pp 713–722
17. Pettie S, Ramachandran V (2008) New randomized minimum spanning tree algorithms using exponentially fewer random bits. ACM Trans Algorithms 4(1):article 5

Randomized Parallel Approximations to Max Flow

Maria Serna
Department of Language and System Information, Technical University of Catalonia, Barcelona, Spain

Keywords

Approximate maximum flow construction

Years and Authors of Summarized Original Work

1991; Serna, Spirakis

Problem Definition

The work of Serna and Spirakis provides a parallel approximation schema for the Maximum Flow problem. An approximate algorithm provides a solution whose cost is within a factor of the optimal solution. The notation and definitions are the standard ones for networks and flows (see for example [2, 7]).

A *network* $N = (G, s, t, c)$ is a structure consisting of a directed graph $G = (V, E)$, two distinguished vertices, $s, t \in V$ (called the *source* and the *sink*), and $c : E \to \mathbb{Z}^+$, an assignment of an integer capacity to each edge in E. A *flow function* f is an assignment of a non-negative number to each edge of G (called the flow into the edge) such that first at no edge does the flow exceed the capacity, and second for every vertex except s and t, the sum of the flows on its incoming edges equals the sum of the flows on its outgoing edges. The *total flow* of a given flow function f is defined as the net sum of flow into the sink t. The Maximum Flow problem can be stated as

Name	Maximum Flow
Input	A network $N = (G, s, t, c)$
Output	Find a flow f for N for which the total flow is maximum.

Maximum Flows and Matchings

The Maximum Flow problem is closely related to the Maximum Matching problem on bipartite graphs.

Given a graph $G = (V, E)$ and a set of edges $M \subseteq E$ is a *matching* if in the subgraph (V, M) all vertices have degree at most one. A *maximum matching* for G is a matching with a maximum number of edges. For a graph $G = (V, E)$ with weight $w(e)$, the *weight* of a matching M is the sum of the weights of the edges in M. The problem can be stated as follows:

Name	Maximum Weight Matching
Input	A graph $G = (V, E)$ and a weight $w(e)$ for each edge $e \in E$
Output	Find a matching of G with the maximum possible weight.

There is a standard reduction from the Maximum Matching problem for bipartite graphs to the Maximum Flow problem [7, 8]. In the general weighted case one has just to look at each edge with capacity $c > 1$ as c edges joining the same points each with capacity one, and transform the multigraph obtained as shown before. Notice that to perform this transformation a c value is required which is polynomially bounded. The whole procedure was introduced by Karp, Upfal, and Wigderson [5] providing the following results

Theorem 1 *The Maximum Matching problem for bipartite graphs is NC equivalent to the Maximum Flow problem on networks with polynomial capacities. Therefore, the Maximum Flow with polynomial capacities problem belongs to the class RNC.*

Key Results

The first contribution is an extension of Theorem 1 to a generalization of the problem, namely the Maximum Flow on networks with polynomially bounded maximum flow. The proof is based on the construction (in NC) of a second network which has the same maximum flow but for which the maximum flow and the maximum capacity in the network are polynomially related.

Lemma 2 *Let $N = (G, s, t, c)$. Given any integer k, there is an NC algorithm that decides whether $f(N) \geq k$ or $f(N) < km$.*

Since Lemma 2 applies even to numbers that are exponential in size, they get

Lemma 3 *Let $N = (G, s, t, c)$ be a network, there is an NC algorithm that computes an integer value k such that $2^k \leq f(N) < m\ 2^{k+1}$.*

The following lemma establishes the NC-reduction from the Maximum Flow problem with polynomial maximum flow to the Maximum Flow problem with polynomial capacities.

Lemma 4 *Let $N = (G, s, t, c)$ be a network, there is an NC algorithm that constructs a second network $N_1 = (G, s, t, c_1)$ such that*

$$\log(\mathrm{Max}(N_1)) \leq \log(f(N_1)) + O(\log n)$$

and $f(N) = f(N_1)$.

Lemma 4 shows that the Maximum Flow problem restricted to networks with polynomially bounded maximum flow is NC-reducible to the Maximum Flow problem restricted to polynomially bounded capacities, the latter problem is a simplification of the former one, so the following results follow.

Theorem 5 *For each polynomial p, the problem of constructing a maximum flow in a network N such that $f(N) \leq p(n)$ is NC-equivalent to the problem of constructing a maximum matching in a bipartite graph, and thus it is in RNC.*

Recall that [5] gave us an $O(\log^2 n)$ randomized parallel time algorithm to compute a maximum matching. The combination of this with the reduction from the Maximum Flow problem to the Maximum Matching leads to the following result.

Theorem 6 *There is a randomized parallel algorithm to construct a maximum flow in a directed network, such that the number of processors is*

bounded by a polynomial in the number of vertices and the time used is $O((\log n)^{\alpha} \log f(N))$ *for some constant* $\alpha > 0$.

The previous theorem is the first step towards finding an approximate maximum flow in a network N by an RNC algorithm. The algorithm, given N and an $\varepsilon > 0$, outputs a solution f' such that $f(N)/f' \leq 1 + 1/\varepsilon$. The algorithm uses a polynomial number of processors (independent of ε) and parallel time $O(\log^{\alpha} n(\log n + \log \varepsilon))$, where α is independent of ε. Thus, the algorithm is an RNC one as long as ε is at most polynomial in n. (Actually ε can be $O(n^{\log^{\beta} n})$ for some β.) Thus, being a Fully RNC approximation scheme (FRNCAS).

The second ingredient is a rough NC approximation to the Maximum Flow problem.

Lemma 7 *Let* $N = (G, s, t, c)$ *be a network. Let* $k \geq 1$ *be an integer, then there is an NC algorithm to construct a network* $M = (G, s, t, c_1)$ *such that* $k\ f(M) \leq f(N) \leq k\ f(M) + km$.

Putting all together and allowing randomization the algorithm can be sketched as follows:

FAST-FLOW($N = (G, s, t, c), \varepsilon$)

1. Compute k such that $2^k \leq F(N) \leq 2^{k+1}m$.
2. Construct a network N_1 such that

$$\log(\text{Max}(N_1)) \leq \log(F(N_1)) + O(\log n).$$

3. If $2^k \leq (1 + \varepsilon)m$ then $F(N) \leq (1 + \varepsilon)m^2$ so use the algorithm given in Theorem 6 to solve the Maximum Flow problem in N as a Maximum Matching and **return**
4. Let $\beta = \lfloor (2^k)/((1 + \varepsilon)m) \rfloor$. Construct N_2 from N_1 and β using the construction in Lemma 7.
5. Solve the Maximum Flow problem in N_2 as a Maximum Matching.
6. Output $F' = \beta F(M_2)$ and for all $e \in E$, $f'(e) = \beta f(e)$.

Theorem 8 *Let* $N = (G, s, t, c)$ *be a network. Then, algorithm FAST-FLOW is an RNC algorithm such that for all* $\varepsilon > 0$ *at most polynomial in the number of network vertices, the algorithm computes a legal flow of value* f' *such that*

$$\frac{f(N)}{f'} \leq 1 + \frac{1}{\varepsilon}.$$

Furthermore, the algorithm uses a polynomial number of processors and runs in expected parallel time $O(\log^{\alpha} n(\log n + \log \varepsilon))$, *for some constant* α, *independent of* ε.

Applications

The *rounding/scaling* technique is used in general to deal with problems that are hard due to the presence of large weights in the problem instance. The technique modifies the problem instance in order to produce a second instance that has no large weights, and thus can be solved efficiently. The way in which a new instance is obtained consists of computing first an estimate of the optimal value (when needed) in order to discard unnecessary high weights. Then the weights are modified, scaling them down by an appropriate factor that depends on the estimation and the allowed error. The rounding factor is determined in such a way that the so-obtained instance can be solved efficiently. Finally, a last step consisting of scaling up the value of the "easy" instance solution is performed in order to meet the corresponding accuracy requirements.

It is known that in the sequential case, the only way to construct FPTAS uses rounding/scaling and interval partition [6]. In general, both techniques can be paralyzed, although sometimes the details of the parallelization are non-trivial [1].

The Maximum Flow problem has a long history in Computer Science. Here are recorded some results about its parallel complexity. Goldschlager, Shaw, and Staples showed that the Maximum Flow problem is P-complete [3]. The P-completeness proof for Maximum Flow uses large capacities on the edges; in fact the values of some capacities are exponential in the number of network vertices. If the capacities are constrained

to be no greater than some polynomial in the number of network vertices the problem is in ZNC. In the case of planar networks it is known that the Maximum Flow problem is in NC, even if arbitrary capacities are allowed [4].

Open Problems

The parallel complexity of the Maximum Weight Matching problem when the weight of the edges are given in binary is still an open problem. However, as mentioned earlier, there is a randomized NC algorithm to solve the problem in $O(\log^2 n)$ parallel steps, when the weights of the edges are given in unary. The scaling technique has been used to obtain fully randomized NC approximation schemes, for the Maximum Flow and Maximum Weight Matching problems (see [10]). The result appears to be the best possible in regard of full approximation, in the sense that the existence of an FNCAS for any of the problems considered is equivalent to the existence of an NC algorithm for perfect matching which is also still an open problem.

Cross-References

- ▸ Maximum Matching
- ▸ Online Paging and Caching

Recommended Reading

1. Díaz J, Serna M, Spirakis PG, Torán J (1997) Paradigms for fast parallel approximation. In: Cambridge international series on parallel computation, vol 8. Cambridge University Press, Cambridge
2. Even S (1979) Graph algorithms. Computer Science Press, Potomac
3. Goldschlager LM, Shaw RA, Staples J (1982) The maximum flow problem is log-space complete for P. Theor Comput Sci 21:105–111
4. Johnson DB, Venkatesan SM (1987) Parallel algorithms for minimum cuts and maximum flows in planar networks. J ACM 34:950–967
5. Karp RM, Upfal E, Wigderson A (1986) Constructing a perfect matching is in random NC. Combinatorica 6:35–48
6. Korte B, Schrader R (1980) On the existence of fast approximation schemes. Nonlinear Prog 4: 415–437
7. Lawler EL (1976) Combinatorial optimization: networks and matroids. Holt, Rinehart and Winston, New York
8. Papadimitriou C (1994) Computational complexity. Addison-Wesley, Reading
9. Peters JG, Rudolph L (1987) Parallel approximation schemes for subset sum and knapsack problems. Acta Informatica 24:417–432
10. Spirakis P (1993) PRAM models and fundamental parallel algorithm techniques: part II. In: Gibbons A, Spirakis P (eds) Lectures on parallel computation. Cambridge University Press, New York, pp 41–66

Randomized Rounding

Rajmohan Rajaraman
Department of Computer Science, Northeastern University, Boston, MA, USA

Years and Authors of Summarized Original Work

1987; Raghavan, Thompson

Problem Definition

Randomized rounding is a technique for designing approximation algorithms for NP-hard optimization problems. Many combinatorial optimization problems can be represented as 0-1 integer linear programs; that is, integer linear programs in which variables take values in $\{0, 1\}$. While 0-1 integer linear programming is NP-hard, the rational relaxations (also referred to as fractional relaxations) of these linear programs are solvable in polynomial time [12, 13]. Randomized rounding is a technique to construct a provably good solution to a 0-1 integer linear program from an optimum solution to its rational relaxation by means of a randomized algorithm.

Let Π be a 0-1 integer linear program with variables $x_i \in \{0, 1\}$, $1 \leq i \leq n$. Let Π_R be the rational relaxation of Π obtained by replacing the $x_i \in \{0, 1\}$ constraints by $x_i \in [0, 1]$, $1 \leq i \leq n$. The randomized rounding approach consists of two phases:

1. Solve Π_R using an efficient linear program solver. Let the variable x_i take on value $x_i^* \in [0, 1]$, $1 \leq i \leq n$.
2. Compute a solution to Π by setting the variables x_i randomly to one or zero according to the following rule:

$$\Pr[x_i = 1] = x_i^* .$$

For several fundamental combinatorial optimization problems, the randomized rounding technique yields simple randomized approximation algorithms that yield solutions provably close to optimal. Variants of the basic approach outlined above, in which the rounding of variable x_i in the second phase is done with a probability that is some appropriate function of x_i^*, have also been studied. The analyses of algorithms based on randomized rounding often rely on Chernoff–Hoeffding bounds from probability theory [5, 11].

The work of Raghavan and Thompson [14] introduced the technique of randomized rounding for designing approximation algorithms for NP-hard optimization problems. The randomized rounding approach also implicitly proves the existence of a solution with certain desirable properties. In this sense, randomized rounding can be viewed as a variant of the probabilistic method, due to Erdös [1], which is widely used for various existence proofs in combinatorics.

Raghavan and Thompson illustrate the randomized rounding approach using three optimization problems: VLSI routing, multicommodity flow, and k-matching in hypergraphs.

Definition 1 In the **VLSI Routing** problem, we are given a two-dimensional rectilinear lattice L_n over n nodes and a collection of m *nets* $\{a_i : 1 \leq i \leq m\}$, where net a_i, is a set of nodes to be connected by means of a Steiner tree in L_n. For each net a_i, we are also given a set $\mathcal{A}_i$ of *allowed* trees that can be used for connecting the nodes in that set. A solution to the problem is a set $\mathcal{T}$ of trees $\{T_i \in \mathcal{A}_i : 1 \leq i \leq m\}$. The *width* of solution $\mathcal{T}$ is the maximum, over all edges e, of the number of trees in $\mathcal{T}$ that contain the edge. The goal of the VLSI routing problem is to determine a solution with minimum width.

Definition 2 In the **Multicommodity Flow Congestiom Minimization** problem (or simply, the Congestion Minimization problem), we are given a graph $G = (V, E)$, and a set of source-destination pairs $\{(s_i, t_i) : 1 \leq i \leq k\}$. For each pair (s_i, t_i), we would like to route one unit of demand from s_i to t_i. A solution to the problem is a set $\mathcal{P} = \{P_i : 1 \leq i \leq k\}$ such that P_i is a path from s_i to t_i in G. We define the *congestion* of $\mathcal{P}$ to be the maximum, over all edges e, of the number of paths containing e. The goal of the undirected multicommodity flow problem is to determine a path set $\mathcal{P}$ with minimum congestion.

In their original work [14], Raghavan and Thompson studied the above problem for the case of undirected graphs and referred to it as the Undirected Multicommodity Flow problem. Here, we adopt the more commonly-used term of Congestion Minimization and consider both undirected and directed graphs since the results of [14] apply to both classes of graphs. Researchers have studied a number of variants of the multicommodity flow problem, which differ in various aspects of the problem such as the nature of demands (e.g., uniform vs. non-uniform), the objective function (e.g., the total flow vs. the maximum fraction of each demand), and edge capacities (e.g., uniform vs. non-uniform).

Definition 3 In the **Hypergraph Simple k-Matching** problem, we are given a hypergraph H over an n-element vertex set V. A k-matching of H is a set M of edges such that each vertex in V belongs to at most k of the edges in M. A k-matching M is simple if no edge in H occurs more than once in M. The goal of the problem is to determine a maximum-size simple k-matching of a given hypergraph H.

Key Results

Raghavan and Thompson present approximation algorithms for the above three problems using randomized rounding. In each case, the algorithm is easy to present: write a 0-1 integer linear program for the problem, solve the rational relaxation of this program, and then apply randomized rounding. They establish bounds on the quality of the solutions (i.e., the approximation ratios of the algorithm) using Chernoff–Hoeffding bounds on the tail of the sums of bounded and independent random variables [5, 11].

The VLSI Routing problem can be easily expressed as a 0-1 integer linear program, say Π_1. Let W^* denote the width of the optimum solution to the rational relaxation of Π_1.

Theorem 1 *For any ε such that $0 < \varepsilon < 1$, the width of the solution produced by randomized rounding does not exceed*

$$W^* + \left[3W^* \ln \frac{2n(n-1)}{\varepsilon}\right]^{1/2}$$

with probability at least $1 - \varepsilon$, provided $W^* \geq 3\ln(2n(n-1)/\varepsilon)$.

Since W^* is a lower bound on the width of an optimum solution to Π_1, it follows that the randomized rounding algorithm has an approximation ratio of $1 + o(1)$ with high probability as long as W^* is sufficiently large.

The Congestion Minimization problem can be easily expressed as a 0-1 integer linear program, say Π_2. Let C^* denote the congestion of the optimum solution to the linear relaxation of Π_2. This optimum solution yields a set of flows, one for each commodity i. The flow for commodity i can be decomposed into a set Γ_i of at most $|E|$ paths from s_i to t_i. The randomized rounding algorithm selects, for each commodity i, one path P_i at random from Γ_i according to the flow values determined by the flow decomposition.

Theorem 2 *For any ε such that $0 < \varepsilon < 1$, the capacity of the solution produced by randomized rounding does not exceed*

$$C^* + \left[3C^* \ln \frac{|E|}{\varepsilon}\right]^{1/2}$$

with probability at least $1 - \varepsilon$, provided $C^* \geq 2\ln|E|$.

Since C^* is a lower bound on the width of an optimum solution to Π_1, it follows that the randomized rounding algorithm achieves a constant approximation ratio with probability $1 - 1/n$ when C^* is $\Omega(\log n)$.

For both the VLSI Routing and the Congestion Minimization problems, slightly worse approximation ratios are achieved if the lower bound condition on W^* and C^*, respectively, is removed. In particular, the approximation ratio achieved is $O(\log n/\log\log n)$ with probability at least $1 - n^{-c}$ for a constant $c > 0$ whose value depends on the constant hidden in the big-Oh notation.

The hypergraph k-matching problem is different than the above two problems in that it is a packing problem with a maximization objective while the latter are covering problems with a minimization objective. Raghavan and Thompson show that randomization rounding, in conjunction with a scaling technique, yields good approximation algorithms for the hypergraph k-matching problem. They first express the matching problem as a 0-1 integer linear program, solve its rational relaxation Π_3, and then round the optimum rational solution by using appropriately scaled values of the variables as probabilities. Let S^* denote the value of the optimum solution to Π_3.

Theorem 3 *Let δ_1 and δ_2 be positive constants such that $\delta_2 > n \cdot e^{-k/6}$ and $\delta_1 + \delta_2 < 1$. Let $\alpha = 3\ln(n/\delta_2)/k$ and*

$$S' = S^*\left(1 - \frac{(\alpha^2 + 4\alpha)^{1/2} - \alpha}{2}\right).$$

Then, there exists a simple k-matching for the given hypergraph with size at least

$$S' - \left(2S' \ln \frac{1}{\delta_1}\right)^{1/2}.$$

Note that the above result is stated as an existence result. It can be modified to yield a randomized algorithm that achieves essentially the same bound with probability $1 - \varepsilon$ for a given failure probability ε.

Applications

Randomized rounding has found applications for a wide range of combinatorial optimization problems. Following the work of Raghavan and Thompson [14], Goemans and Williamson showed that randomized rounding yields an $e/(e-1)$-approximation algorithm for MAXSAT, the problem of finding an assignment that satisfies the maximum number of clauses of a given Boolean formula [7]. For the set cover problem, randomized rounding yields an algorithm with an asymptotically optimal approximation ratio of $O(\log n)$, where n is the number of elements in the given set cover instance [10]. Srinivasan has developed more sophisticated randomized rounding approaches for set cover and more general covering and packing problems [15]. Randomized rounding also yields good approximation algorithms for several flow and cut problems, including variants of undirected multicommodity flow [9] and the multiway cut problem [4].

While randomized rounding provides a unifying approach to obtain approximation algorithms for hard optimization problems, better approximation algorithms have been designed for specific problems. In some cases, randomized rounding has been combined with other algorithms to yield better approximation ratios than previously known. For instance, Goemans and Williamson showed that the better of two solutions, one obtained by randomized rounding and the other obtained by an earlier algorithm due to Johnson, yields a 4/3 approximation for MAXSAT [7].

The work of Raghavan and Thompson applied randomized rounding to a solution obtained for the relaxation of a 0-1 integer program for a given problem. In recent years, more sophisticated approximation algorithms have been obtained by applying randomized rounding to semidefinite program relaxations of the given problem. Examples include the 0.87856-approximation algorithm for MAXCUT due to Goemans and Williamson [8] and an $O(\sqrt{\log n})$-approximation algorithm for the sparsest cut problem, due to Arora, Rao, and Vazirani [3].

An excellent reference for the above and other applications of randomized rounding in approximation algorithms is the text by Vazirani [16].

Open Problems

While randomized rounding has yielded improved approximation algorithms for a number of NP-hard optimization problems, the best approximation achievable by a polynomial-time algorithm is still open for most of the problems discussed in this article, including MAXSAT, MAXCUT, the sparsest cut, the multiway cut, and several variants of the congestion minimization problem. For directed graphs, it has been shown that best approximation ratio achievable for congestion minimization in polynomial time is $\Omega(\log n / \log\log n)$, unless NP $\subset$ ZPTIME($n^{O(\log\log n)}$), matching the upper bound mentioned in section "Key Results" up to constant factors [6]. For undirected graphs, the best known inapproximability lower bound is $\Omega(\log\log n / \log\log\log n)$ [2].

Cross-References

▸ Oblivious Routing

Recommended Reading

1. Alon N, Spencer JH (1991) The probabilistic method. Wiley, New York
2. Andrews M, Zhang L (2005) Hardness of the undirected congestion minimization problem. In: STOC'05: proceedings of the thirty-seventh annual ACM symposium on theory of computing. ACM, New York, pp 284–293
3. Arora S, Rao S, Vazirani UV (2004) Expander flows, geometric embeddings and graph partitioning. In: STOC, pp 222–231

4. Calinescu G, Karloff HJ, Rabani Y (2000) An improved approximation algorithm for multiway cut. J Comput Syst Sci 60(3):564–574
5. Chernoff H (1952) A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. Ann Math Stat 23:493–509
6. Chuzhoy J, Guruswami V, Khanna S, Talwar K (2007) Hardness of routing with congestion in directed graphs. In: STOC'07: proceedings of the thirty-ninth annual ACM symposium on theory of computing. ACM, New York, pp 165–178
7. Goemans MX, Williamson DP (1994) New 3/4-approximation algorithms for the maximum satisfiability problem. SIAM J Discret Math 7:656–666
8. Goemans MX, Williamson DP (1995) Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J ACM 42(6):1115–1145
9. Guruswami V, Khanna S, Rajaraman R, Shepherd B, Yannakakis M (2003) Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. J Comput Syst Sci 67:473–496
10. Hochbaum DS (1982) Approximation algorithms for the set covering and vertex cover problems. SIAM J Comput 11(3):555–556
11. Hoeffding W (1956) On the distribution of the number of successes in independent trials. Ann Math Stat 27:713–721
12. Karmarkar N (1984) A new polynomial-time algorithm for linear programming. Combinatorica 4:373–395
13. Khachiyan LG (1979) A polynomial algorithm for linear programming. Sov Math Dokl 20:191–194
14. Raghavan P, Thompson C (1987) Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica 7
15. Srinivasan A (1995) Improved approximations of packing and covering problems. In: Proceedings of the 27th annual ACM symposium on theory of computing, pp 268–276
16. Vazirani V (2003) Approximation algorithms. Springer

Randomized Searching on Rays or the Line

Stephen R. Tate
Department of Computer Science, University of North Carolina, Greensboro, NC, USA

Keywords

Cow-path problem; Online navigation

Years and Authors of Summarized Original Work

1993; Kao, Reif, Tate

Problem Definition

This problem deals with finding a point at an unknown position on one of a set of w rays which extend from a common point (the origin). In this problem there is a *searcher*, who starts at the origin, and follows a sequence of commands such as "explore to distance d on ray i." The searcher detects immediately when the target point is crossed, but there is no other information provided from the search environment. The goal of the searcher is to minimize the distance traveled.

There are several different ways this problem has been formulated in the literature, including one called the "cow-path problem" that involves a cow searching for a pasture down a set of paths. When $w = 2$, this problem is to search for a point on the line, which has also been described as a robot searching for a door in an infinite wall or a shipwreck survivor searching for a stream after washing ashore on a beach.

Notation

The problem is as described above, with w rays. The position of the target point (or goal) is denoted (g, i) if it is at distance g on ray $i \in \{0, 1, \ldots, w-1\}$. The standard notion of *competitive ratio* is used when analyzing algorithms for this problem: An algorithm that knows which ray the goal is on will simply travel distance g down that ray before stopping, so search algorithms are compared to this optimal, omniscient strategy.

In particular, if $\mathcal{R}$ is a randomized algorithm, then the distance traveled to find a particular goal position is a random variable denoted *distance* $(\mathcal{R}, (g, i))$, with expected value $E[\text{distance}(\mathcal{R}, (g, i))]$. Algorithm $\mathcal{R}$ has competitive ratio c if there is a constant a such that, for all goal positions (g, i),

$$E[\text{distance}(\mathcal{R}, (g, i))] \leq c.g + a. \quad (1)$$

Key Results

This problem is solved optimally using a randomized geometric sweep strategy: Search through the rays in a random (but fixed) order, with each search distance a constant factor longer than the preceding one. The initial search distance is picked from a carefully selected probability distribution, giving the following algorithm:

RAYSEARCH $_{r,w}$
$\sigma \leftarrow$ A random permutation of $\{0,1,2,\ldots,w-1\}$;
$\epsilon \leftarrow$A random real uniformly chosen from $[0,1)$;
$d \leftarrow r^{\epsilon}$;
$p \leftarrow 0$;
repeat
Explore path $\sigma(p)$ up to distance d;
if goal not found then return to origin;
$d \leftarrow d \cdot r$;
$p \leftarrow (p+1) \bmod w$;
until goal found;

The following theorems give the competitive ratio of this algorithm, show how to pick the best r, and establish the optimality of the algorithm.

Theorem 1 ([9]) *For any fixed $r > 1$, Algorithm* RAYSEARCH $_{r,w}$ *has competitive ratio*

$$R(r,w) = 1 + \frac{2}{w} \cdot \frac{1 + r + r^2 + \cdots + r^{w-1}}{\ln r},$$

Theorem 2 ([9]) *The unique solution of the equation*

$$\ln r = \frac{1 + r + r^2 + \cdots + r^{w-1}}{r + 2r^2 + 3r^3 + \cdots + (w-1)r^{w-1}} \tag{2}$$

for $r > 1$, denoted by r_w^*, gives the minimum value for $R(r,w)$.

Theorem 3 ([8, 9, 12]) *The optimal competitive ratio for any randomized algorithm for searching on w rays is*

$$\min_{r>1} \left\{ 1 + \frac{2}{w} \cdot \frac{1 + r + r^2 + \cdots + r^{w-1}}{\ln r} \right\}.$$

Corollary 1 *Algorithm* RAYSEARCH $_{r,w}$ *is optimally competitive.*

Using Theorem 2 and standard numerical techniques, $r_w{}^*$ can be computed to any required degree of precision. The following table shows, for small values of w, approximate values for $r_w{}^*$ and the corresponding optimal competitive ratio (achieved by RAYSEARCH$_{r,w}$) – the optimal deterministic competitive ratio (see [1]) is also shown for comparison (Table 1):

Theorem 4 ([9]) *The competitive ratio for algorithm* RAYSEARCH $_{r,w}$ *(with $r = r_w{}^*$) is $\kappa w + o(w)$, where*

$$k = \min_{s>0} \left[2\frac{e^s - 1}{s^2} \right] \approx 3.088.$$

Applications

The most direct applications of this problem are in geometric searching, such as robot navigation problems. For example, when a robot is traveling in an unknown area and encounters an obstacle, a typical first step is to find the nearest corner to go around [2, 3], which is just an instance of the ray searching problem (with $w = 2$).

In addition, any abstract search problem with a cost function that is linear in the distance to the goal reduces to ray searching. This includes applications in artificial intelligence that search for a goal in a largely unknown search space

Randomized Searching on Rays or the Line, Table 1 The asymptotic growth of the competitive ratio with w is established in the following theorem

w	r_w^*	Optimal randomized ratio	Optimal deterministic ratio
2	3.59112	4.59112	9
3	2.01092	7.73232	14.5
4	1.62193	10.84181	19.96296
5	1.44827	13.94159	25.41406
6	1.35020	17.03709	30.85984
7	1.28726	20.13033	36.30277

[11] and the construction of hybrid algorithms [8]. In hybrid algorithms, a set of algorithms $A_1, A_2, \ldots, A_w$ for solving a problem is considered – algorithm A_1 is run for a certain amount of time, and if the algorithm is not successful algorithm A_1 is stopped and algorithm A_2 is started, repeating through all algorithms as many times as is necessary to find a solution. This notion of hybrid algorithms has been used successfully for several problems (such as the first competitive algorithm for the online k-server problem [4]), and the ray search algorithm gives the optimal strategy for selecting the trial running times of each algorithm.

Open Problems

Several natural extensions of this problem have been studied in both deterministic and randomized settings, including ray searching when an upper bound on the distance to the goal is known (i.e., the rays are not infinite but are line segments) [5, 10, 12], or when a probability distribution of goal positions is known [7]. Other variations of this basic searching problem have been studied for deterministic algorithms only, such as when the searcher's control is imperfect (so distances cannot be specified precisely) [6] and for more general search spaces like points in the plane [1]. A thorough study of these variants with randomized algorithms remains an open problem.

Cross-References

- ▸ Alternative Performance Measures in Online Algorithms
- ▸ Deterministic Searching on the Line
- ▸ Robotics

Recommended Reading

1. Baeza-Yates RA, Culberson JC, Rawlins GJE (1993) Searching in the plane. Inf Comput 16:234–252
2. Berman P, Blum A, Fiat A, Karloff H, Rosén A, Saks M (1996) Randomized robot navigation algorithms. In: Proceedings of the seventh annual ACM-SIAM symposium on discrete algorithms (SODA), pp 75–84
3. Blum A, Raghavan P, Schieber B (1991) Navigating in unfamiliar geometric terrain. In: Proceedings 23rd ACM symposium on theory of computing (STOC), pp 494–504
4. Fiat A, Rabani Y, Ravid Y (1990) Competitive k-server algorithms. In: Proceedings 31st IEEE symposium on foundations of computer science (FOCS), pp 454–463
5. Hipke C, Icking C, Klein R, Langetepe E (1999) How to find a point on a line within a fixed distance. Discret Appl Math 93:67–73
6. Kamphans T, Langetepe E (2005) Optimal competitive online ray search with an error-prone robot. In: 4th international workshop on experimental and efficient algorithms, pp 593–596
7. Kao M-Y, Littman ML (1997) Algorithms for informed cows. In: AAAI-97 workshop on on-line search, pp 55–61
8. Kao M-Y, Ma Y, Sipser M, Yin Y (1994) Optimal constructions of hybrid algorithms. In: Proceedings 5th ACM-SIAM symposium on discrete algorithms (SODA), pp 372–381
9. Kao M-Y, Reif JH, Tate SR (1996) Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. Inf Comput 133:63–80
10. López-Ortiz A, Schuierer S (2001) The ultimate strategy to search on m rays? Theor Comput Sci 261:267–295
11. Pearl J (1984) Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley, Reading
12. Schuierer S (2003) A lower bound for randomized searching on m rays. In: Computer science in perspective, pp 264–277

R

Randomized Self-Assembly

David Doty
Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA, USA

Keywords

Linear assembly; Randomized; Tile complexity

Supported by NSF grants CCF-1219274, CCF-1162589, and 1317694.

Years and Authors of Summarized Original Work

2006; Becker, Rapaport, Rémila
2008; Kao, Schweller
2009; Chandran, Gopalkrishnan, Reif
2010; Doty

Problem Definition

We use the abstract tile assembly model of Winfree [6], which models the aggregation of monomers called *tiles* that attach one at a time to a growing structure, starting from a single *seed* tile, in which bonds ("glues") on the tile are specific (glues only stick to glues of the same type on other tiles) and cooperative (so that multiple weak glues are necessary to attach a tile). The general idea of *randomized* self-assembly is to use the inherent randomness of self-assembly to help the assembly process. If multiple types of tiles are able to bind to a single binding site, then we assume that their relative concentrations determine the probability that each succeeds. With careful design, we can use the same tile set to create different structures, by changing the concentrations to affect what is likely to assemble. Another use of randomness is in reducing the number of different tile types required to assemble a shape.

Definitions

A *shape* is a finite, connected subset of $\mathbb{Z}^2$. A *tile type* is a unit square with four sides, each side consisting of a *glue label* (finite string) and a non-negative integer *strength*. We assume a finite set T of tile types, but an infinite number of copies of each tile type, each copy referred to as a *tile*. An *assembly* is a positioning of tiles on the integer lattice $\mathbb{Z}^2$; i.e., a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$. Write $\alpha \sqsubseteq \beta$ to denote that α is a *subassembly* of β, which means that $\mathrm{dom}\ \alpha \subseteq \mathrm{dom}\ \beta$ and $\alpha(p) = \beta(p)$ for all points $p \in \mathrm{dom}\ \alpha$. In this case, say that β is a *superassembly* of α. Two adjacent tiles in an assembly *interact* if the glue labels on their abutting sides are equal and have positive strength. Each assembly induces a *binding graph*, a grid graph whose vertices are tiles, with an edge between two tiles if they interact. The assembly is τ-*stable* if every cut of its binding graph has strength at least τ, where the weight of an edge is the strength of the glue it represents (energy τ is required to separate the assembly). The τ-*frontier* $\partial^\tau \alpha \subset \mathbb{Z}^2 \setminus \mathrm{dom}\ \alpha$ of α (or *frontier* $\partial\alpha$ when τ is clear from context) is the set of empty locations adjacent to α at which a single tile could bind stably.

A *tile system* is a triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, $\sigma : \mathbb{Z}^2 \dashrightarrow T$ is a *seed assembly* consisting of a single tile (i.e., $|\mathrm{dom}\ \sigma| = 1$), and $\tau \in \mathbb{N}$ is the *temperature*. An assembly α is *producible* if either $\alpha = \sigma$ or if β is a producible assembly and α can be obtained from β by the stable binding of a single tile. In this case, write $\beta \to_1 \alpha$ (α is producible from β by the attachment of one tile), and write $\beta \to \alpha$ if $\beta \to_1^* \alpha$ (α is producible from β by the attachment of zero or more tiles). If α is producible, then there is an *assembly sequence* $\boldsymbol{\alpha} = (\alpha_i \mid 1 \leq i \leq k)$ such that $\alpha_1 = \sigma$, $\alpha_k = \alpha$, and, for each $i \in \{1, \ldots, k-1\}$, $\alpha_i \to_1 \alpha_{i+1}$. An assembly is *terminal* if no tile can be τ-stably attached to it. Write $\mathcal{A}[\mathcal{T}]$ to denote the set of all producible assemblies of $\mathcal{T}$, and write $\mathcal{A}_\square[\mathcal{T}]$ to denote the set of all producible, terminal assemblies of $\mathcal{T}$. We also speak of *shapes* assembled by tile assembly systems, by which we mean $\mathrm{dom}\ \alpha$ if $\alpha \in \mathcal{A}_\square[\mathcal{T}]$, and we consider shapes to be equivalent up to translation.

We now define the semantics of incorporating randomization into self-assembly. Intuitively, there are two sources of nondeterminism in the model as defined: (1) if $|\partial\alpha| > 1$, then there are multiple binding sites, one of which is nondeterministically selected as the next site to receive a tile, and (2) if multiple tile types could bind to a *single* binding site, then one of them is nondeterministically selected. Both concepts are handled by assigning positive real-valued concentrations to each tile type; Ref. [3] gives a full definition that accounts for both of these. However, in the results we discuss, only the latter source of nondeterminism will actually af-

fect the probabilities of various terminal assemblies being produced; the binding sites themselves can be picked in an arbitrary order without affecting these probabilities. Thus we state here a simpler definition based on this assumption.

A *tile concentration assignment* on $\mathcal{T}$ is a function $\rho : T \to [0, \infty)$. If $\rho(t)$ is not specified explicitly for some $t \in T$, then $\rho(t) = 1$. If α is a τ-stable assembly such that $t_1, \ldots, t_j \in T$ are the tiles capable of binding to the same position $\mathbf{m} \in \partial\alpha$, then for $1 \leq i \leq j$, t_i binds at position $\mathbf{m}$ with probability $\frac{\rho(t_i)}{\rho(t_1)+\cdots+\rho(t_j)}$. ρ induces a probability measure on $\mathcal{A}_\square[T]$ in a straightforward way. Formally, let $\alpha \in \mathcal{A}_\square[T]$ be a producible terminal assembly. Let $A(\alpha)$ be the set of all assembly sequences $\boldsymbol{\alpha} = (\alpha_i \mid 1 \leq i \leq k)$ such that $\alpha_k = \alpha$, with $p_{\boldsymbol{\alpha},i}$ denoting the probability of attachment of the tile added to α_{i-1} to produce α_i (noting that $p_{\boldsymbol{\alpha},i} = 1$ if the ith tile attached without contention). Then $\Pr[\alpha] = \sum_{\boldsymbol{\alpha} \in A(\alpha)} \prod_{i=2}^{k} \frac{1}{|\partial\alpha_i|} p_{\boldsymbol{\alpha},i}$. Write $\mathcal{T}(\rho)$ to denote the random variable representing the producible, terminal assembly produced by $\mathcal{T}$ when using tile concentration assignment ρ.

Problems

The general problem is this: given a shape $X \subset \mathbb{Z}^2$ (a connected, finite set), set the concentrations of tile types in some tile system $\mathcal{T}$ so that $\mathcal{T}$ is likely to create a terminal assembly with shape X or "close to it." We now state formal problems that are variations on this theme. The first four problems use "concentration programming": varying the concentrations of tile types in a single tile system $\mathcal{T}$ to get it to assemble different shapes. The last two problems concern a tile system that only does one thing – assemble a line of a desired expected length – because in this setting we will require all concentrations to be equal. However, the tile system uses randomized self-assembly to do this with far fewer tile types than are needed to accomplish the same task in a deterministic tile system.

The first three problems concern the self-assembly of squares, and the problems are listed in order of increasing difficulty. The first asks for a square with a desired expected width, the second for a guarantee that the actual width is likely to be *close* to the expected width, and finally, for a guarantee that the actual width is likely to be *exactly* the expected width.

Formally, design a tile system $\mathcal{T} = (T, \sigma, \tau)$ such that, for any $n \in \mathbb{Z}^+$, there exists a tile concentration assignment $\rho : T \to [0, \infty)$ such that…

Problem 1 …dom $\mathcal{T}(\rho)$ is a square with expected width n.

Problem 2 …with probability at least $1 - \delta$, dom $\mathcal{T}(\rho)$ is a square whose width is between $(1 - \epsilon)n$ and $(1 + \epsilon)n$.

Problem 3 …with probability at least $1 - \delta$, dom $\mathcal{T}(\rho)$ is a square of width n.

The next problem generalizes the previous problems to arbitrary shapes, while making one relaxation: allowing a scaled-up version of a shape to be assembled instead of the exact shape. Formally, for $c \in \mathbb{Z}^+$ and shape $S \subset \mathbb{Z}^2$ (finite and connected), define $S^c = \left\{ (x, y) \in \mathbb{Z}^2 \;\middle|\; (\lfloor x/c \rfloor, \lfloor y/c \rfloor) \in S \right\}$ to be S *scaled by factor* c.

Problem 4 Let $\delta > 0$. Design a tile system $\mathcal{T} = (T, \sigma, \tau)$ such that, for any shape $S \subset \mathbb{Z}^2$, there exists a tile concentration assignment $\rho : T \to [0, \infty)$ and $c \in \mathbb{Z}^+$ so that, with probability at least $1 - \delta$, dom $\mathcal{T}(\rho)$ is S^c.

It is easy to see that for a deterministic tile system to assemble a length n, height 1 line requires n tile types. The next problem concerns using randomization to reduce the number of tile types required, subject to the constraint that all tile type concentrations are equal. (Without this constraint, a solution to Problem 1 would trivially be a solution to the next problem, with optimal $O(1)$ tile types, but since the solution to Problem 1 uses different tile type concentrations to achieve its goal, it cannot be used directly for this purpose.)

Problem 5 Let $n \in \mathbb{Z}^+$. Design a tile system $\mathcal{T} = (T, \sigma, \tau)$ such that, with tile concentration

assignment $\rho : T \to [0, \infty)$ defined by $\rho(t) = 1$ for all $t \in T$, dom $\mathcal{T}(\rho)$ is a height 1 line of expected length n.

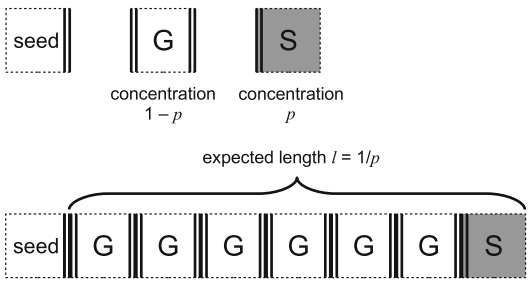
As with the case of concentration programming, it is desirable for the line to have length likely to be close to its expected length.

Problem 6 Let $n \in \mathbb{Z}^+$ and $\delta, \epsilon > 0$. Design a tile system $\mathcal{T} = (T, \sigma, \tau)$ such that, with tile concentration assignment $\rho : T \to [0, \infty)$ defined by $\rho(t) = 1$ for all $t \in T$, dom $\mathcal{T}(\rho)$ is a height 1 line whose length is between $(1 - \epsilon)n$ and $(1 + \epsilon)n$ with probability at least $1 - \delta$.

Key Results

The solutions to Problems 1–4 use temperature 2 tile systems. The solutions to Problems 5 and 6 use a temperature 1 tile system (there is no need for cooperative binding in one dimension).

Figure 1 shows a simple tile system with three tile types that can grow a line of any desired expected length to the right of the seed tile; this is the basis for the solutions to Problems 1–4. The length of the line has a geometric distribution, with expected value controlled by the ratio of the concentrations of G and S. Figure 2 shows the solution to Problem 1, due to Becker, Remilá, and Rapaport [1]. It is essentially the tile system from Fig. 1 (tile types A and B are analogous to G and S in Fig. 1) augmented with a constant number of extra tiles that can assemble the square to be as high as the line is long.



Randomized Self-Assembly, Fig. 1 A randomized temperature $\tau = 2$ tile system that can grow a line of any desired expected length l by setting $p = \frac{1}{l}$. Two tiles compete nondeterministically to bind to the right of the line (using strength 2 glues, indicated by *double black lines*), one of which stops the growth, while the other continues, giving the length of the line (not counting the seed) a geometric distribution with expected value l

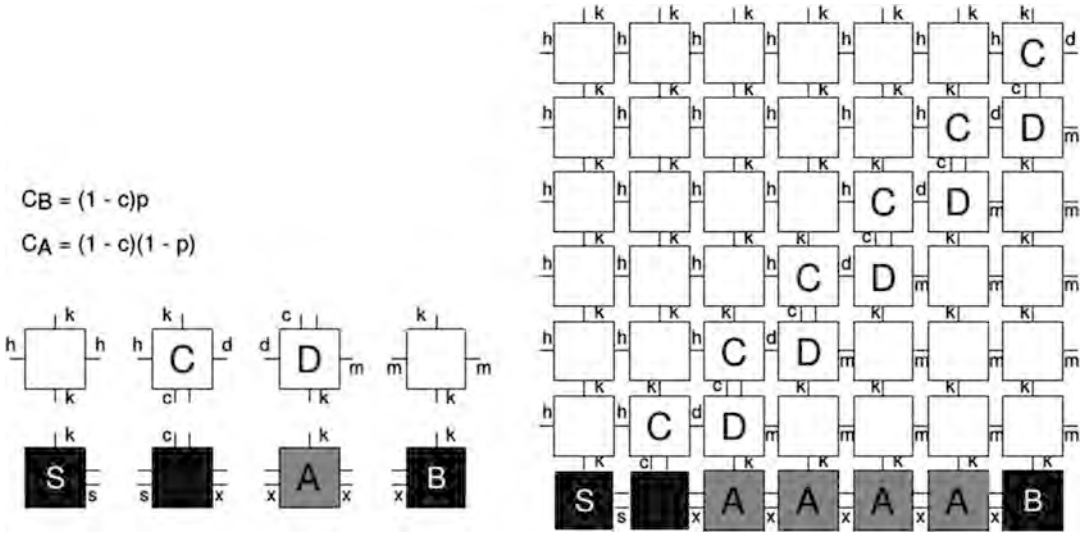
Kao and Schweller [4] showed a solution to Problem 2, and Doty [3] improved their construction to show a solution to Problem 3. Here, we describe only the latter construction, since the two share similar ideas, and the latter construction solves both problems.

Figure 3 shows an improvement to the tile system of Fig. 1, which will be the starting point for the solution. It also can grow a line of any desired expected length. However, by using multiple independent "stages" of growth, each stage having a geometric distribution, the resulting assembly is more likely to have a length that is close to its expected length. More tile types are needed for more stages, but only a constant number of stages are required.

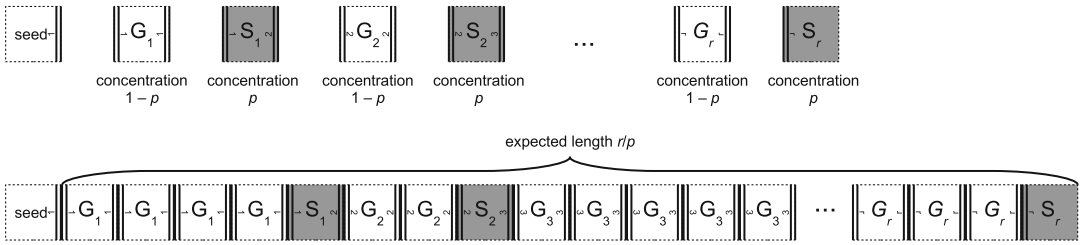
In particular, if the expected length is chosen to be midway between any two consecutive powers of two, i.e., midway in the interval $[2^{a-1}, 2^a)$ for arbitrary $a \in \mathbb{N}$, with $r = 113$ stages, the probability is at most 0.0025 that the actual length is outside the interval $[2^{a-1}, 2^a)$. So although the length is not controlled with exact precision, the number of bits needed to represent the length is controlled with exact precision (with high probability), using a constant number of tile types.

Figure 4 shows a tile system $\mathcal{T}$ with the following property: for any bit string s (equivalently, any natural number m if we assume the most significant bit of s is 1), there is a tile concentration assignment that causes $\mathcal{T}$ to grow an assembly of height $O(\log m)$, width $O(m^2)$, such that the tile types in the upper-right corner of the assembly encode s. The bottom row is the tile system from Fig. 3, with identical strength 2 glues on the north of the tiles (other than the final stop tile on the right).

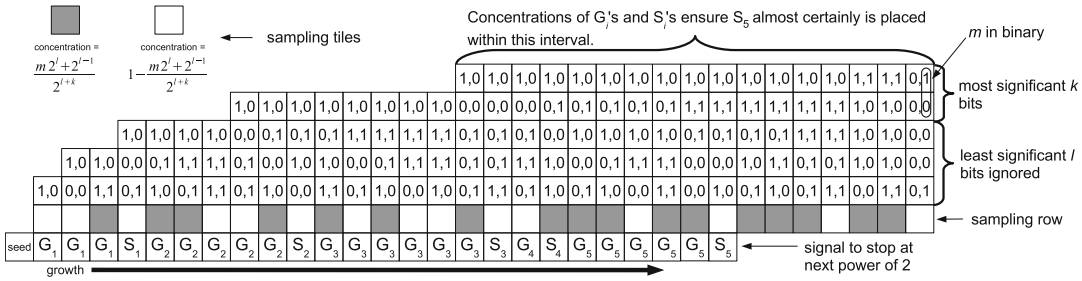
Figure 5 shows a high-level overview of the entire tile system that assembles an $n \times n$ square, solving Problem 3. Using similar ideas to Fig. 4, one can encode three different numbers $m_1, m_2, m_3 \in \mathbb{N}$ into the tile concentrations. We choose these numbers to be such that each $m_i = O(n^{1/3})$, and each of their binary



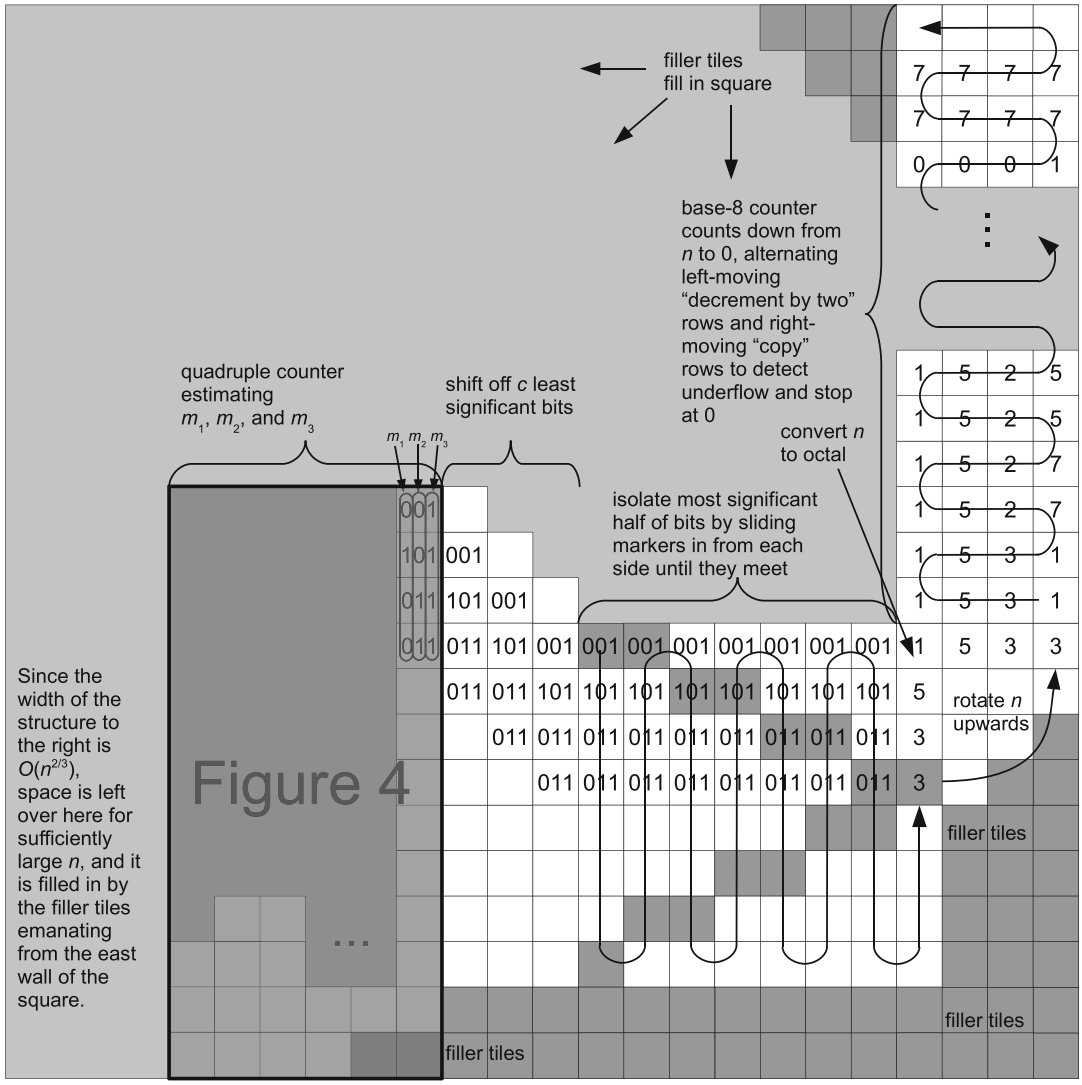
Randomized Self-Assembly, Fig. 2 A tile system that grows a square of any desired expected width (Figure taken from [4]); strength 2 glues are indicated by *two lines* between the tiles. The seed is labeled S, and C_A and C_B respectively represent the concentrations of A and B. p is used the same way as in Fig. 1, and c represents total concentration of all other tile types, since [4] assumed that concentrations of all tile types must sum to 1



Randomized Self-Assembly, Fig. 3 A tile system that grows a line of a given length with greater precision than in Fig. 1. r stages each have expected length $1/p$, making the expected total length r/p, but more tightly concentrated about that expected length than in the case of one stage

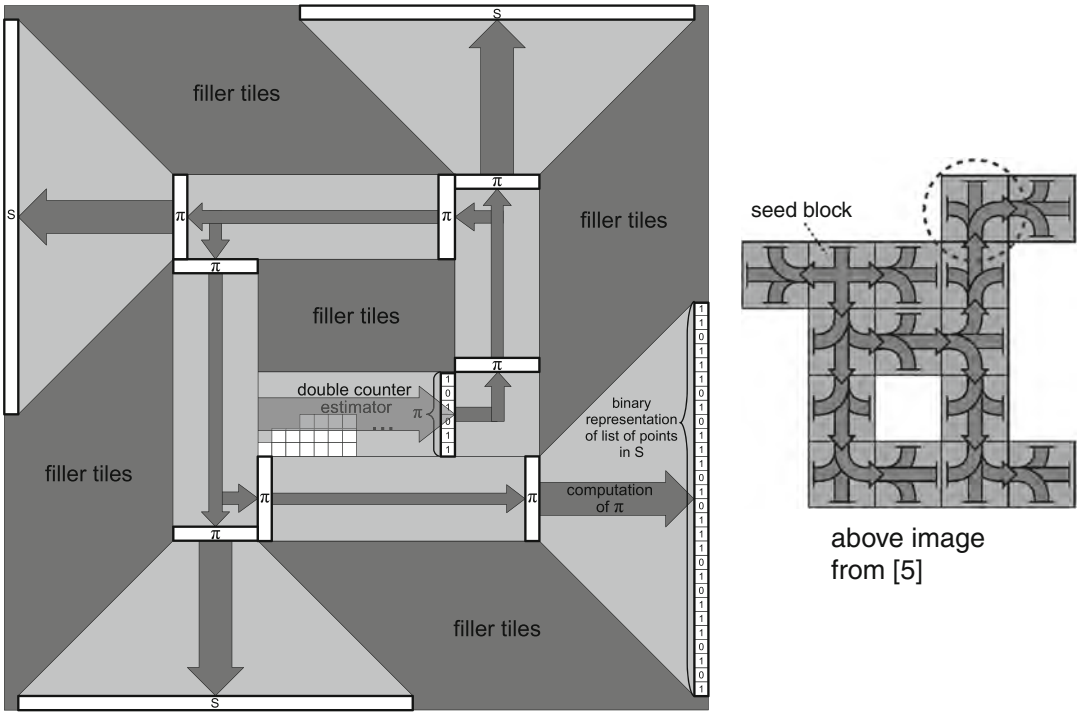


Randomized Self-Assembly, Fig. 4 Computing the binary string 10 (equivalently, the natural number $m = 2$) from tile concentrations. For brevity, glue strengths and labels are not shown. Each column increments the primary counter, represented by the bits on the left of each tile, and each *gray tile* increments the sampling counter, represented by the bits on the right of each tile. The number of bits at the end is $l + k$, where c is a constant coded into the tile set and k depends on m, and $l = k+c$. The most significant k bits of the sampling counter encode m. In this example, $k = 2$ and $c = 1$



Randomized Self-Assembly, Fig. 5 High-level overview of the entire construction solving Problem 3, not at all to scale. For brevity, glue strengths and labels are not shown. The double counter number estimator of Fig. 4 is embedded with two additional counters to create a quadruple counter estimating m_1, m_2, and m_3, shown as a box labeled as "Fig. 4" in the above figure. In this example, $m_1 = 4$, $m_2 = 3$, and $m_3 = 15$, represented vertically in binary in the most significant 4 tiles at the end of the quadruple counter. Concatenating the bits of the tiles results in the string 001101011011, the binary representation of 859, which equals $n - 2k - 4$ for $n = 871$, so this example builds an 871×871 square. Once the counter ends, c tiles ($c = 3$ in this example) are shifted off the bottom, and the top half of the tiles are isolated ($k = 4$ in this example). Each remaining tile represents 3 bits of n, which are converted into octal digits, rotated to face upwards, and then used to initialize a base-8 counter that builds the east wall of the square. Filler tiles cover the remaining area of the square

expansions, interwoven into a single bit string, is the binary expansion of n. Then each tile at the upper right of Fig. 4 encodes not one but 3 bits of n, or equivalently each encodes an octal digit of n. These bits are then used to assemble a counter that counts from n down to 0 as it grows north, and a constant set of tiles (similar to Fig. 2) expand this counter to grow about as far east as the counter grows north, creating an $n \times n$ square that surrounds the assembly of Fig. 4.



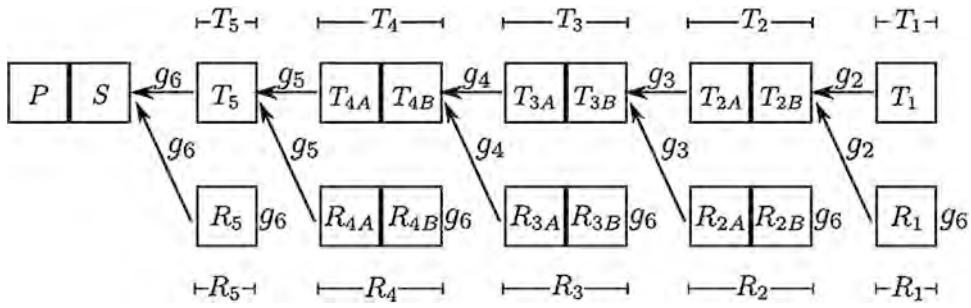
Randomized Self-Assembly, Fig. 6 On the *left* is the seed block used to replace the seed block of [5], from which the construction of [5] can assemble a scaled version of the shape S (encoded by a binary string representing the list of coordinates, also labeled "S" in the figure). S is output by the single-tape Turing machine program π. π is estimated from tile concentrations as in Fig. 4, then four copies of it are propagated to each side of the block, where it is executed in four rotated, but otherwise identical, computation regions. When completed, four copies of the binary representation of S border the seed block, which is sufficient for the construction of [5] to assemble a scaled version of S using a spanning tree of S as shown on the *right*

Since $m_i = O(n^{1/3})$, and the tiles of Fig. 4 create a structure of height $O(\log m_i)$ and width $O(m_i^2) = O(n^{2/3})$, the square is sufficiently large to contain the tiles of Fig. 4.

Finally, the tiles of Fig. 4 are used in a different way to solve Problem 4, shown in Fig. 6. Given a finite shape S, Soloveichik and Winfree [5] use an intricate construction of a "seed block" that "unpacks," from a set of tile types that depend on S, a single-tape Turing machine program $\pi \in \{0,1\}^*$ that outputs a binary string bin(S) representing a list of the coordinates of S.

The width of the seed block is then c, chosen to be large enough to do the unpacking and also large enough to accommodate the simulation of π by a tile set that simulates single-tape Turing machines. Once this seed block is in place, a tile set then assembles the scaled shape by carrying bin(S) through each block. The order in which blocks are assembled is determined by a spanning tree of S, so that any blocks with an ancestor relationship have a dependency, in that the ancestor must be (mostly) assembled before the descendant, whereas blocks without an ancestor relationship can potentially assemble in parallel.

We replace the seed block tiles of [5], which depend on S, with a single tile system that produces the program π from tile concentrations, and use the remainder of the tile set of [5] unchanged. This is illustrated in Fig. 6. Choose c to be sufficiently large that π can be simulated within the trapezoidal region of the $c \times c$ block of Fig. 6 and also sufficiently large that the construction of Fig. 4 has sufficient room to estimate the binary string π from tile concentrations in the center region (the



Randomized Self-Assembly, Fig. 7 Example of solution to Problem 5 for the case of expected length 92

"double counter estimator") of Fig. 6. Once this is done, the construction of [5] can take over and assemble the entire scaled shape S^c. The portion of the construction of [5] that achieves this is a constant-size tile set, so combined with the presented construction remains constant. This solves Problem 4.

Finally, Problems 5 and 6 have solutions due to Chandran, Gopalkrishnan, and Reif [2], which we now explain intuitively (the actual analysis is a bit trickier but is close to the following intuitive argument). Figure 7 shows an example of a solution to Problem 5 for the case of expected length $n = 92$. Each T_{iB} tile type has an east glue, g_i, that matches two tile types $T_{(i-1)A}$ and $R_{(i-1)A}$. There are $O(\log n)$ "stages" (five stages in this case). Each stage has probability $\frac{1}{2}$ to either decrement the stage or reset back to the highest stage. The number n is programmed into the system by choosing each stage to have either 1 or 2 tiles. Given that we are in stage i, to make it from stage i to stage 1 without resetting means that i consecutive unbiased coin flips must come up "heads," which we expect to take 2^i flips before happening. Thus we expect stage i to appear 2^i times; this means that stage i's expected contribution to the total length is either 2^i or $2 \cdot 2^i$, depending on whether it has 1 or 2 tiles. The reason this works to encode arbitrary natural numbers n is that every natural number can be expressed as $n = \sum_{i=0}^{\approx \log n} b_i 2^i$, where $b_i \in \{1, 2\}$. Since there are a constant number of tile types per stage, this implies that the number of tile types required is $O(\log n)$.

This solves Problem 5. To solve Problem 6, it suffices to concatenate k independent assemblies of the kind shown in Fig. 7, where k is a constant that, if chosen sufficiently large based on δ (the desired error probability), solves Problem 6 since it increases the number of tile types required. In addition to proving that this works, Chandran, Gopalkrishnan, and Reif [2] also show a more complex construction with even sharper bounds on the probability that the length differs very much from its expected value.

Open Problems

The construction resolving Problem 3 shows that for every $\delta, \epsilon > 0$, a tile set exists such that, for every $n \in \mathbb{N}$, appropriately programming the tile concentrations results in the self-assembly of a structure of size $O(n^\epsilon) \times O(\log n)$ whose rightmost tiles represent the value n with probability at least $1-\delta$. (In the tile system described, $\epsilon = 2/3$, and it could be made arbitrarily close to 0 by estimating more than 3 numbers at once.) Is this optimal?

Formally, say that a tile assembly system $\mathcal{T} = (T, \sigma, 2)$ is *δ-concentration programmable* (for $\delta > 0$) if there is a (total) computable function $r : \mathcal{A}_\square[\mathcal{T}] \to \mathbb{N}$ (the *representation function*) such that, for each $n \in \mathbb{N}$, there is a tile concentration assignment $\rho : T \to [0, \infty)$ such that $\Pr[r(\mathcal{T}(\rho)) = n] \geq 1 - \delta$. In other words, $\mathcal{T}$, programmed with concentrations ρ, almost certainly self-assembles a structure that "represents" n, according to the representation

function r, and such a ρ can be found to create a high-probability representation of *any* natural number.

Question 1 Is the following statement true? For each $\delta > 0$, there is a tile assembly system $\mathcal{T}$ and a representation function $r : \mathcal{A}_{\square}[\mathcal{T}] \to \mathbb{N}$ such that $\mathcal{T}$ is δ-concentration programmable and, for each $\epsilon > 0$ and all but finitely many $n \in \mathbb{N}$, $\Pr[|\text{dom } \mathcal{T}(\rho)| < n^{\epsilon}] \geq 1 - \delta$. If so, what is the smallest bound that can be written in place of n^{ϵ}?

Cross-References

- Experimental Implementation of Tile Assembly
- Patterned Self-Assembly Tile Set Synthesis
- Robustness in Self-Assembly
- Self-Assembly at Temperature 1
- Self-Assembly of Fractals
- Self-Assembly with General Shaped Tiles
- Staged Assembly
- Temperature Programming in Self-Assembly

Recommended Reading

1. Becker F, Rapaport I, Rémila E (2006) Self-assembling classes of shapes with a minimum number of tiles, and in optimal time. In: FSTTCS 2006: foundations of software technology and theoretical computer science, Kolkata, pp 45–56
2. Chandran H, Gopalkrishnan N, Reif JH (2012) Tile complexity of linear assemblies. SIAM J Comput 41(4):1051–1073. Preliminary version appeared in ICALP 2009
3. Doty D (2010) Randomized self-assembly for exact shapes. SIAM J Comput 39(8):3521–3552. Preliminary version appeared in FOCS 2009
4. Kao M-Y, Schweller RT (2008) Randomized self-assembly for approximate shapes. In: ICALP 2008: international colloqium on automata, languages, and programming, Reykjavik. Volume 5125 of Lecture notes in computer science. Springer, pp 370–384
5. Soloveichik D, Winfree E (2007) Complexity of self-assembled shapes. SIAM J Comput 36(6):1544–1569. Preliminary version appeared in DNA 2004
6. Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology, June 1998

Range Searching

Marc van Kreveld and Maarten Löffler
Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands

Keywords

Approximate range searching; Data structures; Intersection searching; Preprocessing; Query time; Ray shooting; Storage requirements

Years and Authors of Summarized Original Work

1975; Bentley
1978; Lueker
1993; Chazelle
2000; Arya, Mount

Problem Definition

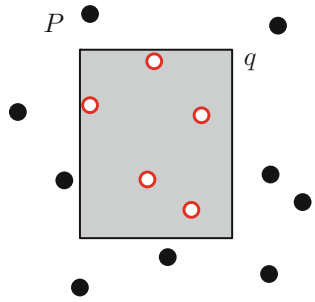
Generally speaking, data structures come in two types: those that represent data and those that allow efficient searching. The collection of results in the area of *range searching* belongs to the latter type. We distinguish two computation phases: during the *preprocessing phase*, data is stored in some suitable structure, so that during the *query phase*, all data that lies inside a query range can be found and reported efficiently.

In the most basic form of range searching, the data consists of points in a one-, two-, or higher-dimensional space, and the query range is a simple shape like a rectangle, triangle, or circle. Even for this basic form, there are many different data structures and corresponding query algorithms.

Problem 1 (Range Searching)

INPUT: Set P of n points in $\mathbb{R}^d$.
OUTPUT: Description of a data structure storing P and query algorithm that will report, for any given query d-rectangle (d-

simplex, d-sphere) q, all points of P that lie inside q.

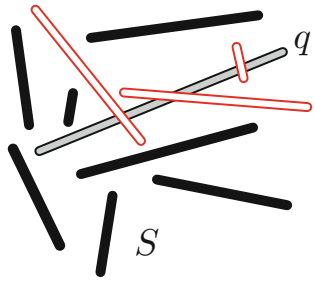


When the data is not a set of points but a set of more complex objects, such as line segments, triangles, circles, or other geometric shapes, we may not be interested in only the objects that lie completely within a query range but also all objects that *intersect* the range.

Problem 1 (Intersection Searching)

INPUT: *Set S of n non-crossing line segments in $\mathbb{R}^2$ (triangles in $\mathbb{R}^3$).*

OUTPUT: *Description of a data structure storing S and query algorithm that will report, for any given query line segment q, all line segments (triangles) of S that intersect q.*



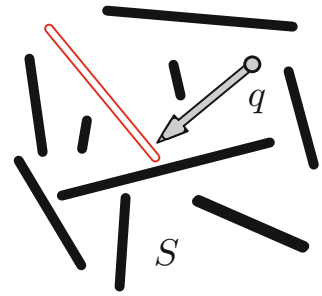
For both range searching and intersection searching, we may be interested in different types of queries. In a *counting query*, we report the number of objects in P or S that lie in the range or intersect the query object. A reporting query must spend time at least linear in the number of objects reported, whereas a counting query returns a single value. Usually, small variations of a data structure for reporting can be used for the counting version.

Ray shooting is closely related to intersection searching. We are not interested in all objects intersected by a line segment but only the first along a directed ray.

Problem 2 (Ray Shooting)

INPUT: *Set S of n non-crossing line segments in $\mathbb{R}^2$ (triangles in $\mathbb{R}^3$).*

OUTPUT: *Description of a data structure storing S and query algorithm that will report, for any given query point q and direction in $\mathbb{R}^2$ ($\mathbb{R}^3$), the first line segment (triangle) of S that is reached when q moves in the query direction.*



A combination of a data structure and a query algorithm forms a solution to a range-searching problem. The most important aspects of efficiency are the storage requirements of the data structure and the query time. Sometimes, preprocessing time and update time are also important. If the data structure is so large that it must be stored on background storage, I/O complexity becomes relevant.

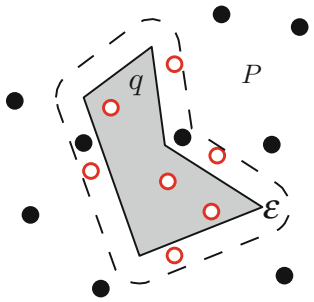
We can distinguish solutions with guaranteed efficiency and heuristics. The heuristic solutions used in practice nearly always have linear size but often have no guaranteed worst-case query time bounds. For example, R-trees [14] are among the most used data structures for range searching in practice.

One of the most interesting practical approaches for range searching with provable bounds is approximate range searching.

Problem 3 (Approximate Range Searching)

INPUT: *Set P of n points in $\mathbb{R}^d$.*

OUTPUT: *Description of a data structure storing P and query algorithm that will report, for any given query d-polyhedron q of constant complexity, all points of P that lie inside q, possibly but not necessarily some points that lie within distance ε from q, and no points that lie farther than ε from q.*



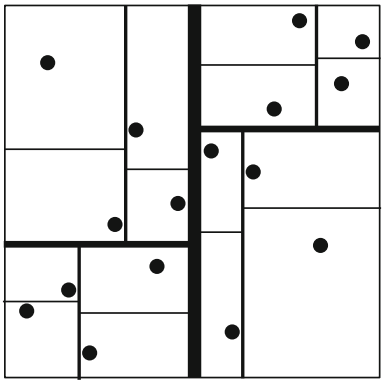
In this entry we concentrate on algorithmic results that have provable worst-case bounds for both the storage requirements and the query time.

Key Results

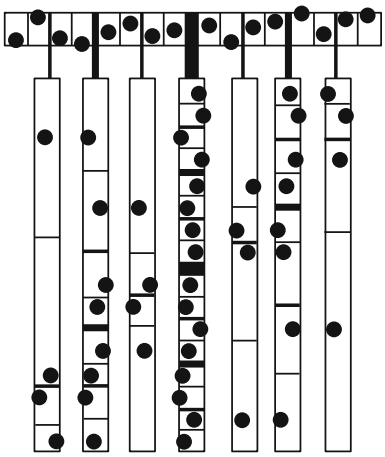
Orthogonal Range Searching

Range searching in one dimension is just searching in a sorted sequence of values. Standard binary search trees for one-dimensional searching can be extended in several ways to allow rectangular range-searching queries. For example, a *kd-tree* [4] is a balanced binary tree on a set of points in $\mathbb{R}^d$ that splits the point set on different coordinates in different nodes: the root splits on x_1-coordinate, its two children on x_2-coordinate, their four children on x_3-coordinate, and so on; after the splitting on the x_d-coordinate, the tree starts over by splitting on x_1-coordinate again. As soon as there is a single point left, it is stored in a leaf.

An *(orthogonal) range tree* [5, 10, 15] uses *associated structures*, a technique that has proved to be very powerful for solving various kinds of query problems. It refers to the fact that the structure has a main tree, and each internal node



v of the tree stores – besides two pointers to children – an extra pointer to a different data structure. Suppose that in the main tree, node v is root of a subtree storing a subset S_v of the whole set S. Then the associated structure of v also stores S_v, but in a different manner.



A range tree for a set S of points in d-space consists of a main tree that is a balanced binary search tree on x_d-coordinate. The leaves of the main tree store the points of S sorted on x_d-coordinate in the leaves. If $d > 1$, then each internal node v stores a pointer to a $(d-1)$-dimensional range tree that stores S_v restricted to their first $d-1$ coordinates.

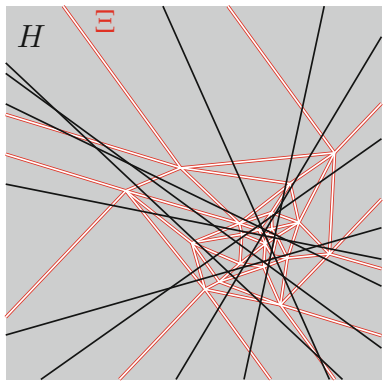
The performance of kd-trees and range trees is given in Table 1. To achieve the stated query time for range trees, an additional technique

called *fractional cascading* is needed [7]. The table also shows that in special cases, like 2-dimensional range queries in which one side of the query rectangle is unbounded (a 3-sided range), better results can be obtained using priority search trees. Other small improvements can be obtained, also depending on the machine model.

Simplex Range Searching

The range-searching problem with d-simplices is considerably harder than when the query shape is an axis-aligned d-box. There are two types of solutions: solutions with near-linear-size data structures and solutions with near-logarithmic query time. The results for d-simplex and d-half-space searching are given in Table 2.

Between the extremes of space-efficient data structures and query-efficient data structures, many other results "in between" can be obtained. For example, if for a problem in the plane one knows that a linear number of triangle range queries are needed, then one can use a data structure of size $O(n^{4/3})$ and query time close to $O(n^{1/3} + k)$, because this balances the preprocessing time (roughly the same as the size) and total query time (without the time for reporting) to something close to $O(n^{4/3})$.



A $(1/r)$-*cutting* of a set H of hyperplanes is a set Ξ of (relatively open) disjoint simplices covering $\mathbb{R}^d$ so that each simplex intersects at most n/r hyperplanes of H. Cutting trees are based on this concept. We state the main result on cuttings as a theorem, because it has implications to multidimensional divide-and-conquer schemes as well.

Theorem 1 ([6]) *Let H be a set of n hyperplanes and $r \leq n$ a parameter. Set $k = \lceil \log_2 r \rceil$. There exist k cuttings $\Xi_1, \ldots, \Xi_k$ so that Ξ_i is a $(1/2^i)$-cutting of size $O(2^{id})$, each simplex of Ξ_i is contained in a simplex of Ξ_{i-1}, and each simplex of Ξ_{i-1} contains a constant number of simplices of Ξ_i. Moreover, $\Xi_1, \ldots, \Xi_k$ can be computed in time $O(nr^{d-1})$.*

Data structures for range searching with curved boundaries can be obtained by linearization techniques. For example, range searching with a d-ball can be done by mapping each point $(x_1, \ldots, x_d)$ from the set to a point $(x_1, \ldots, x_d, x_1^2 + \cdots + x_d^2)$ in $\mathbb{R}^{d+1}$ and storing these points in a $(d+1)$-dimensional half-space range query structure. A d-ball with center $(b_1, \ldots, b_d)$ and radius r is mapped to the half-space $x_{d+1} \leq b_1(2x_1 - b_1) + \cdots + b_d(2x_d - b_d) + r^2$, and now the mapped points inside the mapped half-space correspond exactly to the original points inside the d-ball.

Intersection searching and ray shooting data structures are often based on the technique of associated structures mentioned before. Depending on the type of stored objects and the type of query objects (or query rays), different main trees and associated structures are combined into efficient solutions.

Approximate Range Searching

Many of the given data structures are not very useful in practice, especially in higher dimensions. One of the more interesting approaches toward a practical data structure for range searching that has performance guarantees is the approximate approach. The idea is that the query range is considered a shape with an inner boundary and a buffer zone around it. All points inside the inner boundary must be reported, all points outside the inner boundary but inside the buffer zone may

Range Searching, Table 1 Results on orthogonal range searching. n is the number of points stored and k is the number of points reported

Query range	Storage	Query time	Reference
d-box	$O(n)$	$O(n^{1-\frac{1}{d}} + k)$	kd-tree [4]
d-box	$O(n \log^{d-1} n)$	$O(\log^{d-1} n + k)$	Range tree [10]
3-sided rectangle	$O(n)$	$O(\log n + k)$	Priority search tree [13]

Range Searching, Table 2 Results on simplex and half-space range searching. n is the number of points stored, k is the number of points reported, c is some constant, and $\varepsilon > 0$ is an arbitrarily small constant

Query range	Storage	Query time	Reference
d-simplex	$O(n)$	$O(n^{1-\frac{1}{d}} + k)$	Partition trees [11]
d-simplex	$O(n^{d+\varepsilon})$	$O(\log n + k)$	Cutting trees [8]
d-half-space	$O(n \log \log n)$	$O(n^{1-\frac{1}{\lfloor d/2 \rfloor}} \log^c n + k)$	[12]
d-half-space	$O(n^{\lfloor d/2 \rfloor} \log^c n)$	$O(\log n + k)$	[2, 12]

but need not be reported, and all points outside the buffer zone may not be reported. A query will specify the inner boundary and a distance to the inner boundary that is the width of the buffer zone.

Assuming that the inner boundary has constant complexity and the buffer zone has width $\varepsilon \cdot D$, where D is the diameter of the inner boundary (ε is any positive constant), an approximate range query can be answered in $O(\log n + 1/\varepsilon^d)$ time, where d is the dimension of the space [3]. When the inner boundary is convex, the query time can be improved slightly.

Cross-References

- I/O-Model
- Point Location
- R-Trees

Recommended Reading

1. Agarwal PK (2004) Range searching. In: Goodman JE, O'Rourke J (eds) Handbook of discrete and computational geometry, chapter 36, 2nd edn. Chapman & Hall/CRC, Boca Raton
2. Agarwal PK, Erickson J (1998) Geometric range searching and its relatives. In: Chazelle B, Goodman J, Pollack R (eds) Advances in discrete and computational geometry. American Mathematical Society, Providence, pp 1–56
3. Arya S, Mount DM (2000) Approximate range searching. Comput Geom 17(3–4):135–152
4. Bentley JL (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18(9):509–517
5. Bentley JL (1980) Multidimensional divide-and-conquer. Commun ACM 23(4):214–229
6. Chazelle B (1993) Cutting hyperplanes for divide-and-conquer. Discret Comput Geom 9:145–158
7. Chazelle B, Guibas LJ (1986) Fractional cascading: I and II. Algorithmica 1(2):133–191
8. Chazelle B, Sharir M, Welzl E (1992) Quasi-optimal upper bounds for simplex range searching and new zone theorems. Algorithmica 8(5&6): 407–429
9. de Berg M, Cheong O, van Kreveld M, Overmars M (2008) Computational geometry – algorithms and applications, 3rd edn. Springer, Berlin
10. Lueker GS (1978) A data structure for orthogonal range queries. In: The annual symposium of the foundations of computer science (FOCS), Ann Arbor. IEEE Computer Society, pp 28–34
11. Matousek J (1992) Efficient partition trees. Discret Comput Geom 8:315–334
12. Matousek J (1992) Reporting points in halfspaces. Comput Geom 2:169–186
13. McCreight EM (1985) Priority search trees. SIAM J Comput 14(2):257–276
14. Samet H (2006) Foundations of multidimensional and metric data structures. Morgan Kaufmann, San Francisco
15. Willard DE (1979) The super-b-tree algorithm. Report TR-03-79, Aiken Computer Laboratory, Harvard University, Cambridge

Rank and Select Operations on Bit Strings

Rajeev Raman
Department of Computer Science, University of Leicester, Leicester, UK

Keywords

Bit vectors; Predecessor search; Sets; Succinct data structures

Years and Authors of Summarized Original Work

1974; Elias
1989; Jacobson
1998; Clark
2007; Raman, Raman, Rao
2008; Pătraşcu
2014; Golynski, Orlandi, Raman, Rao

Problem Definition

Given a static bit string $\boldsymbol{b} = b_1 \ldots b_m$, the objective is to preprocess $\boldsymbol{b}$ and to create a space-efficient data structure that supports the following operations rapidly:

$\mathsf{rank}_1(i)$ takes an index i as input, $1 \leq i \leq m$, and returns the number of **1**s among $b_1 \ldots b_i$.
$\mathsf{select}_1(i)$ takes an index $i \geq 1$ as input and returns the position of the i-th **1** in $\boldsymbol{b}$, and -1 if i is greater than the number of **1**s in $\boldsymbol{b}$.

A data structure that supports the operations above will be called a *bit vector*. The operations rank_0 and select_0 are defined analogously for the **0**s in $\boldsymbol{b}$. As $\mathsf{rank}_0(i) = i - \mathsf{rank}_1(i)$, one considers just rank_1 (abbreviated to rank) and refers to select_0 and select_1 collectively as select. In what follows, $|\boldsymbol{x}|$ denotes the length of a bit string $\boldsymbol{x}$ and $w(\boldsymbol{x})$ denotes the number of **1**s in it. $\boldsymbol{b}$ is always used to denote the input bit string, m to denote $|\boldsymbol{b}|$ and n to denote $w(\boldsymbol{b})$.

Memory Usage Models

In terms of space usage, we aim not only to store $\boldsymbol{b}$ in the minimum amount of space but also to minimize any additional space (called the *redundancy*) needed to support rank and select. The notion of redundancy can be formalized in two different ways.

In the *succinct index* model (also known as the *systematic* model), the bit vector does not have direct access to $\boldsymbol{b}$, but can obtain $O(\log m)$ consecutive bits of $\boldsymbol{b}$ in $O(1)$ time. During preprocessing, one can create additional data structures (called *succinct indices*) to allow rapid rank and select queries. Indices allow the representation of $\boldsymbol{b}$ to be decoupled from the auxiliary data structure, e.g., $\boldsymbol{b}$ can be stored (in a potentially highly compressed form) in a data structure such as that of [6]. The redundancy in the succinct index model is the space usage of the index.

In the *unrestricted* model, we give a "space budget" for storing $\boldsymbol{b}$, based upon some compressibility measure (the data structure is usually designed to target a particular measure). We now give some examples of space budgets:

- The obvious space budget for $\boldsymbol{b}$ is m bits, and this is used if $\boldsymbol{b}$ is believed to be incompressible.
- Recalling that $n = w(\boldsymbol{b})$, we define the space budget $B(m,n) = \lceil \log_2 \binom{m}{n} \rceil$, which is the information-theoretic minimum number of bits to store a bit string of length m with n **1**s. Using standard approximations of the factorial function, one can show [17] that $B(m,n) = n \log_2(m/n) + n \log_2 e + O(n^2/m)$. In particular, if $n = o(m)$, then $B(m,n) = o(m)$.
- Yet another space budget is obtained from the *k-th-order empirical entropy*, denoted by $H_k(\boldsymbol{b})$. For any bit string s, define $\#(s)$ as the number of (possibly overlapping) contiguous occurrences of the bit string s in $\boldsymbol{b}$. Then, for any $k \geq 0$,

$$H_k(\boldsymbol{b}) = -\frac{1}{m} \sum_{s \in \{0,1\}^k} \left(\#(s\mathbf{0}) \log_2 \frac{\#(s\mathbf{0})}{\#(s)} + \#(s\mathbf{1}) \log_2 \frac{\#(s\mathbf{1})}{\#(s)} \right) \quad (1)$$

(take $\log_2(0/0) = 0 \log_2 0 = 0$ and $\#(s) = m$ when s is the empty string). $H_k(\boldsymbol{b})$ gives the information content per bit in $\boldsymbol{b}$, when conditioned upon the previous k bits as context. The space budget is therefore $mH_k(\boldsymbol{b})$. Note that $mH_0(\boldsymbol{b}) \sim B(m,n)$, but even $H_1(\boldsymbol{b})$ can be much smaller than $H_0(\boldsymbol{b})$, and in general $H_{k+1} \leq H_k$. For example, if $\boldsymbol{b} = (\mathbf{01})^{m/2}$, then $H_0(\boldsymbol{b}) \sim m$ but $mH_1(\boldsymbol{b})$ vanishes.

The redundancy in the unrestricted model is the difference between the space usage of the data structure and the space budget.

Models of Computation

Three models of computation are commonly considered. One is the *word RAM* model with word size $O(\log m)$ bits [13]. The other models, which are particularly useful for proving lower bounds, are the *cell probe* and *bit probe* models. In the cell probe model, the time complexity of answering a query is the worst-case number of words of $O(\log m)$ consecutive bits of the data structure that are read by the algorithm to answer that query. All other computation is "free." The bit probe model is similar, except that we only count the number of bits of the data structure that are read when answering a query. Clearly, $O(\log m)$ bit probes can be more useful than reading $O(1)$ consecutive words, so $O(\log m)$ bit probes are more powerful than $O(1)$ cell probes. Also, $O(1)$ cell probes are more powerful than $O(1)$ time on the word RAM, since computation on values read into registers is for free in the cell probe model. Thus, an $O(t)$ upper bound in the word RAM is stronger than $O(t)$ upper bound in the cell probe model, which is stronger than an $O(t \log m)$ upper bound in the bit probe model. For lower bounds, the situation is of course reversed, with cell probe lower bounds being stronger than equivalent word RAM lower bounds.

Key Results

Relation to Predecessor Search

Given a static set $S \subseteq \{0, \ldots, m-1\}$, $|S| = n$, the *predecessor search* problem is to preprocess S to answer the query $\mathsf{pred}(x, S) = \max\{y \in S | y \leq x\}$. The predecessor search can easily be solved using a bit vector: we simply create a bit string $\boldsymbol{b}$ that is the characteristic vector of S, and note that (i) $|\boldsymbol{b}| = m$, (ii) $w(\boldsymbol{b}) = n = |S|$, and (iii) $\mathsf{pred}(x, S) = \mathsf{select}_1(\mathsf{rank}_1(x))$.

Clearly, if we are interested in highly space-efficient solutions, space usages of significantly more than $O(n \log m)$ bits are not of interest, since any bit string $\boldsymbol{b}$ can be represented as a set using $O(n \log m)$ bits by enumerating the positions of its **1**s. However, this close connection of the bit vector problem to the predecessor search problem means that lower bounds for the predecessor search problem also apply to the bit vector problem. In particular, if rank should take $O(1)$ time and the space should be at most $O(n \log m)$ bits, then this is only possible if $n = m/(\log m)^{O(1)}$ [19]. Since constant-time rank (and select) is taken by the succinct data structure community to be a "standard" expectation, this lower bound means that we only consider moderately sparse bit strings $\boldsymbol{b}$ in this entry.

Reductions

It has been already noted that rank_0 and rank_1 reduce to each other and that operations on sets reduce to select operations on a bit string. Some other reductions, whereby one can support operations on $\boldsymbol{b}$ by performing operations on bit strings derived from $\boldsymbol{b}$, are:

Theorem 1 *(a)* rank *reduces to* select_0 *on a bit string* $\boldsymbol{c}$ *such that* $|\boldsymbol{c}| = m + n$ *and* $w(\boldsymbol{c}) = n$.
(b) If $\boldsymbol{b}$ *has no consecutive* **1***s, then* select_0 *on* $\boldsymbol{b}$ *can be reduced to* rank *on a bit string* $\boldsymbol{c}$ *such that* $|\boldsymbol{c}| = m - n$ *and* $w(\boldsymbol{c})$ *is either* $n-1$ *or* n.
(c) From $\boldsymbol{b}$*, one can derive two-bit string* $\boldsymbol{b_0}$ *and* $\boldsymbol{b_1}$ *such that* $|\boldsymbol{b_0}| = m - n$, $|\boldsymbol{b_1}| = n$, $w(\boldsymbol{b_0}), w(\boldsymbol{b_1}) \leq \min\{m-n, n\}$, *and* select_0 *and* select_1 *on* $\boldsymbol{b}$ *can be supported by supporting* select_1 *and* rank *on* b_0 *and* b_1.

Parts (a) and (b) follow from Elias's observations on multiset representations, specialized to sets. For part (a), create $\boldsymbol{c}$ from $\boldsymbol{b}$ by adding a **0** after every **1**. For example, if $\boldsymbol{b} = \mathbf{01100100}$, then $\boldsymbol{c} = \mathbf{01}0\mathbf{1}0\mathbf{00}1\mathbf{0}0\mathbf{0}$. Then, $\mathsf{rank}_1(i)$ on $\boldsymbol{b}$ equals

$\mathsf{select}_0(i) - i$ on $\boldsymbol{c}$. For part (b), essentially invert the mapping of part (a). Part (c) is shown in [3].

Succinct Indices for Bit Vectors

The following is known about the sizes of succinct indices for bit vectors:

Theorem 2 ([11]) *Given a bit string $\boldsymbol{b}$ with $|\boldsymbol{b}| = m$, $w(\boldsymbol{b}) = n$, and $m/n = (\log m)^{O(1)}$, there is an index of size*

$$\begin{cases} O(m' \log(n/m')) & \text{if } n = \omega(m') \\ O(n(1 + \max\{0, \log(m'/n)\})) & \text{if } n = O(m') \end{cases} \text{bits},$$

where $m' = m/\log m$, that supports rank, select_0, *and* select_1 *in $O(1)$ time. This index size is optimal for any data structure that makes $O(\log m)$ bit probes to $\boldsymbol{b}$.*

This result generalizes an earlier result by Golynski, who showed that the index size must be $\Theta(m \log\log m/\log m)$ bits for $O(1)$ time operations [9]. The bound of Theorem 2 is asymptotically the same when n is relatively close to m, e.g., when $n = \Omega(m/(\log m)^{1/2})$, but is smaller thereafter, e.g., for $n = \Theta(m/(\log m)^2)$ the index size implied by Theorem 2 $O(m \log\log m/(\log m)^2)$ bits, which is a $\Theta(\log m)$ factor better than that given by [9].

Elias [5] previously gave an $o(m)$-bit index that supported select in $O(\log m)$ bit probes on average (where the average was computed across all select queries). Jacobson [14] gave $o(m)$-bit indices that supported rank and select in $O(\log m)$ bit probes in the worst case. Clark and Munro [2] gave the first $o(m)$-bit indices that support both rank and select in $O(1)$ time on the RAM.

Bit Vectors in the Unrestricted Model

In the unrestricted model, the best redundancy, if one is targeting the $B(m,n)$ space budget, is given by the following result due to Pătraşcu:

Theorem 3 ([18]) *A bit string $\boldsymbol{b}$ with $|\boldsymbol{b}| = m$ and $w(\boldsymbol{b}) = n$ can be represented using $B(m,n) + m/((\log m)/t)^t + m^{3/4}(\log m)^{O(1)}$ bits of memory, supporting* rank *and* select_1 *queries in $O(t)$ time.*

Earlier results, with a significantly higher redundancy, were given by [17, 21]. Thus, for $t = O(1)$, the redundancy is $m/(\log m)^{O(1)}$. There is an almost matching lower bound:

Theorem 4 ([20]) *Any representation of a bit string $\boldsymbol{b}$ with $|\boldsymbol{b}| = m$ and $w(\boldsymbol{b}) = n$ that answers* rank *or* select_1 *queries in $O(t)$ time on the cell probe model must use $B(m,n) + m/(\log m)^t$ bits of memory.*

The case where we aim for higher-order entropy appears to be less well studied. The best-known result is as follows:

Theorem 5 ([10]) *A bit string $\boldsymbol{b}$ with $|\boldsymbol{b}| = m$ can be represented using $mH_k(\boldsymbol{b}) + O(mk/\log m)$ bits of memory, supporting* rank *and* select *queries in $O(1)$ time, for any $k \geq 1$.*

Applications

Bit vectors are fundamental building blocks in a huge number of space-efficient data structures, in real-world and theoretical applications such as XML document representation [1, 4, 7], text retrieval [16], bioinformatics [15], and data mining [22], to name but a few. In the Cross-References, we list the various succinct data structures that build on or are related to bit vectors.

Experimental Results

Bit vectors have been extensively experimentally evaluated. Mature implementations are available in the libraries `SDSL` [8] and `Succinct` [12]. Other libraries of note are Vigna's `Sux4J` (http://

sux.di.unimi.it) and Claude's `libcds` (https://github.com/fclaude/libcds).

Cross-References

- Compressed Document Retrieval on String Collections
- Compressed Range Minimum Queries
- Compressed Representations of Graphs
- Compressed Suffix Array
- Compressed Suffix Trees
- Compressed Tree Representations
- Compressing and Indexing Structured Text
- Minimal Perfect Hash Functions
- Monotone Minimal Perfect Hash Functions
- Predecessor Search
- Rank and Select Operations on Sequences
- Succinct and Compressed Data Structures for Permutations and Integer Functions
- Wavelet Trees

Recommended Reading

1. Arroyuelo D, Claude F, Maneth S, Mäkinen V, Navarro G, Nguyen K, Sirén J, Välimäki N (2015) Fast in-memory XPath search using compressed indexes. Softw Pract Exp 45(3):399–434
2. Clark DR (1998) Compact PAT trees. PhD thesis, University of Waterloo, Waterloo
3. Delpratt O, Rahman N, Raman R (2006) Engineering the LOUDS succinct tree representation. In: Àlvarez C, Serna MJ (eds) WEA. Lecture notes in computer science, vol 4007. Springer, Berlin/Heidelberg, pp 134–145
4. Delpratt O, Raman R, Rahman N (2008) Engineering succinct DOM. In: Kemper A, Valduriez P, Mouaddib N, Teubner J, Bouzeghoub M, Markl V, Amsaleg L, Manolescu I (eds) EDBT. ACM international conference proceeding series, vol 261. ACM, New York, pp 49–60
5. Elias P (1974) Efficient storage and retrieval by content and address of static files. J ACM 21:246–260
6. Ferragina P, Venturini R (2007) A simple storage scheme for strings achieving entropy bounds. Theor Comput Sci 372(1):115–121
7. Ferragina P, Luccio F, Manzini G, Muthukrishnan S (2009) Compressing and indexing labeled trees, with applications. J ACM 57(1)
8. Gog S, Beller T, Moffat A, Petri M (2014) From theory to practice: plug and play with succinct data structures. In: Gudmundsson J, Katajainen J (eds) Experimental algorithms – 13th international symposium, SEA 2014, Copenhagen, 29 June–1 July 2014. Proceedings. Lecture notes in computer science, vol 8504. Springer, Heidelberg, pp 326–337
9. Golynski A (2007) Optimal lower bounds for rank and select indexes. Theor Comput Sci 387(3):348–359
10. Golynski A, Raman R, Rao SS (2008) On the redundancy of succinct data structures. In: Gudmundsson J (ed) SWAT. Lecture notes in computer science, vol 5124. Springer, Heidelberg, pp 148–159
11. Golynski A, Orlandi A, Raman R, Rao SS (2014) Optimal indexes for sparse bit vectors. Algorithmica 69(4):906–924
12. Grossi R, Ottaviano G (2013) Design of practical succinct data structures for large data collections. In: Bonifaci V, Demetrescu C, Marchetti-Spaccamela A (eds) Experimental algorithms, 12th international symposium, SEA 2013, Rome, 5–7 June 2013. Proceedings. Lecture notes in computer science, vol 7933. Springer, Heidelberg/New York, pp 5–17
13. Hagerup T (1998) Sorting and searching on the word RAM. In: Morvan M, Meinel C, Krob D (eds) STACS 98, 15th annual symposium on theoretical aspects of computer science, Paris, 25–27 Feb 1998, Proceedings. Lecture notes in computer science, vol 1373. Springer, Berlin/New York, pp 366–398
14. Jacobson G (1989) Succinct static data structures. PhD thesis, Carnegie Mellon University, Pittsburgh
15. Li H, Durbin R (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics 25(14):1754–1760
16. Navarro G, Mäkinen V (2007) Compressed full-text indexes. ACM Comput Surv 39(1)
17. Pagh R (2001) Low redundancy in static dictionaries with constant query time. SIAM J Comput 31(2):353–363
18. Patrascu M (2008) Succincter. In: 49th annual IEEE symposium on foundations of computer science, FOCS 2008, Philadelphia, 25–28 Oct 2008. IEEE Computer Society, Los Alamitos, pp 305–313
19. Patrascu M, Thorup M (2006) Time-space trade-offs for predecessor search. In: Kleinberg JM (ed) Proceedings of the 38th annual ACM symposium on theory of computing, Seattle, 21–23 May 2006, ACM, New York, pp 232–240
20. Patrascu M, Viola E (2010) Cell-probe lower bounds for succinct partial sums. In: Charikar M (ed) Proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms, SODA 2010, Austin, 17–19 Jan 2010. SIAM, Philadelphia, pp 117–122
21. Raman R, Raman V, Satti SR (2007) Succinct indexable dictionaries with applications to encoding *k*-ary trees, prefix sums and multisets. ACM Trans Algorithms 3(4):Article 43, 25pp
22. Tabei Y, Tsuda K (2011) Kernel-based similarity search in massive graph databases with wavelet trees. In: Proceedings of the eleventh SIAM international conference on data mining, SDM 2011, 28–30 Apr 2011, Mesa. SIAM/Omnipress, Philadelphia, pp 154–163

Rank and Select Operations on Sequences

Travis Gagie
Department of Computer Science, University of Eastern Piedmont, Alessandria, Italy
Department of Computer Science, University of Helsinki, Helsinki, Finland

Keywords

String data structures; Succinct and compressed data structures

Years and Authors of Summarized Original Work

2003; Grossi, Gupta, Vitter
2006; Golynski, Munro, Rao
2007; Ferragina, Manzini, Mäkinen, Navarro
2011; Barbay, He, Munro, Rao
2012; Belazzougui, Navarro
2013; Navarro, Nekrich
2014; Barbay, Claude, Gagie, Navarro, Nekrich

Problem Definition

The query $S.\text{rank}_a(i)$ on a sequence S is defined to return the number of occurrences of the distinct character a among the first i characters of S, and the query $S.\text{select}_a(j)$ is defined to return the position of the jth occurrence of a in S (if it exists). Since rank and select queries are fundamental to the field of succinct and compressed data structures, researchers have proposed several data structures that answer them quickly while using little space. Most of these data structures also support fast random access to S, and a few of them support fast insertions and deletions of characters in S. Some of them return $S.\text{rank}_a(i)$ more quickly when the ith character of S is itself an a; the query is then called partial rank.

Key Results

While considering how to store trees and graphs in small space while supporting fast navigation, Jacobson [16] considered the problem of supporting rank and select on binary sequences. He showed how to store an n-bit binary sequence using $o(n)$ bits in addition to the sequence itself, such that we can answer rank and select using $O(\log n)$ bit probes. Later authors have considered the problem in the word-RAM model with $\Omega(\log n)$-bit words, in which Jacobson's implementation of rank takes $O(1)$ time; they showed how to answer also select in this model in $O(1)$ time while still using $o(n)$ extra bits. Pătraşcu [20] showed how we can store an n-bit binary sequence containing m 1s in a total of $\lg \binom{n}{m} + O(n/\log^c n)$ bits, where c is any constant, and still answer rank and select in $O(1)$ time.

Grossi, Gupta, and Vitter [12] described a data structure, called a wavelet tree, that uses rank and select on several binary sequences to answer access, rank, and select on sequences over larger alphabets. If S is a sequence of length n over an alphabet of size σ and a wavelet tree for S is implemented with uncompressed data structures for rank and select on the binary sequences, then it takes $n \log \sigma + o(n \log \sigma)$ bits and answers access, rank, and select in $O(\log \sigma)$ time. With instances of Pătraşcu's data structure, the space becomes $nH_0(S) + o(n)$ bits, where $H_0(S)$ is the 0th-order empirical entropy of S. To simplify, we assume throughout that $\sigma = o(n/\log n)$.

Ferragina, Manzini, Mäkinen, and Navarro [7] described a multiary version of the wavelet tree that uses only $O\left(\frac{\log \sigma}{\log \log n} + 1\right)$ time for access, rank, and select, which is $O(1)$ when $\sigma = \lg^{O(1)} n$. Their implementation takes $nH_0(S) + o(n)$ bits when $\sigma = \lg^{O(1)} n$ and $nH_0(S) + o(n \log \sigma)$ bits otherwise. Golynski, Raman, and Rao [11] reduced the space to $nH_0(S) + o(n)$ bits in the general case.

Golynski, Munro, and Rao [10] described a data structure that takes $n \lg \sigma + o(n \lg \sigma)$ bits

and either answers select in $O(1)$ time and access and rank in $O(\log\log\sigma)$ time, or answers access in $O(1)$ time, rank in $O(\log\log(\sigma)\log\log\log\sigma)$ time, and select in $O(\log\log\sigma)$ time. If the space is increased to $(1+\epsilon)n\lg\sigma$ bits, where ϵ is any positive constant, then both access and select take $O(1)$ time and rank takes $O(\log\log\sigma)$ time. Golynski [9] showed that the product of the query times for access and select and the per-character redundancy in bits must be $\Omega\left(\frac{\log^2\sigma}{w}\right)$ in general, where w is the length of a machine word.

Barbay, He, Munro, and Rao [1] described a data structure that takes $nH_k(S) + o(n\log\sigma)$ bits, where $H_k(S)$ is the kth-order empirical entropy of S, and answers access in $O(1)$ time, rank in $O(\log\log\sigma(\log\log\log\sigma)^2)$ time, and select in $O(\log\log(\sigma)\log\log\log\sigma)$ time. We assume throughout that $k = o(\log_\sigma n)$. They also reduced to $nH_0(S)+o(n\log\sigma)$ bits, the space for the version of Golynski, Munro, and Rao's data structure with $O(1)$-time select and $O(\log\log\sigma)$-time access and rank. Grossi, Orlandi, and Raman [13] reduced the space of the version with $O(1)$-time access and $O(\log\log\sigma)$-time select to $nH_k(S) + o(n\log\sigma)$ bits and reduced the time for rank to $O(\log\log\sigma)$.

Barbay, Claude, Gagie, Navarro, and Nekrich [2] combined multiary wavelet trees with the versions of Golynski, Munro and, Rao's data structure, to obtain a data structure that takes $nH_0(S) + o(n)(H_0(S) + 1)$ bits and answers one of access and select in $O(1)$ time and the other in $O(\log\log\sigma)$ time, and rank also in $O(\log\log\sigma)$ time. If the space is increased to $(1+\epsilon)nH_0(S) + o(n)$ bits, then both access and select take $O(1)$ time. They partition the alphabet into sub-alphabets such that all the characters in each sub-alphabet have roughly the same frequency, and then store a data structure that answers access, rank, and select queries on the subsequence of characters in S from that sub-alphabet.

Belazzougui and Navarro [3, 4] showed that any data structure that takes $n \cdot w^{O(1)}$ space must use $\Omega\left(\log\frac{\log\sigma}{\log w}\right)$ time for rank. They also gave the following upper bounds:

- We can store S in $nH_0(S) + o(n)$ bits and answer access, rank, and select in $O\left(\frac{\log\sigma}{\log w} + 1\right)$ time, which is $O(1)$ when $\sigma = \lg^{O(1)} n$.
- We can store S in $nH_0(S) + o(n)(H_0(S) + 1)$ bits and answer access in $O(1)$ time and select in $O(f(n,\sigma))$ time or vice versa, where $f(n,\sigma)$ is any function in $\omega(1)$, and answer rank in $O\left(\log\frac{\log\sigma}{\log w}\right)$ time.
- We can store S in $nH_k(S) + o(n\log\sigma)$ bits and answer access in $O(1)$ time, select in $O(f(n,\sigma))$ time, and rank in $O\left(\log\frac{\log\sigma}{\log w}\right)$ time in general and in $O(f(n,\sigma))$ time when $\sigma = w^{O(1)}$.

These and the other bounds described above are summarized in Table 1. In another paper [5], Belazzougui and Navarro showed how we can add $o(n)(H_0(S) + 1)$ bits to any of these representations and answer partial rank queries in the same time as access.

Dynamic Sequences

Several authors have described data structures that store binary sequences in succinct or compressed space and support fast rank, select, and update operations, typically insertions and deletions of bits. In particular, Navarro and Sadakane [19] described data structures that store a binary sequence B in $|B|H_0(B) + o(|B|)$ bits and support rank, select, insert, and delete in $O\left(\frac{\log|B|}{\log\log|B|}\right)$ time, which is optimal [8]. These can be used in wavelet trees to obtain data structures that support rank and select on dynamic sequences over larger alphabets. Navarro and Sadakane [19] and He and Munro [15] described data structures that store a sequence S in $nH_0(S) + o(n\log\sigma)$ bits, where n is the current length of S, and support access, rank, and select queries and insertions and deletions of characters in $O\left(\frac{\log n}{\log\log n}\left(\frac{\log\sigma}{\log\log n} + 1\right)\right)$ time, which is $O\left(\frac{\log n}{\log\log n}\right)$ when $\sigma = \lg^{O(1)} n$.

Navarro and Nekrich [17, 18] recently described a data structure that stores S in $nH_0(S) + o(n\log\sigma)$ bits and supports access,

Rank and Select Operations on Sequences, Table 1 A summary of previous and current upper bounds for rank and select on a sequence S of length n over an alphabet of size $\sigma = o(n/\log n)$, with ϵ a positive constant, $k = o(\log_\sigma n)$, and $f(n,\sigma) = \omega(1)$. The bounds in the second row hold when $\sigma = \lg^{O(1)} n$ and those in the last row hold when $\sigma = w^{O(1)}$, with w the word length

Source	Space (bits)	Access	Rank	Select
[12]	$nH_0(S) + o(n)$	$O(\log\sigma)$	$O(\log\sigma)$	$O(\log\sigma)$
[7]	$nH_0(S) + o(n)$	1	1	1
[7]	$nH_0(S) + o(n\log\sigma)$	$O\left(\frac{\log\sigma}{\log\log n}\right)$	$O\left(\frac{\log\sigma}{\log\log n}\right)$	$O\left(\frac{\log\sigma}{\log\log n}\right)$
[11]	$nH_0(S) + o(n)$	$O\left(\frac{\log\sigma}{\log\log n}\right)$	$O\left(\frac{\log\sigma}{\log\log n}\right)$	$O\left(\frac{\log\sigma}{\log\log n}\right)$
[10]	$n\lg\sigma + o(n\log\sigma)$	$O(\log\log\sigma)$	$O(\log\log\sigma)$	1
[10]	$n\lg\sigma + o(n\log\sigma)$	1	$O((\log\log\sigma)^{1+\epsilon})$	$O(\log\log\sigma)$
[10]	$(1+\epsilon)n\lg\sigma$	1	$O(\log\log\sigma)$	1
[1]	$nH_k(S) + o(n\log\sigma)$	1	$O((\log\log\sigma)^{1+\epsilon})$	$O((\log\log\sigma)^{1+\epsilon})$
[1]	$nH_0(S) + o(n\log\sigma)$	$O(\log\log\sigma)$	$O(\log\log\sigma)$	1
[13]	$nH_k(S) + o(n\log\sigma)$	1	$O(\log\log\sigma)$	$O(\log\log\sigma)$
[2]	$nH_0(S) + o(n)(H_0(S)+1)$	1	$O(\log\log\sigma)$	$O(\log\log\sigma)$
[2]	$nH_0(S) + o(n)(H_0(S)+1)$	$O(\log\log\sigma)$	$O(\log\log\sigma)$	1
[2]	$(1+\epsilon)nH_0(S) + o(n)$	1	$O(\log\log\sigma)$	1
[3,4]	$nH_0(S) + o(n)$	$O\left(\frac{\log\sigma}{\log w} + 1\right)$	$O\left(\frac{\log\sigma}{\log w} + 1\right)$	$O\left(\frac{\log\sigma}{\log w} + 1\right)$
[3,4]	$nH_0(S) + o(n)(H_0(S)+1)$	1	$O\left(\log\frac{\log\sigma}{\log w}\right)$	$O(f(n,\sigma))$
[3,4]	$nH_0(S) + o(n)(H_0(S)+1)$	$O(f(n,\sigma))$	$O\left(\log\frac{\log\sigma}{\log w}\right)$	1
[3,4]	$nH_k(S) + o(n\log\sigma)$	1	$O\left(\log\frac{\log\sigma}{\log w}\right)$	$O(f(n,\sigma))$
[3,4]	$nH_k(S) + o(n\log\sigma)$	1	$O(f(n,\sigma))$	$O(f(n,\sigma))$

rank, select, insert, and delete in $O\left(\frac{\log n}{\log\log n}\right)$ time. This time bound is worst-case for the queries and amortized for the updates; the update times can be made worst-case as well at the cost of increasing the times for rank, insert, and delete from $O\left(\frac{\log n}{\log\log n}\right)$ to $O(\log n)$. Their structure is essentially a multiary wavelet tree built using rank and select data structures for dynamic sequences over sublogarithmic alphabets, much like He and Munro's or Navarro and Sadakane's, but they divide those component sequences into polylogarithmic-sized blocks and augment them with pointers such that they can ascend and descend the tree using only the pointers and rank and select on individual blocks.

Grossi, Raman, Rao, and Venturini [14] later reduced the time for access to $O(1)$ while using $nH_k(S) + o(n\log\sigma)$ bits but at the cost of being able only to replace characters instead of inserting and deleting them. The time for rank and select is the same.

Applications

Jacobson [16] first studied rank and select for representing unlabeled trees succinctly and planar graphs almost succinctly, while supporting fast navigation queries. Since then, rank and select on binary sequences have been used in succinct and compressed representations of several other combinatorial objects, such as binary relations and general graphs. Rank and select on sequences over larger alphabets have been used in succinct and compressed representations of labeled trees and permutations and in compressed full-text indexes such as compressed suffix arrays. Notice that with a data structure for rank and select that achieves compression in terms of 0th-order empirical entropy, we can build a full-text index that achieves compression in terms of kth-order empirical entropy.

Open Problems

The current main open problems regarding rank and select on static sequences are to answer access and select in $O(1)$ time while storing S in $nH_0(S) + o(n \log \sigma)$ bits when $\lg \sigma = o(w)$, to answer select in constant time and access in almost constant time while storing S in $nH_k(S) + o(n \log \sigma)$ bits when $k > 0$, and to answer access, rank, and select queries in $O(1)$ time while storing S in $nH_k(S) + o(n)$ bits when $\sigma = \lg^{O(1)} w$.

The current main open problems regarding rank and select on dynamic sequences are to achieve $O\left(\frac{\log n}{\log \log n}\right)$ worst-case time for all operations while still using compressed space, to achieve a similar space bound in terms of $H_k(S)$ instead of $H_0(S)$ while supporting the same operations, and to support a wider range of updates.

Experimental Results

The most recent experimental results for rank and select on static sequences are by Barbay et al. [2] and Claude, Navarro, and Ordóñez [6]. These results show that rank and select data structures can be implemented in a time- and space-efficient way in practice, even when the alphabet size is large. There are no current experimental results for rank and select on succinct or compressed dynamic sequences.

Cross-References

- Compressed Suffix Array
- Compressing and Indexing Structured Text
- Rank and Select Operations on Bit Strings
- Wavelet Trees

Recommended Reading

1. Barbay J, He M, Munro JI, Rao SS (2011) Succinct indexes for strings, binary relations and multilabeled trees. ACM Trans Algorithms 7(4):1–27
2. Barbay J, Claude F, Gagie T, Navarro G, Nekrich Y (2014) Efficient fully-compressed sequence representations. Algorithmica 69(1):232–268 [20] was presented in Philadelphia, USA
3. Belazzougui D, Navarro G (2012) New lower and upper bounds for representing sequences. In: Proceedings of the 20th European symposium on algorithms, Ljubljana, Slovenia, pp 181–192
4. Belazzougui D, Navarro G (2013) New lower and upper bounds for representing sequences. CoRR abs/1111.2621v2. To appear in ACM Transactions on Algorithms
5. Belazzougui D, Navarro G (2014) Alphabet-independent compressed text indexing. ACM Trans Algorithms 10(4):1–19
6. Claude F, Navarro G, Ordóñez A (2015) The wavelet matrix: an efficient wavelet tree for large alphabets. Inf Syst 47:15–32
7. Ferragina P, Manzini G, Mäkinen V, Navarro G (2007) Compressed representations of sequences and full-text indexes. ACM Trans Algorithms 3(2)
8. Fredman ML, Saks ME (1989) The cell probe complexity of dynamic data structures. In: Proceedings of the 21st symposium on theory of computing, Seattle, USA pp 345–354
9. Golynski A (2009) Cell probe lower bounds for succinct data structures. In: Proceedings of the 20th symposium on discrete algorithms, New York, USA, pp 625–634
10. Golynski A, Munro JI, Rao SS (2006) Rank/select operations on large alphabets: a tool for text indexing. In: Proceedings of the 17th symposium on discrete algorithms, Miami, USA, pp 368–373
11. Golynski A, Raman R, Rao SS (2008) On the redundancy of succinct data structures. In: Proceedings of the 11th scandinavian workshop on algorithm theory, Gothenburg, Sweden, pp 148–159
12. Grossi R, Gupta A, Vitter JS (2003) High-order entropy-compressed text indexes. In: Proceedings of the 14th symposium on discrete algorithms, Baltimore, USA, pp 841–850
13. Grossi R, Orlandi A, Raman R (2010) Optimal trade-offs for succinct string indexes. In: Proceedings of the 37th international colloquium on automata, languages and programming, Bordeaux, France, pp 678–689
14. Grossi R, Raman R, Satti SR, Venturini R (2013) Dynamic compressed strings with random access. In: Proceedings of the 40th international colloquium on languages, automata and programming, Riga, Latvia, pp 504–515
15. He M, Munro JI (2010) Succinct representations of dynamic strings. In: Proceedings of the 17th symposium on string processing and information retrieval, Los Cabos, Mexico, pp 334–346
16. Jacobson G (1989) Space-efficient static trees and graphs. In: Proceedings of the 30th symposium on foundations of computer science, Research Triangle Park, North Carolina, USA, pp 549–554
17. Navarro G, Nekrich Y (2013) Optimal dynamic sequence representations. In: Proceedings of the 24th

symposium on discrete algorithms, New Orleans, USA, pp 865–876
18. Navarro G, Nekrich Y (2013) Optimal dynamic sequence representations. CoRR abs/1206.6982v2. To appear in SIAM Journal on Computing
19. Navarro G, Sadakane K (2014) Fully functional static and dynamic succinct trees. ACM Trans Algorithms 10(3):1–39
20. Pătraşcu M (2008) Succincter. In: Proceedings of the 49th symposium on foundations of computer science, Philadelphia, USA, pp 305–313

Ranked Matching

Kavitha Telikepalli
CSA Department, Indian Institute of Science, Bangalore, India

Keywords

Popular matching

Years and Authors of Summarized Original Work

2005; Abraham, Irving, Kavitha, Mehlhorn

Problem Definition

This problem is concerned with matching a set of *applicants* to a set of *posts*, where each applicant has a *preference list*, ranking a non-empty subset of posts in order of preference, possibly involving ties. Say that a matching M is *popular* if there is no matching M' such that the number of applicants preferring M' to M exceeds the number of applicants preferring M to M'. The ranked matching problem is to determine if the given instance admits a popular matching and if so, to compute one. There are many practical situations that give rise to such large-scale matching problems involving two sets of participants – for example, pupils and schools, doctors and hospitals – where participants of one set express preferences over the participants of the other set; an allocation determined by a popular matching can be regarded as an optimal allocation in these applications.

Notations and Definitions

An instance of the *ranked matching problem* is a bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, E)$ and a partition $E = E_1 \dot\cup E_2 \ldots \dot\cup E_r$ of the edge set. Call the nodes in $\mathcal{A}$ *applicants*, the nodes in $\mathcal{P}$ *posts*, and the edges in E_i the edges of rank i. If $(a, p) \in E_i$ and $(a, p') \in E_j$ with $i < j$, say that a prefers p to p'. If $i = j$, say that a is indifferent between p and p'. An instance is *strict* if the degree of every applicant in every E_i is at most one.

A matching M is a set of edges, no two of which share an endpoint. In a matching M, a node $u \in \mathcal{A} \cup \mathcal{P}$ is either *unmatched*, or *matched* to some node, denoted by $M(u)$. Say that an applicant a *prefers* matching M' to M if (i) a is matched in M' and unmatched in M, or (ii) a is matched in both M' and M, and a prefers $M'(a)$ to $M(a)$.

Definition 1 M' is *more popular than* M, denoted by $M' \succ M$, if the number of applicants preferring M' to M exceeds the number of applicants preferring M to M'. A matching M is popular if and only if there is no matching M' that is more popular than M.

Figure 1 shows an instance with $A = \{a_1, a_2, a_3\}$, $P = \{p_1, p_2, p_3\}$, and each applicant prefers p_1 to p_2, and p_2 to p_3 (assume throughout that preferences are transitive). Consider the three symmetrical matchings $M_1 = \{(a_1, p_1), (a_2, p_2), (a_3, p_3)\}$, $M_2 = \{(a_1, p_3), (a_2, p_1), (a_3, p_2)\}$ and $M_3 = \{(a_1, p_2), (a_2, p_3), (a_3, p_1)\}$. It is easy to verify that none of these matchings is popular, since $M_1 \prec M_2$, $M_2 \prec M_3$, and $M_3 \prec M_1$. In fact, this instance admits no popular matching – the problem being, of course, that the *more popular than* relation is not acyclic, and so there need not be a maximal element.

The *ranked matching problem* is to determine if a given instance admits a popular matching, and to find such a matching, if one exists. Popular matchings may have different sizes, and a largest such matching may be smaller than a maximum-

$$\begin{array}{lccc} a_1: & p_1 & p_2 & p_3 \\ a_2: & p_1 & p_2 & p_3 \\ a_3: & p_1 & p_2 & p_3 \end{array}$$

Ranked Matching, Fig. 1 An instance for which there is no popular matching

cardinality matching. The *maximum-cardinality popular matching problem* then is to determine if a given instance admits a popular matching, and to find a *largest* such matching, if one exists.

Key Results

First consider *strict instances*, that is, instances $(\mathcal{A} \cup \mathcal{P}, E)$ where there are no ties in the preference lists of the applicants. Let n be the number of vertices and m be the number of edges in G.

Theorem 1 *For a strict instance $G = (\mathcal{A} \cup \mathcal{P}, E)$, it is possible to determine in $O(m + n)$ time if G admits a popular matching and compute one, if it exists.*

Theorem 2 *Find a maximum-cardinality popular matching of a strict instance $G = (\mathcal{A} \cup \mathcal{P}, E)$, or determine that no such matching exists, in $O(m + n)$ time.*

Next consider the general problem, where preference lists may have ties.

Theorem 3 *Find a popular matching of $G = (\mathcal{A} \cup \mathcal{P}, E)$, or determine that no such matching exists, in $O(\sqrt{n}m)$ time.*

Theorem 4 *Find a maximum-cardinality popular matching of $G = (\mathcal{A} \cup \mathcal{P}, E)$, or determine that no such matching exists, in $O(\sqrt{n}m)$ time.*

Techniques

Our results are based on a novel characterization of popular matchings. For exposition purposes, create a unique *last resort* post $l(a)$ for each applicant a and assign the edge $(a, l(a))$ a rank higher than any edge incident on a. In this way, assume that every applicant is matched, since any unmatched applicant can be allocated to his/her last resort. From now on then, matchings are *applicant-complete*, and the size of a matching is just the number of applicants not matched to their last resort. Also assume that instances have no gaps, i.e., if an applicant has a rank i edge incident to it then it has edges of all smaller ranks incident to it. First outline the characterization in strict instances and then extend it to general instances.

Strict Instances

For each applicant a, let $f(a)$ denote the most preferred post on a's preference list. That is, $(a, f(a)) \in E_1$. Call any such post p an *f-post*, and denote by $f(p)$ the set of applicants a for which $f(a) = p$.

For each applicant a, let $s(a)$ denote the most preferred non-f-post on a's preference list; note that $s(a)$ must exist, due to the introduction of $l(a)$. Call any such post p an *s-post*, and remark that f-posts are disjoint from s-posts.

Using the definitions of f-posts and s-posts, show three conditions that a popular matching must satisfy.

Lemma 1 *Let M be a popular matching.*

1. *For every f-post p, (i) p is matched in M, and (ii) $M(p) \in f(p)$.*
2. *For every applicant a, $M(a)$ can never be strictly between $f(a)$ and $s(a)$ on a's preference list.*
3. *For every applicant a, $M(a)$ is never worse than $s(a)$ on a's preference list.*

It is then shown that these three necessary conditions are also sufficient. This forms the basis of the following preliminary characterization of popular matchings.

Lemma 2 *A matching M is popular if and only if (i)* every *f-post is matched in M, and (ii) for each applicant a, $M(a) \in \{f(a), s(a)\}$.*

Given an instance graph $G = (\mathcal{A} \cup \mathcal{P}, E)$, define the *reduced graph* $G' = (\mathcal{A} \cup \mathcal{P}, E')$

as the subgraph of G containing two edges for each applicant a: one to $f(a)$, the other to $s(a)$. The authors remark that G' need not admit an applicant-complete matching, since $l(a)$ is now isolated whenever $s(a) \neq l(a)$. Lemma 2 shows that a matching is popular if and only if it belongs to the graph G' and it matches every f-post. Recall that all popular matchings are applicant-complete through the introduction of last resorts. Hence, the following characterization is immediate.

Theorem 5 *M is a popular matching of G if and only if (i) every f-post is matched in M, and (ii) M is an applicant-complete matching of the reduced graph G'.*

The characterization in Theorem 5 immediately suggests the following algorithm for solving the popular matching problem. Construct the reduced graph G'. If G' does not admit an applicant-complete matching, then G admits no popular matching. If G' admits an applicant-complete matching M, then modify M so that every f-post is matched. So for each f-post p that is unmatched in M, let a be any applicant in $f(p)$; remove the edge $(a, M(a))$ from M and instead match a to p. This algorithm can be implemented in $O(m + n)$ time. This shows Theorem 1.

Now, consider the maximum-cardinality popular matching problem. Let $\mathcal{A}_1$ be the set of all applicants a with $s(a) = l(a)$. Let $\mathcal{A}_1$ be the set of all applicants with $s(a) = l(a)$. Our target matching must satisfy conditions (i) and (ii) of Theorem 5, and among all such matchings, allocate the fewest $\mathcal{A}_1$-applicants to their last resort. This scheme can be implemented in $O(m + n)$ time. This proves Theorem 2.

General Instances

For each applicant a, let $f(a)$ denote the *set* of first-ranked posts on a's preference list. Again, refer to all such posts p as *f-posts*, and denote by $f(p)$ the set of applicants a for which $p \in f(a)$. It may no longer be possible to match *every* f-post p with an applicant in $f(p)$ (as in Lemma 1), since, for example, there may now be more f-posts than applicants. Let M be a popular matching of some instance graph $G = (\mathcal{A} \cup \mathcal{P}, E)$. Define the *first-choice graph* of G as $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$, where E_1 is the set of all rank one edges. Next the authors show the following lemma.

Lemma 3 Let M be a popular matching. Then $M \cap E_1$ is a maximum matching of G_1.

Next, work towards a generalized definition of $s(a)$. Restrict attention to rank-one edges, that is, to the graph G_1 and using M_1, partition $\mathcal{A} \cup \mathcal{P}$ into three disjoint sets. A node v is *even* (respectively *odd*) if there is an even (respectively odd) length alternating path (with respect to M_1) from an unmatched node to v. Similarly, a node v is *unreachable* if there is no alternating path (w.r.t. M_1) from an unmatched node to v. Denote by $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ the sets of even, odd, and unreachable nodes, respectively. Conclude the following facts about $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ by using the well-known Gallai–Edmonds decomposition theorem.

(a) $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ are pairwise disjoint. Every maximum matching in G_1 partitions the vertex set into the same partition of even, odd, and unreachable nodes.

(b) In any maximum-cardinality matching of G_1, every node in $\mathcal{O}$ is matched with some node in $\mathcal{E}$, and every node in $\mathcal{U}$ is matched with another node in $\mathcal{U}$. The size of a maximum-cardinality matching is $|\mathcal{O}| + |\mathcal{U}|/2$.

(c) No maximum-cardinality matching of G_1 contains an edge between two nodes in $\mathcal{O}$, or a node in $\mathcal{O}$ and a node in $\mathcal{U}$. And there is no edge in G_1 connecting a node in $\mathcal{E}$ with a node in $\mathcal{U}$.

The above facts motivate the following definition of $s(a)$: let $s(a)$ be the set of most preferred posts in a's preference list that are *even* in G_1 (note that $s(a) \neq \emptyset$, since $l(a)$ is always even in G_1). Recall that our original definition of $s(a)$ led to parts (2) and (3) of Lemma 1 which restrict the set of posts to which an applicant can be matched in a popular matching. This shows that the generalized definition leads to analogous results here.

Lemma 4 *Let M be a popular matching. Then for every applicant a, $M(a)$ can never be strictly*

between f(a) and s(a) on a's preference list and M(a) can never be worse than s(a) in a's preference list.

The following characterization of popular matchings is formed.

Lemma 5 *A matching M is popular in G if and only if (i) $M \cap E_1$ is a maximum matching of G_1, and (ii) for each applicant a, $M(a) \in f(a) \cup s(a)$.*

Given an instance graph $G = (\mathcal{A} \cup \mathcal{P}, E)$, we define the *reduced graph* $G' = (\mathcal{A} \cup \mathcal{P}, E')$ as the subgraph of G containing edges from each applicant a to posts in $f(a) \cup s(a)$. The authors remark that G' need not admit an applicant-complete matching, since $l(a)$ is now isolated whenever $s(a) \neq \{l(a)\}$. Lemma 11 tells us that a matching is popular if and only if it belongs to the graph G' and it is a maximum matching on rank one edges. Recall that all popular matchings are applicant-complete through the introduction of last resorts. Hence, the following characterization is immediate.

Theorem 6 *M is a popular matching of G if and only if (i) $M \cap E_1$ is a maximum matching of G_1, and (ii) M is an applicant-complete matching of G'.*

Using the characterization in Theorem 6, the authors now present an efficient algorithm for solving the ranked matching problem.

Popular-Matching ($G = (\mathcal{A} \cup \mathcal{P}, E)$)

1. Construct the graph $G' = (\mathcal{A} \cup \mathcal{P}, E')$, where $E' = \{(a, p) \mid p \in f(a) \cup s(a), a \in \mathcal{A}\}$.
2. Compute a maximum matching M_1 on rank one edges i.e., M_1 is a maximum matching in $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$.
   (M_1 is also a matching in G' because $E' \supseteq E_1$)
3. Delete all edges in G' connecting two nodes in the set $\mathcal{O}$ or a node in $\mathcal{O}$ with a node in $\mathcal{U}$, where $\mathcal{O}$ and $\mathcal{U}$ are the sets of odd and unreachable nodes of $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$.
   Determine a maximum matching M in the modified graph G' by augmenting M_1.
4. If M is not applicant-complete, then declare that there is no popular matching in G. Else return M.

The matching returned by the algorithm Popular-Matching is an applicant-complete matching in G' and it is a maximum matching on rank one edges. So the correctness of the algorithm follows from Theorem 6. It is easy to see that the running time of this algorithm is $O(\sqrt{n}m)$. The algorithm of Hopcroft and Karp [7] is uesd to compute a maximum matching in G_1 and identify the set of edges E' and construct G' in $O(\sqrt{n}m)$ time. Repeatedly augment M_1 (by the Hopcroft–Karp algorithm) to obtain M. This proves Theorem 3.

It is now a simple matter to solve the maximum-cardinality popular matching problem. Assume that the instance $G = (\mathcal{A} \cup \mathcal{P}, E)$ admits a popular matching. (Otherwise, the process is done.) In order to compute an applicant-complete matching in G' that is a maximum matching on rank one edges and which maximizes the number of applicants not matched to their last resort, first compute an arbitrary popular matching M' and remove all edges of the form $(a, l(a))$ from M' and from the graph G'. Call the resulting subgraph of G' as H. Determine a maximum matching N in H by augmenting M'. N need not be a popular matching, since it need not be a maximum matching in the graph G'. However, this is easy to mend. Determine a maximum matching M in G' by augmenting N. It is easy to show that M is a popular matching which maximizes the number of applicants not matched to their last resort. Since the algorithm takes $O(\sqrt{n}m)$ time, Theorem 4 is shown.

Applications

The bipartite matching problem with a graded edge set is well-studied in the economics literature, see for example [1, 10, 12]. It models some important real-world problems, including the allocation of graduates to training positions [8], and families to government-owned housing [11]. The concept of a popular matching was first

introduced by Gardenfors [5] under the name *majority assignment* in the context of the stable marriage problem [4, 6].

Various other definitions of optimality have been considered. For example, a matching is *Pareto-optimal* [1, 2, 10] if no applicant can improve his/her allocation (say by exchanging posts with another applicant) without requiring some other applicant to be worse off. Stronger definitions exist: a matching is *rank-maximal* [9] if it allocates the maximum number of applicants to their first choice, and then subject to this, the maximum number to their second choice, and so on. A matching is *maximum utility* if it maximizes $\sum_{(a,p)\in M} u_{a,p}$, where $u_{a,p}$ is the utility of allocating post p to applicant a. Neither rank-maximal nor maximum-utility matchings are necessarily popular.

Cross-References

- ▸ Hospitals/Residents Problem
- ▸ Maximum Matching
- ▸ Weighted Popular Matchings

Recommended Reading

1. Abdulkadiroğlu A, Sönmez T (1998) Random serial dictatorship and the core from random endowments in house allocation problems. Econometrica 66(3):689–701
2. Abraham DJ, Cechlárová K, Manlove DF, Mehlhorn K (2004) Pareto-optimality in house al- location problems. In: Proceedings of the 15th international symposium on algorithms and computation. LNCS, vol 3341. Springer, Sanya, pp 3–15
3. Abraham DJ, Irving RW, Kavitha T, Mehlhorn K (2005) Popular matchings. In: Proceedings of the 16th ACM-SIAM symposium on discrete algorithms. SIAM, Vancouver, pp 424–432
4. Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69:9–15
5. Gardenfors P (1975) Match making: assignments based on bilateral preferences. Behav Sci 20:166–173
6. Guseld D, Irving RW (1989) The stable marriage problem: structure and algorithms. MIT, Cambridge
7. Hopcroft JE, Karp RM (1973) A $n^{5/2}$ algorithm for maximum matchings in Bipartite graphs. SIAM J Comput 2:225–231
8. Hylland A, Zeckhauser R (1979) The ecient allocation of individuals to positions. J Polit Econ 87(2):293–314
9. Irving RW, Kavitha T, Mehlhorn K, Michail D, Paluch K (2004) Rank-maximal matchings. In: Proceedings of the 15th ACM SIAM symposium on discrete algorithms. SIAM, New Orleans, pp 68–75
10. Roth AE, Postlewaite A (1977) Weak versus strong domination in a market with indivisible goods. J Math Econ 4:131–137
11. Yuan Y (1996) Residence exchange wanted: a stable residence exchange problem. Eur J Oper Res 90: 536–546
12. Zhou L (1990) On a conjecture by Gale about one-sided matching problems. J Econ Theory 52(1): 123–135

Rate-Monotonic Scheduling

Nathan Fisher[1] and Sanjoy K. Baruah[2]
[1]Department of Computer Science, Wayne State University, Detroit, MI, USA
[2]Department of Computer Science, The University of North Carolina, Chapel Hill, NC, USA

Keywords

Fixed-priority scheduling; Rate-monotonic analysis; Real-time systems; Static-priority scheduling

Years and Authors of Summarized Original Work

1973; Liu, Layland

Problem Definition

Liu and Layland [11] introduced rate-monotonic scheduling in the context of the scheduling of recurrent real-time processes upon a computing platform comprising a single preemptive processor.

The Periodic Task Model

The *periodic task* abstraction models real-time processes that make repeated requests for computation. As defined by Liu and Layland [11], each periodic task τ_i is characterized by an ordered pair of positive real-valued parameters (C_i, T_i), where C_i is the *worst-case execution requirement* and T_i the *period* of the task. The requests for computation that are made by task τ_i (subsequently referred to as *jobs* that are *generated* by τ_i) satisfy the following assumptions:

A1: τ_i's first job arrives at system start time (assumed to equal time zero), and subsequent jobs arrive every T_i time units, i.e., one job arrives at time instant $k \times T_i$ for all integer $k \geq 0$.

A2: Each job needs to execute for at most C_i time units, i.e., C_i is the maximum amount of time that a processor would require to execute each job of τ_i, without interruption.

A3: Each job of τ_i must complete before the next job arrives. That is, each job of task τ_i must complete execution by a *deadline* that is T_i time units after its arrival time.

A4: Each task is *independent* of all other tasks – the execution of any job of task τ_i is not contingent on the arrival or completion of jobs of any other task τ_j.

A5: A job of τ_i may be *preempted* on the processor without additional execution cost. In other words, if a job of τ_i is currently executing, then it is permitted that this execution be halted and a job of a different task τ_j begins execution immediately.

A periodic task system $\tau \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_n\}$ is a collection of n periodic tasks. The *utilization* $U(\tau)$ is defined as follows:

$$U(\tau) \stackrel{\text{def}}{=} \sum_{i=1}^{n} C_i / T_i \ . \tag{1}$$

Intuitively, this denotes the fraction of time that may be spent by the processor executing jobs of tasks in τ, in the worst case.

The Rate-monotonic Scheduling Algorithm

A (uniprocessor) scheduling algorithm determines which task executes on the shared processor at each time instant. If a scheduling algorithm is guaranteed to always meet all deadlines when scheduling a task system τ, then τ is said to be *schedulable* with respect to that scheduling algorithm.

Many scheduling algorithms work as follows: at each time instant, they assign a priority to each job and select for execution the greatest-priority job with remaining execution. A *static -priority* (often called *fixed-priority*) scheduling algorithm for scheduling periodic tasks is one in which it is required that all the jobs of each periodic task be assigned the same priority.

Liu and Layland [11] proposed the *rate-monotonic* (RM) static-priority scheduling algorithm, which assigns priority to jobs according to the period parameter of the task that generates them: *the smaller the period, the higher the priority*. Hence, if $T_i < T_j$ for two tasks τ_i and τ_j, then each job of τ_i has higher priority than all jobs of τ_j and hence any executing job of τ_j will be preempted by the arrival of one of τ_i's jobs. Ties may be broken arbitrarily, but consistently – if $T_i = T_j$, then either all jobs of τ_i are assigned higher priority than all jobs of τ_j or all jobs of τ_j are assigned higher priority than all jobs of τ_i.

Key Results

First, key results from the original paper by Liu and Layland [11] are presented. Following this, results extending the work of Liu and Layland [11] are summarized.

Results from [11]

Optimality. Liu and Layland were concerned with designing "good" static- priority scheduling algorithms. They defined a notion of optimality for such algorithms: a static-priority algorithm $\mathcal{A}$ is *optimal* if any periodic task system that is

schedulable with respect to some static-priority algorithm is also schedulable with respect to $\mathcal{A}$.

Liu and Layland obtained the following result for the rate-monotonic scheduling algorithm (RM):

Theorem 1 *For periodic task systems,* RM *is an optimal static-priority scheduling algorithm.*

Schedulability testing. A *schedulability test* for a particular scheduling algorithm determines, for any periodic task system τ, whether τ is schedulable with respect to that scheduling algorithm. A schedulability test is said to be *exact* if it is the case that it correctly identifies all schedulable task systems and *sufficient* if it identifies some, but not necessarily all, schedulable task systems.

In order to derive good schedulability tests for the rate-monotonic scheduling algorithm, Liu and Layland considered the concept of *response time*. The response time of a job is defined as the elapsed time between the arrival of a job and its completion time in a schedule; the response time of a task is defined to be the largest response time that may be experienced by one of its jobs. For static- priority scheduling, Liu and Layland obtained the following result on the response time:

Theorem 2 *The maximum response time for a periodic task τ_i occurs when a job of τ_i arrives simultaneously with jobs of all higher-priority tasks. Such a time instant is known as the* critical instant *for task τ_i.*

Observe that the critical instant of the lowest-priority task in a periodic task system is also a critical instant for all tasks of higher priority. An immediate consequence of the previous theorem is that the response time of each task in the periodic task system can be obtained by simulating the scheduling of the periodic task system starting at the critical instant of the lowest-priority task. If the response time for each task τ_i obtained from such simulation does not exceed T_i, then the task system will always meet all deadlines when scheduled according to the given priority assignment. This argument immediately gives rise to a schedulability analysis test [9] for any static-priority scheduling algorithm. Since the simulation may need to be carried out until $\max_{i=1}^{n}\{T_i\}$, this schedulability test has run-time pseudo-polynomial in the representation of the task system:

Theorem 3 (Lehoczky, Sha, and Ding [9]) *Exact rate-monotonic schedulability testing of a periodic task system may be done in time pseudo-polynomial in the representation in the task system.*

Liu and Layland also derived a polynomial-time sufficient (albeit not exact) schedulability test for RM, based upon the utilization of the task system:

Theorem 4 *Let n denote the number of tasks in periodic task system τ. If $U(\tau) \leq n(2^{1/n} - 1)$, then τ is schedulable with respect to the* RM *scheduling algorithm.*

Results Since [11]

The utilization-bound sufficient schedulability test (Theorem 4) was shown to be tight in the sense that for all n, there are unschedulable task systems comprising n tasks with utilization exceeding $n(2^{1/n} - 1)$ by an arbitrarily small amount. However, tests have been devised that exploit more knowledge about tasks' period parameters. For instance, Kuo and Mok [8] provide a potentially superior utilization bound for task systems in which the task period parameters tend to be harmonically related – exact multiples of one another. Suppose that a collection of numbers is said to comprise a *harmonic chain* if for every two numbers in the set, it is the case that one is an exact multiple of the other. Let $\tilde{n}$ denote the minimum number of harmonic chains into which the period parameters $\{T_i\}_{i=1}^{n}$ of tasks in τ may be partitioned; a sufficient condition for task system τ to be RM schedulable is that

$$U(\tau) \leq \tilde{n}(2^{1/\tilde{n}} - 1) .$$

Since $\tilde{n} \leq n$ for all task systems τ, this utilization bound above is never inferior to the one in Theorem 4 and is superior for all τ for which $\tilde{n} < n$.

A different polynomial-time schedulability test was proposed by Bini, Buttazzo, and Buttazzo [4]: they showed that

$$\Pi_{i=1}^{n}((C_i/T_i)+1) \leq 2$$

is sufficient to guarantee that the periodic task system $\{\tau_1, \tau_2, \ldots, \tau_n\}$ is rate-monotonic schedulable. This test is commonly referred to as the *hyperbolic* schedulability test for rate-monotonic schedulability. The hyperbolic test is in general known to be superior to the utilization-based test of Theorem 4 – see [4] for details.

Other work done since the seminal paper of Liu and Layland has focused on relaxing the assumptions of the periodic task model. The (implicit-deadline) *sporadic* task model relaxed assumption A17 by allowing T_i to be the *minimum* (rather than exact) separation between arrivals of successive jobs of task τ_i. It turns out that the Theorems 1–4 continue to hold for systems of such tasks as well.

A more general sporadic task model has also been studied that relaxes assumption A17 in addition to assumption A17, by allowing for the explicit specification of a deadline parameter for each task (which may differ from the task's period). The *deadline-monotonic* scheduling algorithm [10] generalizes rate-monotonic scheduling to such task systems.

Work has also been done [2, 12] in removing the independence assumption of A4, by allowing for different tasks to use critical sections to access non-preemptable serially reusable resources.

Applications

The periodic task model has been invaluable for modeling several different types of systems. For control systems, the periodic task model is well suited for modeling the periodic requests and computations of sensors and actuators. Multimedia and network applications also typically involve computation of periodically arriving packets and data.

Many of the results described in section "Key Results" above have been integrated into powerful tools, techniques, and methodologies for the design and analysis of real-time application systems [1, 7]. The general methodology framework is commonly referred to as the *rate-monotonic analysis* (RMA) methodology. Furthermore, most operating systems provide standard primitives for supporting rate-monotonic scheduling.

Open Problems

There are plenty of interesting and challenging open problems in real-time scheduling theory; however, most of these are concerned with extensions to the basic task and scheduling model considered in the original Liu and Layland paper [11]. Perhaps the most interesting open problem with respect to the task model in [11] is regarding the computational complexity of schedulability analysis of static-priority scheduling. Recent research by Eisenbrand and Rothvoß [5] has shown that determining the maximum response time of any periodic task is NP-hard. This result shows that any exact schedulability test that utilizes response time cannot run in polynomial time (unless P = NP); however, it does not settle the open question of whether there are polynomial-time schedulability tests for static-priority periodic task systems that do not (implicitly or explicitly) calculate task response time.

URLs to Code and Data Sets

Research efforts have been made to develop a standardized methodology for evaluating the efficacy and efficiency of algorithms and analysis proposed for rate-monotonic scheduling problems. Bini and Buttazzo [3] derived an unbiased method for synthetically generating random periodic task systems (http://retis.sssup.it/~bini/publications/2005BinBut.html). Additionally, researchers have proposed suites of benchmarks as representative of embedded and real-time applications in practice. Notably, the Mälardalen WCET benchmarks [6] (http://

www.mrtc.mdh.se/projects/wcet/benchmarks.html) maintain a collection of programs that are typical for real-time applications.

Cross-References

- ▸ List Scheduling
- ▸ Online Load Balancing of Temporary Tasks
- ▸ Shortest Elapsed Time First Scheduling

Recommended Reading

1. Audsley N, Burns A, Wellings A (1993) Deadline monotonic scheduling theory and application. Control Eng Pract 1:71–78
2. Baker TP (1991) Stack-based scheduling of real-time processes. Real-Time Syst Int J Time-Crit Comput 3:67–100
3. Bini E, Buttazzo G (2005) Measuring the performance of schedulability tests. Real-Time Syst 30:129–154
4. Bini E, Buttazzo GC, Buttazzo GM (2003) Rate monotonic scheduling: the hyperbolic bound. IEEE Trans Comput 52:933–942
5. Eisenbrand F, Rothvoß T (2008) Static-priority real-time scheduling: response time computation is NP-hard. In: Proceedings of the IEEE real-time systems symposium, Barcelona, Nov 2008. IEEE Computer Society Press, pp 397–406
6. Gustafsson J, Betts A, Ermedahl A, Lisper B (2010) The Mälardalen WCET benchmarks – past, present and future. In: Proceedings of 10th international workshop on worst-case execution time analysis (WCET'2010), Brussels, July 2010, pp 137–147
7. Klein M, Ralya T, Pollak B, Obenza R, Harbour MG (1993) A Practitioner's handbook for real-time analysis: guide to rate monotonic analysis for real-time systems. Kluwer Academic, Boston
8. Kuo T-W, Mok AK (1991) Load adjustment in adaptive real-time systems. In: Proceedings of the IEEE real-time systems symposium, San Antonio, Dec 1991. IEEE Computer Society Press, pp 160–171
9. Lehoczky J, Sha L, Ding Y (1989) The rate monotonic scheduling algorithm: exact characterization and average case behavior. In: Proceedings of the real-time systems symposium, Santa Monica, Dec 1989. IEEE Computer Society Press, pp 166–171
10. Leung J, Whitehead J (1982) On the complexity of fixed-priority scheduling of periodic, real-time tasks. Perform Eval 2:237–250
11. Liu C, Layland J (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. J ACM 20:46–61
12. Rajkumar R (1991) Synchronization in real-time systems – a priority inheritance approach. Kluwer Academic, Boston

Rectilinear Spanning Tree

Hai Zhou
Electrical Engineering and Computer Science (EECS) Department, Northwestern University, Evanston, IL, USA

Keywords

Metric minimum spanning tree; Rectilinear spanning graph

Years and Authors of Summarized Original Work

2001; Zhou, Shenoy, Nicholls

Problem Definition

Given a set of n points in a plane, a spanning tree is a set of edges that connects all the points and contains no cycles. When each edge is weighted using some distance metric of the incident points, the *metric minimum spanning tree* is a tree whose sum of edge weights is minimum. If the Euclidean distance (L_2) is used, it is called the *Euclidean minimum spanning tree*; if the rectilinear distance (L_1) is used, it is called the *rectilinear minimum spanning tree*.

Since the minimum spanning tree problem on a weighted graph is well studied, the usual approach for metric minimum spanning tree is to first define a weighted graph on the set of points and then to construct a spanning tree on it.

Much like a connection graph is defined for the maze search [4], a spanning graph can be defined for the minimum spanning tree construction.

Definition 1 Given a set of points V in a plane, an undirected graph $G = (V, E)$ is called a *spanning graph* if it contains a minimum spanning tree of V in the plane.

Since spanning graphs with fewer edges give more efficient minimum spanning tree construction, the *cardinality* of a spanning graph is defined as its number of edges. It is easy to see that a complete graph on a set of points contains all spanning trees, thus is a spanning graph. However, such a graph has a cardinality of $O(n^2)$. A rectilinear spanning graph of cardinality $O(n)$ can be constructed within $O(n \log n)$ time [6] and will be described here.

Minimum spanning tree algorithms usually use two properties to infer the inclusion and exclusion of edges in a minimum spanning tree. The first property is known as the *cut property*. It states that an edge of smallest weight crossing any partition of the vertex set into two parts belongs to a minimum spanning tree. The second property is known as the *cycle property*. It says that an edge with largest weight in any cycle in the graph can be safely deleted. Since the two properties are stated in connection with the construction of a minimum spanning tree, they are useful for a spanning graph.

Key Results

Using the terminology given in [3], the *uniqueness property* is defined as follows.

Definition 2 Given a point s, a region R has the *uniqueness property* with respect to s if for every pair of points $p, q \in R$, $||pq|| < \max(||sp||, ||sq||)$. A partition of space into a finite set of disjoint regions is said to have the uniqueness property with respect to s if each of its regions has the uniqueness property with respect to s.

The notation $||sp||$ is used to represent the distance between s and p under the L_1 metric. Define the *octal partition* of the plane with respect to s as the partition induced by the two rectilinear lines and the two 45° lines through s, as shown in Fig. 1a. Here, each of the regions R_1 through R_8 includes only one of its two bounding half lines as shown in Fig. 1b. It can be shown that the octal partition has the uniqueness property.

Lemma 1 *Given a point s in the plane, the octal partition with respect to s has the uniqueness property.*

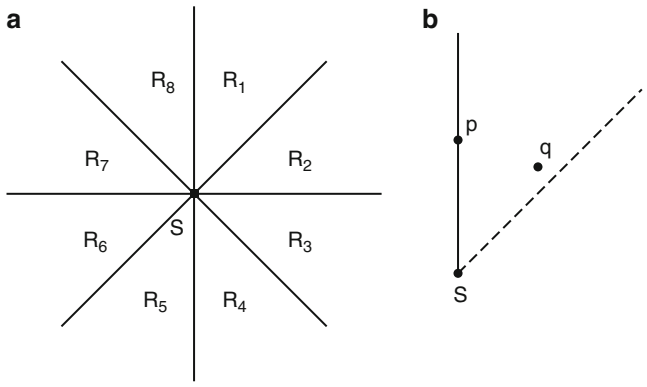
Proof To show a partition has the uniqueness property, it needs to prove that each region of the partition has the uniqueness property. Since the regions R_1 through R_8 are similar to each other, a proof for R_1 will be sufficient.

The points in R_1 can be characterized by the following inequalities:

$$x \geq x_s,$$

$$x - y < x_s - y_s.$$

Suppose there are two points p and q in R_1. Without loss of generality, it can be assumed $x_p \leq x_q$. If $y_p \leq y_q$, then $||sq|| = ||sp|| +$



Rectilinear Spanning Tree, Fig. 1 Octal partition and the uniqueness property

$||pq|| > ||pq||$. Therefore it only needs to consider the case when $y_p > y_q$. In this case,

$$\begin{aligned}\|pg\| &= |x_p - x_q| + |y_p - y_p| \\ &= x_q - x_p + y_p - y_q \\ &= (x_q - y_q) + y_p - x_p \\ &< (x_s - y_s) + y_p - x_s \\ &= y_p - y_s \\ &\leq x_p - x_s + y_p - y_s \\ &= \|sp\|.\end{aligned}$$

Given two points p, q in the same octal region of point s, the uniqueness property says that $||pq|| < \max(||sp||, ||sq||)$. Consider the cycle on points s, p, and q. Based on the cycle property, only one point with the minimum distance from s needs to be connected to s. An interesting property of the octal partition is that the contour of equidistant points from s forms a line segment in each region. In regions R_1, R_2, R_5, and R_6, these segments are captured by an equation of the form $x + y = c$; in regions R_3, R_4, R_7, and R_8, they are described by the form $x - y = c$.

From each point s, the closest neighbor in each octant needs to be found. It will be described how to efficiently compute the neighbors in R_1 for all points. The case for other octant is symmetric. For the R_1 octant, a sweep line algorithm will run on all points according to nondecreasing $x + y$. During the sweep, maintained will be an *active* set consisting of points whose nearest neighbors in R_1 are yet to be discovered. When a point p is processed, all points in the active set that have p in their R_1 regions will be found. If s is such a point in the active set, since points are scanned in nondecreasing $x + y$, then p must be the nearest point in R_1 for s. Therefore, the edge sp will be added and s will be deleted from the active set. After processing those active points, the point p will be added into the active set. Each point will be added and deleted at most once from the active set.

A fundamental operation in the sweep line algorithm is to find a subset of active points such that a given point p is in their R_1 regions. Based on the observation that point p is in the R_1 region of point s if and only if s is in the R_5 region of p, it needs to find the subset of active points in the R_5 region of p. Since R_5 can be represented as a two-dimensional range $(-\infty, x_p] \times (x_p - y_p, +\infty)$ on $(x, x - y)$, a priority search tree [1] can be used to maintain the active point set. Since each of the insertion and deletion operations takes $O(\log n)$ time, and the query operation takes $O(\log n + k)$ time where k is the number of objects within the range, the total time for the sweep is $O(n \log n)$. Since other regions can be processed in the similar way as in R_1, the algorithm is running in $O(n \log n)$ time. Priority search tree is a data structure that relies on maintaining a balanced structure for the fast query time. This works well for static input sets. When the input set is dynamic, rebalancing the tree can be quite challenging. Fortunately, the active set has a structure that can be explored for an alternate representation. Since a point is deleted from the active set if a point in its R_1 region is found, no point in the active set can be in the R_1 region of another point in the set.

Lemma 2 *For any two points p, q in the active set, it must be* $x_p \neq x_q$, *and if* $x_p < x_q$, *then* $x_p - y_p \leq x_q - y_q$.

Based on this property, the active set can be ordered in increasing order of x. This implies a nondecreasing order on $x - y$. Given a point s, the points which have s in their R_1 region must obey the following inequalities:

$$\begin{aligned}x &\leq x_s, \\ x - y &> x_s - y_s.\end{aligned}$$

To find the subset of active points which have s in their R_1 regions, it can first find the largest x such that $x \leq x_s$ and then proceed in decreasing order of x until $x - y \geq x_s - y_s$. Since the ordering is kept on only one dimension, using any binary search tree with $O(\log n)$ insertion, deletion, and query time will also give us an $O(n \log n)$ time algorithm. Binary search trees also need to be balanced. An alternative is to use

skip lists [2] which use randomization to avoid the problem of explicit balancing but provide $O(\log n)$ expected behavior.

A careful study also shows that after the sweep process for R_1, there is no need to do the sweep for R_5, since all edges needed in that phase are either connected or implied. Moreover, based on the information in R_5, the number of edge connections can be further reduced. When the sweep step processes point s, it finds a subset of active points which have s in their R_1 regions. Without lost of generality, suppose p and q are two of them. Then p and q are in the R_5 region of s, which means $||pq|| < \max(||sp||, ||sq||)$. Therefore, it needs only to connect s with the nearest active point.

Since R_1 and R_2 have the same sweep sequence, they can be processed together in one pass. Similarly, R_3 and R_4 can be processed together in another pass. Based on the above discussion, the pseudo-code of the algorithm is presented in Fig. 2.

The correctness of the algorithm is stated in the following theorem.

Theorem 1 *Given n points in the plane, the rectilinear spanning graph algorithm constructs a spanning graph in $O(n \log n)$ time, and the number of edges in the graph is $O(n)$.*

Proof The algorithm can be considered as deleting edges from the complete graph. As described, all deleted edges are redundant based on the cycle property. Thus, the output graph of the algorithm will contain at least one rectilinear minimum spanning tree.

In the algorithm, each given point will be inserted and deleted at most once from the active set for each of the four regions R_1 through R_4. For each insertion or deletion, the algorithm requires $O(\log n)$ time. Thus, the total time is upper bounded by $O(n \log n)$. The storage is needed only for active sets, which is at most $O(n)$.

Applications

Rectilinear minimum spanning tree problem has wide applications in VLSI CAD. It is frequently used as a metric of wire length estimation during placement. It is often constructed to approximate a minimum Steiner tree and is also a key step in many Steiner tree heuristics. It is also used in an approximation to the traveling salesperson problem which can be used to generate scan chains in testing. It is important to emphasize that for real- world applications, the input sizes are usually very large. Since it is a problem that will be computed hundreds of thousands times and many of them will have very large input sizes, the rectilinear minimum spanning tree problem needs a very efficient algorithm.

Experimental Results

The experimental results using the rectilinear spanning graph (RSG) followed by Kruskal's algorithm for a rectilinear minimum spanning tree were reported in Zhou et al. [5]. Two other approaches were compared. The first approach used

Rectilinear Spanning Tree, Fig. 2 The rectilinear spanning graph algorithm

```
Rectilinear Spanning Graph Algorithm
  for (i = 0; i < 2; i + +) {
    if (i == 0) sort points according to x + y;
    else sort points according to x – y;
    A[1] = A[2] = ∅;
    for each point p in the order {
        find points in A[1], A[2] such that p is in their
            R_{2i+1} and R_{2i+2} regions, respectively;
        connect p with the nearest point in each subset;
        delete the subsets from A[1], A[2], respectively;
        add p to A[1], A[2];
    }
  }
```

Rectilinear Spanning Tree, Table 1 Experimental results

Input		Complete		Bound degree		RSG	
Orig	Distinct	#edge	Time	#edge	Time	#edge	Time
1,000	999	498,501	5.095 s	3,878	0.299 s	2,571	0.112 s
2,000	1,996	1,991,010	24.096 s	7,825	0.996 s	5,158	0.218 s
4,000	3,995	7,978,015	2 min 7.233 s	15,761	3.452 s	10,416	0.337 s
6,000	5,991	17,943,045	5 min 54.697 s	23,704	7.515 s	15,730	0.503 s
8,000	7,981	31,844,190	13 min 7.682 s	31,624	13.141 s	21,149	0.672 s
10,000	9,962	49,615,741	–	39,510	20.135 s	26,332	0.934 s
12,000	11,948	–	–	47,424	32.300 s	31,586	1.052 s
14,000	13,914	–	–	55,251	46.842 s	36,853	1.322 s
16,000	15,883	–	–	63,089	1 min 3.759 s	42,251	1.486 s
18,000	17,837	–	–	70,876	1 min 19.812 s	47,511	1.701 s
20,000	19,805	–	–	78,723	1 min 45.792 s	52,732	1.907 s

the complete graph on the point set as the input to Kruskal's algorithm. The second approach is an implementation of concepts described in [3]; namely, for each point, scan all other points but only connect the nearest one in each quadrant region. With sizes ranging from 1,000 to 20,000, randomly generated point sets were used in the experiments. The results are reproduced here in Table 1. The first column gives the number of generated points; the second column gives the number of distinct points. For each approach, the number of edges in the given graph and the total running time are reported. For input size larger than 10,000, the complete graph approach simply runs out of memory.

Cross-References

▸ Rectilinear Steiner Tree

Recommended Reading

1. McCreight EM (1985) Priority search trees. SIAM J Comput 14:257–276
2. Pugh W (1990) Skip lists: a probabilistic alternative to balanced trees. Commun ACM 33:668–676
3. Robins G, Salowe JS (1995) Low-degree minimum spanning tree. Discret Comput Geom 14:151–165
4. Zheng SQ, Lim JS, Iyengar SS (1996) Finding obstacle-avoiding shortest paths using implicit connection graphs. IEEE Trans Comput Aided Des 15:103–110
5. Zhou H, Shenoy N, Nicholls W (2001) Efficient minimum spanning tree construction without delaunay triangulation. In: Proceedings of Asian and South Pacific design automation conference, Yokohama
6. Zhou H, Shenoy N, Nicholls W (2002) Efficient spanning tree construction without delaunay triangulation. Inf Proc Lett 81:271–276

Rectilinear Steiner Tree

Hai Zhou
Electrical Engineering and Computer Science (EECS) Department, Northwestern University, Evanston, IL, USA

Keywords

Metric minimum Steiner tree; Shortest routing tree

Years and Authors of Summarized Original Work

2003; Zhou

Problem Definition

Given n points on a plane, a Steiner minimal tree connects these points through some extra points (called Steiner points) to achieve a minimal total

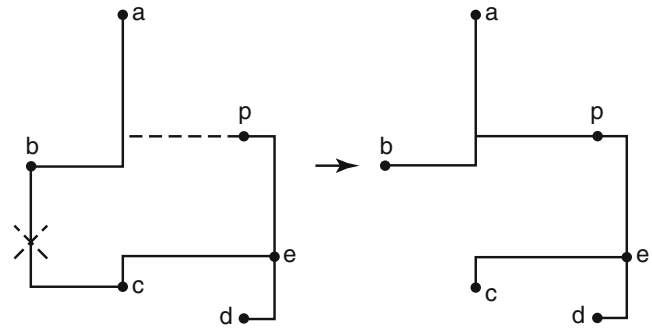
length. When the length between two points is measured by the rectilinear distance, the tree is called a rectilinear Steiner minimal tree.

Because of its importance, there is much previous work to solve the SMT problem. These algorithms can be grouped into two classes: exact algorithms and heuristic algorithms. Since SMT is NP-hard, any exact algorithm is expected to have an exponential worst-case running time. However, two prominent achievements must be noted in this direction. One is the *GeoSteiner* algorithm and implementation by Warme, Winter, and Zacharisen [14, 15], which is the current fastest exact solution to the problem. The other is a Polynomial Time Approximation Scheme (PTAS) by Arora [1], which is mainly of theoretical importance. Since exact algorithms have long running time, especially on large input sizes, much more previous efforts were put on heuristic algorithms. Many of them generate a Steiner tree by improving on a minimal spanning tree topology [7], since it was proved that a minimal spanning tree is a 3/2 approximation of a SMT [8]. However, since the backbones are restricted to the minimal spanning tree topology in these approaches, there is a reported limit on the improvement ratios over the minimal spanning trees. The iterated 1-Steiner algorithm by Kahng and Robins [10] is an early approach to deviate from that restriction, and an improved implementation [6] is a champion among such programs in public domain. However, the implementation in [10] has a running time of $O(n^4 \log n)$, and the implementation in [6] has a running time of $O(n^3)$. A much more efficient approach was later proposed by Borah et al. [2]. In their approach, a spanning tree is iteratively improved by connecting a point to an edge and deleting the longest edge on the created circuit. Their algorithm and implementation had a worst-case running time of $\Theta(n^2)$, even though an alternative $O(n \log n)$ implementation was also proposed. Since the backbone is no longer restricted to the minimal spanning tree topology, its performance was reported to be similar to the iterated 1-Steiner algorithm [2]. A recent effort in this direction is a new heuristic by Mandoiu et al. [11] which is based on a 3/2 approximation algorithm of the metric Steiner tree problem on quasi-bipartite graphs [12]. It performs slightly better than the iterated 1-Steiner algorithm, but its running time is also slightly longer than the iterated 1-Steiner algorithm (with the empty rectangle test [11] used). More recently, Chu [3] and Chu and Wong [4] proposed an efficient lookup table- based approach for rectilinear Steiner tree construction.

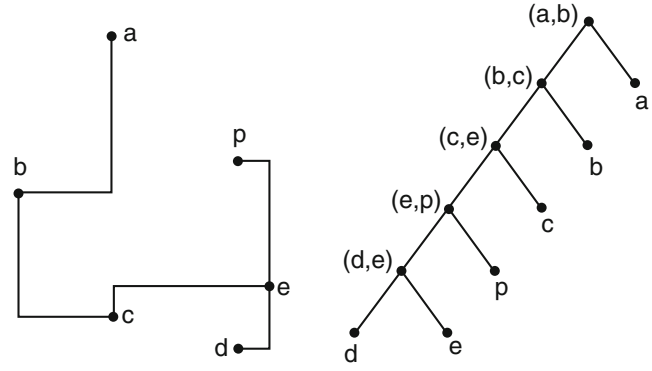
Key Results

The presented algorithm is based on the edge substitution heuristic of Borah et al. [2]. The heuristic works as follows. It starts with a minimal spanning tree and then iteratively considers connecting a point (e.g., p in Fig. 1) to a nearby edge (e.g., (a, b)) and deleting the longest edge $((b, c))$ on the circuit thus formed. The algorithm employs the spanning graph [17] as a backbone of the computation: it is first used to generate the initial minimal spanning tree and then to generate point-edge pairs for tree improvements. This kind of unification happens also in the spanning tree computation and the longest edge computation for each point-edge pair: using Kruskal's algorithm with disjoint set operations (instead of Prim's algorithm) [5] will unify these two computations.

In order to reduce the number of point-edge pair candidates from $O(n^2)$ to $O(n)$, Borah et al. suggested to use the visibility of a point from an edge, that is, only a point visible from an edge can be considered to connect to that edge. This requires a sweep line algorithm to find visibility relations between points and edges. In order to skip this complex step, the geometrical proximity information embedded within the spanning graph is leveraged. Since a point has eight nearest points connected around it, it is observed that if a point is visible to an edge, then the point is *usually* connected in the graph to at least one end point. In the algorithm, the spanning graph is used to generate point-edge pair candidates. For each edge in the current tree, all points that are neighbors of either of the end points will be considered to form point-edge pairs with the edge. Since the cardinality



Rectilinear Steiner Tree, Fig. 1 Edge substitution by Borah et al.



Rectilinear Steiner Tree, Fig. 2 A minimal spanning tree and its merging binary tree

of the spanning graph is $O(n)$, the number of possible point-edge pairs generated in this way is also $O(n)$.

When connecting a point to an edge, the longest edge on the formed circuit needs to be deleted. In order to find the corresponding longest edge for each point-edge pair efficiently, it explores how the spanning tree is formed through Kruskal's algorithm. This algorithm first sorts the edges into nondecreasing lengths, and each edge is considered in turn. If the end points of the edge have been connected, then the edge will be excluded from the spanning tree; otherwise, it will be included. The structure of these connecting operations can be represented by a binary tree, where the leaves represent the points and the internal nodes represent the edges. When an edge is included in the spanning tree, a node is created representing the edge and has as its two children the trees representing the two components connected by this edge. To illustrate this, a spanning tree with its representing binary tree is shown in Fig. 2. As can be seen, the longest edge between two points is the least common ancestor of the two points in the binary tree. For example, the longest edge between p and b in Fig. 2 is (b, c), which is the least common ancestor of p and b in the binary tree. To find the longest edge on the circuit formed by connecting a point to an edge, it needs to find the longest edge between the point and one end point of the edge that are in the same component before connecting the edge. For example, consider the pair p and (a, b); since p and b are in the same component before connecting (a, b), the edge that needs to be deleted is the longest between p and b.

Based on the above discussion, the pseudocode of the algorithm can be described in Fig. 3. At the beginning of the algorithm, Zhou et al.'s rectilinear spanning graph algorithm [17] is used to generate the spanning graph G for the given set of points. Then, Kruskal's algorithm is used on the graph to generate a minimal spanning tree. The data structure of disjoint sets [5] is used to merge components and check whether two points are in the same component (the first **for** loop). During this process, the merging binary tree and

Rectilinear Steiner Tree, Fig. 3 The rectilinear Steiner tree algorithm

Rectilinear Steiner Tree (RST) Algorithm

```
T = ∅;
Generate the spanning graph G by RSG algorithm;
for (each edge (u, v) ∈ G in non-decreasing length) {
    s1 = find_set(u); s2 = find_set(v);
    if (s1 ! = s2) {
        add (u, v) in tree T;
        for (each neighbor w of u, v in G)
            if (s1 == find_set(w))
                lca_add_query(w, v, (u, v));
            else lca_add_query(w, v, (u, v));
        lca_tree_edge((u, v), s1.edge);
        lca_tree_edge((u, v), s2.edge);
        s = union_set(s1, s2); s.edge = (u, v) ;
    }
}
generate point-edge pairs by lca_answer_queries;
for (each pair (p, (a, b), (c, d)) in non-increasing positive gains)
    if ((a, b), (c, d) has not been deleted from T) {
        connect p to (a, b) by adding three edges to T;
        delete (a, b), (c, d) from T;
    }
```

the queries for least common ancestors of all point-edge pairs are also generated. Here, s, $s1$, and $s2$ represent disjoint sets, and each records the root of the component in the merging binary tree. For each edge (u, v) adding to T, each neighbor w of either u or v will be considered to connect to (u, v). The longest edge for this pair is the least common ancestor of w, u or w, v depending on which point is in the same component as w. The procedure *lca_add_query* is used to add this query. Connecting the two components by (u, v) will also be recorded in the merging binary tree by the procedure *lca_tree_edge*. After generating the minimal spanning tree, it also has the corresponding merging binary tree and the least common ancestor queries ready. Using Tarjan's off-line least common ancestor algorithm [5] (represented by *lca_answer_queries*), it can generate all longest edges for the pairs. With the longest edge for each point-edge pair, the gain of connecting the point to the edge can be calculated. Then, each of the point to edge connections will be realized in a nonincreasing order of their gains. A connection can only be realized if both the connection edge and deletion edge have not been deleted yet.

The running time of the algorithm is dominated by the spanning graph generation and edge sorting, which take $O(n \log n)$ time. Since the number of edges in the spanning graph is $O(n)$, both Kruskal's algorithm and Tarjan's off-line least common ancestor algorithm take $O(n\alpha(n))$ time, where $\alpha(n)$ is the inverse of Ackermann's function, which grows extremely slow.

Applications

The Steiner minimal tree (SMT) problem has wide applications in VLSI CAD. A SMT is generally used in initial topology creation for noncritical nets in physical synthesis. For timing

critical nets, minimization of wire length is generally not enough. However, since most nets are noncritical in a design and a SMT gives the most desirable route of such a net, it is often used as an accurate estimation of congestion and wire length during floor planning and placement. This implies that a Steiner tree algorithm will be invoked millions of times. On the other hand, there exist many large pre-routes in modern VLSI design. The pre-routes are generally modeled as large sets of points, thus increasing the input sizes of the Steiner tree problem. Since the SMT is a problem that will be computed millions of times and many of them will have very large input sizes, highly efficient solutions with good performance are desired.

Experimental Results

As reported in [16], the first set of experiments were conducted on a Linux system with a 928 MHz Intel Pentium III processor and 512 M memory. The *RST* algorithm was compared with other publicly available programs: the exact algorithm *GeoSteiner* (version 3.1) by Warme, Winter, and Zacharisen [14]; the Batched Iterated 1-Steiner (*BI1S*) by Robins; and the Borah et al.'s algorithm implemented by Madden (*BOI*).

Table 1 gives the results of the first set of experiments. For each input size ranging from 100 to 5,000, 30 different test cases are randomly generated through the *rand_points* program in *GeoSteiner*. The improvement ratios of a Steiner tree *St* over its corresponding minimal spanning tree *MST* are defined as `100 × (MST - St)/MST`. For each input size, the average of the improvement ratios and the average running time (in seconds) on each of the programs are reported. As can be seen, *RST* always gives better improvements than *BOI* with less running times.

The second set of experiments compared *RST* with Borah's implementation of Borah et al.'s algorithm (*Borah*), Rohe's Prim-based algorithm (*Rohe*) [13], and Kahng et al.'s Batched Greedy Algorithm (*BGA*) [9]. They were run on a different Linux system with a 2.4 GHz Intel Xeon processor and 2 G memory. Besides the randomly generated test cases, the VLSI industry test cases used in [9] were also used. The results are reported in Table 2.

Cross-References

▸ Rectilinear Spanning Tree

Recommended Reading

1. Arora S (1998) Polynomial-time approximation schemes for Euclidean TSP and other geometric problem. J ACM 45:753–782
2. Borah M, Owens RM, Irwin MJ (1994) An edge-based heuristic for steiner routing. IEEE Trans Comput Aided Des 13:1563–1568

Rectilinear Steiner Tree, Table 1 Comparison with other algorithms I

Input size	*GeoSteiner*		*BI1S*		*BOI*		*RST*	
	Improve	Time	Improve	Time	Improve	Time	Improve	Time
100	11.440	0.487	10.907	0.633	9.300	0.0267	10.218	0.004
200	11.492	3.557	10.897	4.810	9.192	0.1287	10.869	0.020
300	11.492	12.685	10.931	18.770	9.253	0.2993	10.255	0.041
500	11.525	72.192	–	–	9.274	0.877	10.381	0.084
800	11.343	536.173	–	–	9.284	2.399	10.719	0.156
1,000	–	–	–	–	9.367	4.084	10.433	0.186
2,000	–	–	–	–	9.326	31.098	10.523	0.381
3,000	–	–	–	–	9.390	104.919	10.449	0.771
5,000	–	–	–	–	9.356	307.977	10.499	1.330

Rectilinear Steiner Tree, Table 2 Comparison with other algorithms II

Input size	*BGA*		*Borah*		*Rohe*		*RST*	
	Improve	Time	Improve	Time	Improve	Time	Improve	Time
Randomly generated testcases								
100	10.272	0.006	10.341	0.004	9.617	0.000	10.218	0.002
500	10.976	0.068	10.778	0.178	10.028	0.010	10.381	0.041
1,000	10.979	0.162	10.829	0.689	9.768	0.020	10.433	0.121
5,000	11.012	1.695	11.015	25.518	10.139	0.130	10.499	0.980
10,000	11.108	4.135	11.101	249.924	10.111	0.310	10.559	2.098
50,000	11.120	59.147	–	–	10.109	1.890	10.561	13.029
100,000	11.098	161.896	–	–	10.079	4.410	10.514	28.527
500,000	–	–	–	–	10.059	27.210	10.527	175.725
VLSI testcases								
337	6.434	0.035	6.503	0.037	5.958	0.010	5.870	0.016
830	3.202	0.070	3.185	0.213	3.102	0.020	2.966	0.033
1,944	7.850	0.342	7.772	2.424	6.857	0.040	7.533	0.238
2,437	7.965	0.549	7.956	4.502	7.094	0.050	7.595	0.408
2,676	8.928	0.623	8.994	3.686	8.067	0.060	8.507	0.463
12,052	8.450	4.289	8.465	232.779	7.649	0.300	8.076	2.281
22,373	9.848	11.330	9.832	1,128.365	8.987	0.570	9.462	4.605
34,728	9.046	18.416	9.010	2,367.629	8.158	0.900	8.645	5.334

3. Chu C (2004) FLUTE: Fast lookup table based wirelength estimation technique. In: Proceedings of the international conference on computer-aided design, San Jose, pp 696–701
4. Chu C, Wong YC (2005) Fast and accurate rectilinear steiner minimal tree algorithm for VLSI design. In: Proceedings of the international symposium on physical design, San Francisco, pp 28–35
5. Cormen TH, Leiserson CE, Rivest RL (1989) Introduction to algorithms. MIT, Cambridge
6. Griffith J, Robins G, Salowe JS, Zhang T (1994) Closing the gap: near-optimal steiner trees in polynomial time. IEEE Trans Comput Aided Des 13:1351–1365
7. Ho JM, Vijayan G, Wong CK (1990) New algorithms for the rectilinear steiner tree problem. IEEE Trans Comput Aided Des 9:185–193
8. Hwang FK (1976) On steiner minimal trees with rectilinear distance. SIAM J Appl Math 30:104–114
9. Kahng AB, Mandoiu II, Zelikovsky A (2003) Highly scalable algorithms for rectilinear and octilinear steiner trees. In: Proceedings of the Asia and South Pacific design automation conference, Kitakyushu, pp 827–833
10. Kahng AB, Robins G (1992) A new class of iterative steiner tree heuristics with good performance. IEEE Trans Comput Aided Des 11:893–902
11. Mandoiu II, Vazirani VV, Ganley JL (1999) A new heuristic for rectilinear steiner trees. In: Proceedings of the international conference on computer-aided design, San Jose
12. Rajagopalan S, Vazirani VV (1999) On the bidirected cut relaxation for the metric steiner tree problem. In: Proceedings of the 10th ACM-SIAM symposium on discrete algorithms, Baltimore, pp 742–751
13. Rohe A (2001) Sequential and parallel algorithms for local routing. Ph.D. thesis, Bonn University, Bonn
14. Warme DM, Winter P, Zacharisen M (2003) GeoSteiner 3.1 package. ftp://ftp.diku.dk/diku/users/martinz/geosteiner-3.1.tar.gz. Accessed Oct 2003
15. Warme DM, Winter P, Zacharisen M (1998) Exact algorithms for plane steiner tree problems: a computational study. Tech. Rep. DIKU-TR-98/11, Dept. of Computer Science, University of Copenhagen
16. Zhou H (2003) Efficient Steiner tree construction based on spanning graphs. In: ACM international symposium on physical design, Monterey
17. Zhou H, Shenoy N, Nicholls W (2002) Efficient spanning tree construction without delaunay triangulation. Inf Process Lett 81:271–276

Recursive Separator Decompositions for Planar Graphs

Shay Mozes
Efi Arazi School of Computer Science, The Interdisciplinary Center (IDC), Herzliya, Israel

Keywords

Divide-and-conquer; Planar separators; *r*-division

Years and Authors of Summarized Original Work

1987; Frederickson
1995; Goodrich
2013; Klein, Mozes, Sommer

Problem Definition

Graph decompositions are the basis for many divide-and-conquer algorithms. Two main properties make a decomposition useful. The first is *balance*, namely, that the parts of the decomposition have roughly the same size. Balanced decompositions lead to logarithmic depth recursion. The second is small *overlap* between the parts of the decomposition. The overlap affects the time it takes to combine solutions of different parts into a solution for the union of the parts.

A decomposition of a graph G is a collection of subgraphs of G, called *regions*, whose union is G. A *decomposition tree* of G is a tree $\mathcal{T}$ whose nodes correspond to subgraphs of G. The root of $\mathcal{T}$ consists of the entire graph G. For a node v of $\mathcal{T}$ that corresponds to a subgraph R, the children $v_1, v_2, \ldots, v_k$ of v correspond to subgraphs of R whose union is R. Every maximal set D of nodes of $\mathcal{T}$, such that no node in D is an ancestor of another (i.e., every maximal antichain in $\mathcal{T}$ with respect to the ancestry partial order), corresponds to a decomposition of G.

A vertex v of G that belongs to a unique region R in a decomposition is called an *interior* vertex (of R). A vertex v that belongs to more than one region is called a *boundary* vertex.

Let G be a graph with n vertices. Given a parameter $r < n$, an *r-division* of G is a decomposition of G into $\Theta(n/r)$ regions, each with at most r vertices and $O(\sqrt{r})$ boundary vertices. The bounds on the number of regions and on the number of vertices in each region imply that an r-division is a balanced decomposition of G. The $O(\sqrt{r})$ bound on the number of boundary vertices immediately implies the same bound for the overlap between different regions in the decomposition. For an increasing sequence $\mathbf{r} = r_1, r_2, \ldots$, a *recursive* $\mathbf{r}$*-division* is a decomposition tree $\mathcal{T}$ in which the nodes at height i form an r_i-division.

For various applications it is useful to impose additional requirements, such as requiring regions to be connected, requiring that each region share vertices with a constant number of other regions, etc. One particularly useful requirement that is relevant to planar graphs is that the boundary vertices of each region lie on a small number of faces. Formally, every region R inherits its embedding from that of the planar graph G. A hole of R is a face of R that is not a face of G. An *r-division with few holes* is an r-division in which each region has a constant number of holes.

Key Results

Balanced graph decompositions with small overlap are based on small balanced separators. An n-vertex graph G has a $f(n)$-separator if there exists a partition A, B, S of the vertices of G, such that the size of S is at most $f(n)$, the sizes of A and B are at most $2n/3$, and no edge exists between A and B. The set S is called a *separator*. The subgraphs induced on $A \cup S$ and $B \cup S$ form a balanced decomposition of G into 2 regions with $f(n)$ boundary vertices.

The best-known separator result for planar graphs is the $O(\sqrt{n})$ vertex separator of Lipton and Tarjan [16]. Consider a breadth-first-search tree T of a planar graph G. Each BFS level (i.e., the set of vertices at a specific distance from the root) is a separator of G, albeit not necessarily a small or a balanced one. Lipton and Tarjan's separator is based on the observation that it is possible to construct an $O(\sqrt{n})$ balanced separator by combining two appropriately chosen BFS levels with a fundamental cycle with respect to the BFS tree T.

Theorem 1 (Lipton-Tarjan separator) *Let G be an n-vertex planar graph, equipped with non-negative vertex weights summing to one. There exists a linear-time algorithm that returns a separation A, B, S of G such that S consists of at*

most $2\sqrt{2n}$ *vertices, and neither* A *nor* B *has total weight exceeding* $2/3$.

Note that the formulation of the theorem allows for a balanced separation with respect to a general weight function, rather than just with respect to the number of vertices.

Miller gave an $O(\sqrt{n})$ simple cycle separator [19] for planar graphs. Miller's result can be viewed as a version of Lipton and Tarjan's separator applied to the planar dual of G.

Theorem 2 (Miller's cycle separator) *Let* G *be an* n*-vertex 2-connected planar graph, equipped with nonnegative face weights summing to 1, such that no face weighs more than 2/3. Let* d *denote the maximum over all face sizes in* G*. There exists a linear-time algorithm that returns a simple cycle* C *with at most* $2\sqrt{2\lfloor d/2 \rfloor n}$ *vertices, such that neither the interior of* C *nor the exterior of* C *has total weight exceeding* $2/3$.

Similar formulations exist for vertex and edge weights.

$O(\sqrt{n})$ separators are known for other families of sparse graphs, such as graphs excluding a fixed minor [1, 11]. However, some sparse graphs (e.g., expanders) do not have small separators.

By applying a separator recursively, Frederickson [6] showed that r-divisions exist for graphs with $O(\sqrt{n})$ separators. Frederickson's construction generates a decomposition tree whose leaves correspond to the regions of an r-division. It consists of two phases. In the first phase, the separator theorem is applied to each region consisting of more than r vertices. This results in $\Theta(n/r)$ regions, each with at most r vertices and $\sqrt{r}$ boundary vertices *on average*. In a second phase, the separator theorem is applied to each region with more than $\sqrt{r}$ boundary vertices, assigning weight only to boundary vertices. Frederickson proves that this two-phase process results in an r-division. A naïve implementation of Frederickson's approach to construct an r-division takes $O(n \log n)$ time. By applying this approach to a contracted graph, and then further subdividing some of the resulting regions, an r-division can be constructed in $O(n \log r + (n/\sqrt{r}) \log n)$ time [6].

Goodrich [7] showed that for planar graphs an entire binary decomposition tree whose leaves correspond to regions with a constant number of vertices can be computed in $O(n)$ time. This is achieved by showing that, after linear time preprocessing, each invocation of Lipton and Tarjan's separator theorem can be implemented in sublinear time in the number of vertices of a region. The key components of Goodrich's algorithm are the use of a tree-cotree pair of spanning trees of G and of its planar dual to facilitate the search for balanced fundamental cycles, representing these trees using dynamic trees [22], and the use of balanced binary search trees to maintain BFS levels. At each recursive iteration a separator is found in a region with n' vertices in $O(\sqrt{n'} \log n')$ time. This leads to a total linear running time for computing a complete decomposition tree.

Subramanian and Klein [12] were the first to suggest r-divisions with few holes in planar graphs. The idea for achieving a constant number of holes is to use Miller's simple cycle separator instead of Lipton and Tarjan's. To keep the number of holes constant, one needs to alternate the separation criterion between balance with respect to the number of vertices and balance with respect to the number of holes [5]. Since a cycle separator introduces at most one new hole into each of the resulting two regions, reducing the number of holes by a constant factor every constant number of iterations ensures that the number of holes in each region is bounded by a (small) constant. Using Frederickson's approach, an r-division with few holes can be constructed in $O(n \log r + (n/\sqrt{r}) \log n)$ [9].

Klein, Mozes, and Sommer [14] presented a modified version of Miller's cycle separator and used it to obtain a linear-time algorithm for computing r-divisions with few holes in planar graphs (see also [2] for a similar result). Following Miller, their cycle separator algorithm uses BFS levels in the planar dual of G. They show a choice of a spanning tree T of G that makes their cycle separator algorithm very similar to Lipton and Tarjan's vertex separator. Using a technique similar to that of Goodrich, they use this cycle separator algorithm to generate an entire decom-

position tree of G in linear time. They show that alternating between three balance criteria (number of vertices, number of boundary vertices, and number of holes) results in a decomposition tree that contains an r-division for any value of r. This is in contrast with Frederickson's two-phase construction which targets a specific value of r. As a consequence, the resulting decomposition tree also contains a recursive **r**-division, for practically any choice of **r**.

Theorem 3 *There exists a linear-time algorithm that, given a planar graph G and an increasing sequence* **r***, computes a recursive* **r***-division with few holes of G.*

Applications

Separator-based decompositions are wildly used in divide-and-conquer algorithms. Lipton and Tarjan [17] used their separator theorem to show a variety of approximation algorithms and subexponential-time algorithms for NP-hard problems such as the maximum independent set, as well as $O(n^{3/2})$-time algorithms for problems such as maximum matching and Gaussian elimination [18]. These recursive algorithms implicitly generate a complete binary decomposition tree of the input graph. Typically, such algorithms only use the existence of small balanced separators and do not rely on planarity. Hence, they are applicable to families of graphs other than planar graphs.

Frederickson introduced r-divisions for computing shortest paths in a planar graph with nonnegative arc lengths in $O(n\sqrt{\log n})$ time and for finding a minimum st-cut or a maximum st-flow in undirected planar graphs in $O(n \log n)$ time. Since then r-divisions were used, along with Goodrich's linear-time construction, in many algorithms and in different settings (sequential, parallel, dynamic graph problems). A very partial list includes dynamic planar graph algorithms [4], Laplacian solvers and electrical flow algorithms [15, 20], and parallel algorithms in computational geometry [7]. Henzinger et al. [8] used a recursive **r**-division with roughly $\log^* n$ levels to compute shortest paths with nonnegative arc lengths in linear time.

Decompositions based on simple cycle separators are also wildly used in efficient algorithms for planar graphs. Examples include maximum flow [3, 10], shortest paths [13], and many others. These algorithms typically rely on additional structural properties specific to planar graphs, such as non-crossing of shortest paths (also known as the Monge property). Decompositions with few holes were introduced by Klein and Subramanian [12] to construct approximate dynamic distance oracles for planar graphs. Fakcharoenphol and Rao [5] used a complete decomposition with few holes for computing shortest paths with negative lengths in planar graphs in $O(n \log^3 n)$-time. The currently fastest algorithm for this problem uses r-divisions with few holes and runs in $O(n \log^2 n / \log\log n)$ time [21]. Italiano et al. [9] used an r-division with few holes to find a min st-cut and max st-flow in $O(n \log\log n)$ time.

Cross-References

- ▸ Fully Dynamic Higher Connectivity for Planar Graphs
- ▸ Separators in Graphs
- ▸ Shortest Paths in Planar Graphs with Negative Weight Edges

Recommended Reading

1. Alon N, Seymour PD, Thomas R (1990) A separator theorem for graphs with an excluded minor and its applications. In: Proceedings of the 22nd annual ACM symposium on theory of computing (STOC), Baltimore, pp 293–299
2. Arge L, van Walderveen F, Zeh N (2013) Multiway simple cycle separators and I/O-efficient algorithms for planar graphs. In: Proceedings of the 24th annual ACM-SIAM symposium on discrete algorithms (SODA), New Orleans, pp 901–918
3. Borradaile G, Klein PN, Mozes S, Nussbaum Y, Wulff-Nilsen C (2011) Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. In: Proceedings of the 52nd annual symposium on foundations of computer science (FOCS), Palm Springs, pp 170–179

4. Eppstein D, Galil Z, Italiano GF, Spencer TH (1993) Separator based sparsification for dynamic planar graph algorithms. In: Proceedings of the 25th symposium theory of computing, San Diego. ACM, pp 208–217. http://www.acm.org/pubs/citations/proceedings/stoc/167088/p208-eppstein/
5. Fakcharoenphol J, Rao S (2006) Planar graphs, negative weight edges, shortest paths, and near linear time. J Comput Syst Sci 72(5):868–889. http://dx.doi.org/10.1016/j.jcss.2005.05.007, preliminary version in FOCS 2001
6. Frederickson GN (1987) Fast algorithms for shortest paths in planar graphs with applications. SIAM J Comput 16:1004–1022
7. Goodrich MT (1995) Planar separators and parallel polygon triangulation. J Comput Syst Sci 51(3):374–389
8. Henzinger MR, Klein PN, Rao S, Subramanian S (1997) Faster shortest-path algorithms for planar graphs. J Comput Syst Sci 55(1):3–23. doi:10.1145/195058.195092
9. Italiano GF, Nussbaum Y, Sankowski P, Wulff-Nilsen C (2011) Improved algorithms for min cut and max flow in undirected planar graphs. In: Proceedings of the 43rd annual ACM symposium on theory of computing (STOC). ACM, New York, pp 313–322. http://doi.acm.org/10.1145/1993636.1993679,http://doi.acm.org/10.1145/1993636.1993679
10. Johnson DB, Venkatesan S (1982) Using divide and conquer to find flows in directed planar networks in $O(n^{3/2}\log n)$ time. In: Proceedings of the 20th annual allerton conference on communication, control, and computing, Monticello, pp 898–905
11. Kawarabayashi K, Reed BA (2010) A separator theorem in minor-closed classes. In: 51th annual IEEE symposium on foundations of computer science (FOCS), Las Vegas, pp 153–162
12. Klein PN, Subramanian S (1998) A fully dynamic approximation scheme for shortest paths in planar graphs. Algorithmica 22(3):235–249
13. Klein PN, Mozes S, Weimann O (2010) Shortest paths in directed planar graphs with negative lengths: a linear-space $O(n\log^2 n)$-time algorithm. ACM Trans Algorithms 6(2):1–18. http://doi.acm.org/10.1145/1721837.1721846, preliminary version in SODA 2009
14. Klein PN, Mozes S, Sommer C (2013) Structured recursive separator decompositions for planar graphs in linear time. In: Symposium on theory of computing conference (STOC), Palo Alto, pp 505–514
15. Koutis I, Miller GL (2007) A linear work, o(n1/6) time, parallel algorithm for solving planar laplacians. In: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms, society for industrial and applied mathematics (SODA '07), Philadelphia,pp 1002–1011. http://dl.acm.org/citation.cfm?id=1283383.1283491
16. Lipton RJ, Tarjan RE (1979) A separator theorem for planar graphs. SIAM J Appl Math 36(2):177–189
17. Lipton RJ, Tarjan RE (1980) Applications of a planar separator theorem. SIAM J Comput 9(3):615–627
18. Lipton RJ, Rose DJ, Tarjan RE (1979) Generalized nested dissection. SIAM J Numer Anal 16:346–358
19. Miller GL (1986) Finding small simple cycle separators for 2-connected planar graphs.J Comput Syst Sci 32(3):265–279. doi:10.1016/0022-0000(86)90030-9
20. Miller GL, Peng R (2013) Approximate maximum flow on separable undirected graphs. In: Proceedings of the twenty-fourth annual ACM-SIAM symposium on discrete algorithms (SODA), New Orleans, pp 1151–1170
21. Mozes S, Wulff-Nilsen C (2010) Shortest paths in planar graphs with real lengths in $O(n\log^2 n/\log\log n)$ time. In: Proceedings of the 18th European symposium on algorithms (ESA), Liverpool, pp 206–217
22. Sleator D, Tarjan R (1983) A data structure for dynamic trees. J Comput Syst Sci 26(3):362–391. doi:10.1016/0022-0000(83)90006-5

Reducing Bayesian Mechanism Design to Algorithm Design

Yang Cai[1], Constantinos Daskalakis[2], and Matthew Weinberg[3]
[1]Computer Science, McGill University, Montreal, QC, Canada
[2]EECS, Massachusetts Institute of Technology, Cambridge, MA, USA
[3]Computer Science, Princeton University, Princeton, NJ, USA

Keywords

Equivalence of separation and optimization; Fair allocation; Job scheduling; Mechanism design; Revenue maximization

Years and Authors of Summarized Original Work

STOC2012; Cai, Daskalakis, Weinberg
FOCS2012; Cai, Daskalakis, Weinberg
SODA2013; Cai, Daskalakis, Weinberg
FOCS2013; Cai, Daskalakis, Weinberg
SODA2015; Daskalakis, Weinberg

Problem Definition

The goal is to design algorithms that succeed in models where input is reported by strategic agents (henceforth referred to as *strategic input*), as opposed to standard models where the input is directly given (henceforth referred to as *honest input*). For example, consider a resource allocation problem where a single user has m jobs to process on n self-interested machines. Each machine i can process job j in time t_{ij}, and this is privately known only to the machine. Each machine reports some processing times $\hat{t}_{ij}$ to the user, who then runs some algorithm to determine where to process the jobs. Good approximation algorithms are known when machines are honest (i.e., $\hat{t}_{ij} = t_{ij}$ for all i, j) if the user's goal is to minimize the *makespan*, the time elapsed until all jobs are completed, going back to seminal work of Lenstra, Shmoys, and Tardos [13]. However, such algorithms do not account for the strategic nature of the machines, which may want to minimize their own work: why would they report honestly their processing time for each job if they can elicit a more favorable schedule by lying? To accommodate such challenges, new algorithmic tools must be developed that draw inspiration from Game Theory.

Requiring solutions that are robust against potential strategic manipulation potentially increases the computational difficulty of whatever problem is at hand. The discussed works provide a framework with which to design such solutions (henceforth called *mechanisms*) and address the following important question.

Question 1 How much (computationally) more difficult is mechanism design than algorithm design?

Using this framework, we resolve this question with an answer of "not at all" for several important problems including job scheduling and fair allocation. Another application of our framework provides efficient algorithms and structural characterization results for multi-item revenue-optimal auction design, a central open problem in mathematical economics.

Model

Environment

1. Set $\mathcal{F}$ of feasible outcomes. Interpret $\mathcal{F}$ as the set of all (feasible) allocations of jobs to machines, allocations of items to bidders, etc.
2. n agents who all care about which outcome is chosen.

Strategic Agents

1. Each agent i has a value $t_i(x)$ for each outcome $x \in \mathcal{F}$. t_i induces a function from $\mathcal{F} \to \mathbb{R}$ and is called the agent's *type*.
2. Each t_i is drawn *independently* from some distribution $\mathcal{D}_i$ of finite support.
3. Agent i knows t_i; all other agents and the designer know only $\mathcal{D}_i$.
4. Agents are *quasi-linear* and *risk neutral*. That is, the utility of an agent of type t for a randomized outcome (distribution over outcomes) $X \in \Delta(\mathcal{F})$, when he is charged price p, is $\mathbb{E}_{x \leftarrow X}[t(x)] - p$.
5. Agents behave in a way that maximizes utility, taking into consideration beliefs about the behavior of other agents.

Designer

1. Designs an *allocation rule* A and *price rule* P. A takes as input a type profile $(t_1, \ldots, t_n)$ and outputs (possibly randomly) an outcome $A(\mathbf{t}) \in \mathcal{F}$. P takes as input a type profile and outputs (possibly randomly) a price vector $P(\mathbf{t})$. The pair (A, P) is called a (direct) *mechanism*. Note that it is without loss of generality to consider only the design of direct mechanisms by the revelation principle [14].
2. Announces A and P to agents. Invites agents to report a type. When $\mathbf{t}$ is reported, selects the outcome $A(\mathbf{t})$ and charges agent i price $P_i(\mathbf{t})$.
3. Has some objective function $\mathcal{O}$ to optimize. $\mathcal{O}$ may depend on the agents' types, the outcome selected, and the prices charged, so we write $\mathcal{O}(\mathbf{t}, x, \mathbf{P})$. Examples include:
   - Social welfare: $\mathcal{O}(\mathbf{t}, x, \mathbf{P}) = \sum_i t_i(x)$.
   - Revenue: $\mathcal{O}(\mathbf{t}, x, \mathbf{P}) = \sum_i P_i(\mathbf{t})$.
   - Makespan: $\mathcal{O}(\mathbf{t}, x, \mathbf{P}) = \max_i \{-t_i(x)\}$ (In job scheduling, agents' values from alloca-

tions are nonpositive, since they have cost for processing jobs. An agent's cost for allocation x is then $-t_i(x)$.).

- Fairness: $\mathcal{O}(\mathbf{t}, x, \mathbf{P}) = \min_i \{t_i(x)\}$.

Game Theoretic Definitions

1. The *interim rule* of a mechanism is a function that takes as input an agent i and type t_i and outputs the distribution of allocations and prices that agent i sees when reporting type t_i over the randomness of the mechanism and the other agents' types, assuming they tell the truth. So the interim allocation rule $(\boldsymbol{\pi}, \mathbf{p})$ of the mechanism (A, P) satisfies:

$$\Pr[x \leftarrow \pi_i(t_i)] = \mathbb{E}_{\mathbf{t}_{-i} \leftarrow \mathcal{D}_{-i}} [\Pr[A(t_i; \mathbf{t}_{-i}) = x]] .$$

$$\Pr[p \leftarrow p_i(t_i)] = \mathbb{E}_{\mathbf{t}_{-i} \leftarrow \mathcal{D}_{-i}} [\Pr[P_i(t_i; \mathbf{t}_{-i}) = p]].$$

2. A mechanism is *Bayesian Incentive Compatible (BIC)* if every agent receives at least as much utility by reporting their true type as any other type (assuming other agents report truthfully). Formally, $t_i(\pi_i(t_i)) - p_i(t_i) \geq t_i(\pi_i(t_i')) - p_i(t_i')$ for all i, t_i, t_i' (We use the shorthand $t_i(\pi_i(t_i'))$ to denote the expected value of t_i for the random allocation drawn from $\pi_i(t_i')$. Formally, $t_i(\pi_i(t_i')) = \mathbb{E}_{x \leftarrow \pi_i(t_i')}[t_i(x)]$.). A commonly used relaxation of BIC is called *ϵ-Bayesian Incentive Compatible (ϵ-BIC)*. A mechanism is ϵ-BIC if every agent derives at most ϵ less utility by reporting their true type comparing to any other type (assuming other agents report truthfully). Formally, $t_i(\pi_i(t_i)) - p_i(t_i) \geq t_i(\pi_i(t_i')) - p_i(t_i') - \epsilon$ for all i, t_i, t_i'.
3. A mechanism is *individually rational (IR)* if every agent has nonnegative expected utility by participating in the mechanism (assuming other agents report truthfully). Formally, $t_i(\pi_i(t_i)) - p_i(t_i) \geq 0$ for all i, t_i.

Bayesian Mechanism Design (BMeD)

Here we describe formally the mechanism design problem we study. BMeD is parameterized by a set of feasible outcomes $\mathcal{F}$, objective function $\mathcal{O}$, and set of possible types $\mathcal{V}$. Both $\mathcal{V}$ and $\mathcal{F}$ can be discrete or continuous. We assume that every element $v \in \mathcal{V}$ and $x \in \mathcal{F}$ can be represented by a finite bit string $\langle v \rangle$ and $\langle x \rangle$. $\mathcal{V}$ and $\mathcal{F}$ also specify how those bit strings are interpreted. For instance, $\mathcal{V}$ might be the class of all submodular functions, and the bit strings used to represent them may be interpreted as indexing a black-box value oracle. Or $\mathcal{V}$ might be the class of all subadditive functions, and the bit strings used to represent them may be interpreted as an explicit circuit. Or $\mathcal{V}$ could be the class of all additive functions, and the bit strings used to represent them may be interpreted as a vector containing values for each item. So we are parameterizing our problems both by the actual classes $\mathcal{V}$ and $\mathcal{F}$ but also by how elements of these classes are represented. Now, we are ready to formally discuss the problem BMeD$(\mathcal{F}, \mathcal{V}, \mathcal{O})$.

BMeD$(\mathcal{F}, \mathcal{V}, \mathcal{O})$:

INPUT*: For each agent $i \in [n]$, a discrete distribution $\mathcal{D}_i$ over types in $\mathcal{V}$, described* explicitly *by listing the support of $\mathcal{D}_i$ and the corresponding probabilities.*

OUTPUT*: A BIC, IR mechanism.*

GOAL*: Find the mechanism that optimizes $\mathcal{O}$ in expectation, with respect to all BIC, IR mechanisms (when n bidders with types drawn from $\times_i \mathcal{D}_i$ report truthfully).*

APPROXIMATION*: A mechanism is said to be an (ϵ, α)-approximation to BMeD if it outputs an ϵ-BIC mechanism whose expected value of $\mathcal{O}$ (when n bidders with types drawn from $\times_i \mathcal{D}_i$ report truthfully) is at least $\alpha\mathrm{OPT} - \epsilon$ (or at most $\alpha\mathrm{OPT} + \epsilon$ for minimization problems).*

Generalized Objective Optimization Problem (GOOP)

Here we describe formally the algorithmic problem we show has strong connections to BMeD. GOOP is parameterized by a set of feasible outcomes $\mathcal{F}$, objective function $\mathcal{O}$, and set of possible types $\mathcal{V}$. We therefore formally discuss the problem GOOP$(\mathcal{F}, \mathcal{V}, \mathcal{O})$. Below, $\mathcal{V}^{\times}$ denotes the closure of $\mathcal{V}$ under linear combinations. Functions in $\mathcal{V}^{\times}$ are represented by a finite list of elements of $\mathcal{V}$, along with (possibly negative) scalar multipliers.

GOOP$(\mathcal{F}, \mathcal{V}, \mathcal{O})$:

INPUT: *For each agent* $i \in [n]$*, a type* $g_i \in \mathcal{V}$*, multiplier* $m_i \in \mathbb{R}$*, and cost function* $f_i \in \mathcal{V}^\times$*. Additionally, an indicator bit* b *(The indicator bit* b *is included so that the optimization of just* $\sum_i f_i(x)$ *(without price multipliers or* $\mathcal{O}$*) is formally a special case of* GOOP$(\mathcal{F}, \mathcal{V}, \mathcal{O})$*.).*

OUTPUT: *An allocation* $x \in \mathcal{F}$*, and price vector* $\mathbf{p} \in \mathbb{R}^n$*.*

GOAL: *Find* $\arg\max_{x \in \mathcal{F}, \mathbf{p}} \{b \cdot \mathcal{O}(\mathbf{g}, x, \mathbf{p}) + \sum_i m_i p_i + \sum_i f_i(x)\}$ *(or* $\arg\min$*, if* $\mathcal{O}$ *is a minimization objective like makespan).*

APPROXIMATION: $(x, \mathbf{p})$ *is said to be an* (α, β)*-approximation to GOOP if* $\beta \cdot b \cdot \mathcal{O}(\mathbf{g}, x, \mathbf{p}) + \sum_i m_i p_i + \sum_i f_i(x)$ *is at least/most* $\alpha \cdot OPT$*. Note that a* $(\alpha, 1)$*-approximation is the standard notion of an* α*-approximation. Allowing* $\beta \neq 1$ *boosts/discounts the value of* $\mathcal{O}$ *(the objective) before comparing to* $\alpha \cdot OPT$*. Note also that allowing* $\beta \neq 1$ *provides no benefit if* $b = 0$*.*

Key Results

We provide a poly-time black-box reduction from BMeD$(\mathcal{F}, \mathcal{V}, \mathcal{O})$ to GOOP$(\mathcal{F}, \mathcal{V}, \mathcal{O})$. That is, we provide a reduction from Bayesian mechanism design to traditional algorithm design.

Theorem 1 *Let* G *be an* (α, β)*-approximation algorithm for* GOOP$(\mathcal{F},\mathcal{V},\mathcal{O})$*. Then for all* $\epsilon > 0$*, there is an* $(\epsilon, \alpha/\beta)$*-approximation algorithm for* BMeD$(\mathcal{F},\mathcal{V},\mathcal{O})$*. If* ℓ *is the length of the input to a* BMeD$(\mathcal{F},\mathcal{V},\mathcal{O})$ *instance, the algorithm succeeds with probability* $1 - \exp(-\text{poly}(\ell, 1/\epsilon))$*, makes* $\text{poly}(\ell, 1/\epsilon)$ *black-box calls to* G *on inputs of size* $\text{poly}(\ell, 1/\epsilon)$*, and terminates in time* $\text{poly}(\ell, 1/\epsilon)$ *(times the running time of each oracle call to* G*).*

This reduction is developed in a recent series of papers by the authors [4–7, 9]. The possibility of failure and additive error is due to a sampling procedure in the reduction. In addition to the computational aspect provided in Theorem 1, our reduction also has a structural aspect. Namely, we provide a characterization of the optimal mechanism in Bayesian settings.

Theorem 2 *For all objectives* $\mathcal{O}$*, feasibility constraints* $\mathcal{F}$*, set of possible types* $\mathcal{V}$*, and inputs* $\mathcal{D}$ *to* BMeD$(\mathcal{F}, \mathcal{V}, \mathcal{O})$*, the optimal mechanism is a distribution over generalized objective maximizers. Formally, there exists a joint distribution* Δ *over an indicator bit* b^δ *and mappings* $(f_1^\delta, \ldots, f_n^\delta)$*, where each* f_i^δ *maps types* t_i *to multipliers* $m_i^\delta(t_i) \in \mathbb{R}$ *and cost functions* $\phi_i^\delta(t_i) \in \mathcal{V}^\times$*, such that the optimal mechanism first samples* $(b^\delta, \mathbf{f}^\delta)$ *from* Δ *then maps the type profile* $\mathbf{t}$ *to the allocation and price vector* $(x(\mathbf{t}), \mathbf{p}(\mathbf{t})) = \arg\max_{x \in \mathcal{F}, \mathbf{p}} \{b^\delta \cdot \mathcal{O}(\mathbf{t}, x, \mathbf{p}) + \sum_i m_i^\delta(t_i) p_i + \sum_i \phi_i^\delta(t_i)(x)\}$.

Perhaps the most interesting case of Theorem 2 is when the objective is revenue. In this case, we may interpret the cost functions $\phi_i^\delta \in \mathcal{V}^\times$ as the *virtual valuation function* of bidder i. By virtual valuations, we do *not* mean Myerson's specific virtual valuation functions [14], which aren't even defined for multi-item instances. Instead we simply mean *some* virtual valuation functions that may or may not be the same as the types/valuations reported by the agents. We include this and other applications of Theorems 1 and 2 below.

Applications

In this section, we apply Theorem 1 to the objectives of revenue, makespan, and fairness.

Revenue Maximization

We apply Theorem 1 to reduce the BMeD problem of optimizing revenue in multi-item settings to GOOP. In [7], it is shown that for this case, one need only consider instances of GOOP with $b = m_1 = \ldots = m_n = 0$, so the GOOP instances that must be solved require just optimization of the cost function (which we call *virtual welfare* for this application). We obtain the following computational and structural results on optimal auction design in general multi-item settings, addressing a long-standing open question following Myerson's seminal work on single-item auctions [14].

Theorem 3 (Revenue Maximization, Computational) *Let G be an α-approximation algorithm for maximizing virtual welfare over $\mathcal{F}$ when all virtual types are from $\mathcal{V}^{\times}$. Then for all $\epsilon > 0$, there is an (ϵ, α)-approximation algorithm for the problem* BMeD($\mathcal{F}, \mathcal{V}$, REVENUE) *that makes polynomially many black-box calls to G. If ℓ is the length of the input to a* BMeD($\mathcal{F}$,$\mathcal{V}$,REVENUE) *instance, the algorithm succeeds with probability $1 - \exp(-\text{poly}(\ell, 1/\epsilon))$, makes $\text{poly}(\ell, 1/\epsilon)$ black-box calls to G on inputs of size $\text{poly}(\ell, 1/\epsilon)$, and terminates in time $\text{poly}(\ell, 1/\epsilon)$ (times the running time of each oracle call to G).*

Theorem 4 (Revenue Maximization, Structural) *In any multi-item setting with arbitrary feasibility constraints and possible agent types, the allocation rule of the revenue-optimal auction is a distribution over virtual welfare maximizers. Formally, there exists a distribution Δ over mappings $(\phi_1, \ldots, \phi_n)$, where each ϕ_i maps types t_i to cost functions $f_i \in \mathcal{V}^{\times}$, such that the allocation rule for the optimal mechanism first samples $\boldsymbol{\phi}$ from Δ then maps type profile $\mathbf{t}$ to the allocation* $\arg\max_{x \in \mathcal{F}} \{\sum_i \phi_i(t_i)(x)\}$.

We further consider the following important special case: There are m items for sale to n buyers. Any allocation of items to buyers is feasible (that is, each item can be awarded to at most one buyer), so we can denote the set of feasible allocations as $\mathcal{F} = [n+1]^m$. Furthermore, each buyer i has a value v_{ij} for item j and is *additive* across items, meaning that their value for a set S of items is $\sum_{j \in S} v_{ij}$. So we can denote the set of possible types as $\mathbb{R}_+^m$ (and have types represented as such).

Theorem 5 (Revenue Maximization for Additive Buyers, Computational) *There is a poly-time algorithm for* GOOP($[n+1]^m$, $\mathbb{R}_+^m$, REVENUE). *Therefore, there is a poly-time algorithm for* BMeD($[n+1]^m, \mathbb{R}_+^m$, REVENUE) *(In this special case, no sampling is required in the reduction, so the theorem holds even for $\epsilon = 0$. Formally, this is a $(0, 1)$-approximation (an exact algorithm). See [4] for details.).*

Theorem 6 (Revenue Maximization for Additive Buyers, Structural) *In any multi-item setting with n additive buyers and m items for sale, the allocation rule of the revenue-optimal auction is a distribution over virtual welfare maximizers. Formally, there exists a distribution Δ over mappings $(\phi_1, \ldots, \phi_n)$, where each ϕ_i maps types t_i to cost functions $f_i \in \mathbb{R}^m$, such that the allocation rule for the optimal mechanism first samples $\boldsymbol{\phi}$ from Δ then awards every item j to a buyer in* $\arg\max_i \{\phi_{ij}(\mathbf{v}_i)\}$ *if their virtual value for item j is nonnegative and does not allocate the item otherwise.*

Job Scheduling on Unrelated Machines

The problem of job scheduling on unrelated machines consists of m jobs and n machines, with machine i able to process job j in time t_{ij}. The goal is to find a schedule (that assigns each job to exactly one machine) minimizing the *makespan*. Specifically, if S_i are the jobs assigned to machine i, the makespan is $\max_i \{\sum_{j \in S_i} t_{ij}\}$. As a mechanism design problem, one considers the machines to be strategic agents who know their processing time for each job (but the designer and other machines do not). In the language of BMeD, we can denote the feasibility constraints as $[n]^m$, the set of possible types as $\mathbb{R}_+^m$, and the objective as MAKESPAN. Theorem 1 reduces BMeD($[n]^m, \mathbb{R}_+^m$, MAKESPAN) to GOOP($[n]^m, \mathbb{R}_+^m$, MAKESPAN). It is shown in [7] that for objectives that don't depend on the prices charged at all (called "allocation-only"), only instances of GOOP with $m_i = 0 \ \forall i$ need be considered. It is further shown in [9] that GOOP($[n]^m, \mathbb{R}_+^m$, MAKESPAN) can be interpreted as a job scheduling problem with costs. Specifically, GOOP($[n]^m, \mathbb{R}_+^m$, MAKESPAN) takes as input a processing time $t_{ij} \geq 0$, and monetary cost $c_{ij} \in \mathbb{R}$ for all machines i and jobs j. The goal is to find a schedule that minimizes the makespan plus cost. Formally, partition the jobs into disjoint sets S_i to minimize $\max_i \{\sum_{j \in S_i} t_{ij}\} + \sum_i \sum_j c_{ij}$. While it is NP-hard to approximate GOOP($[n]^m, \mathbb{R}_+^m$, MAKESPAN) within any finite factor, a result of Shmoys and Tardos from the early 1990s obtains a polynomial time $(1, 1/2)$-

approximation algorithm [15]. In combination with Theorem 1, this yields the following theorem:

Theorem 7 (Job Scheduling on Unrelated Machines) *For all $\epsilon > 0$, there is a poly-time $(\epsilon, 2)$-approximation algorithm for* BMeD($[n]^m, \mathbb{R}_+^m$, MAKESPAN)*. If ℓ is the length of the input to a* BMeD($[n]^m, \mathbb{R}_+^m$, MAKESPAN) *instance, the algorithm succeeds with probability $1 - \exp(-\text{poly}(\ell, 1/\epsilon))$ and terminates in time $\text{poly}(\ell, 1/\epsilon)$.*

Fair Allocation of Indivisible Goods

The problem of fairly allocating indivisible goods consists of m indivisible goods and n children, with child i receiving value v_{ij} for good j. The goal is to find an allocation of goods (that assigns each good to at most one child) maximizing the *fairness*. Specifically, if S_i are the goods allocated to child i, the fairness is $\min_i\{\sum_{j\in S_i} v_{ij}\}$. As a mechanism design problem, one considers the children to be strategic agents who know their own value for each good (but the designer and other children do not). In the language of BMeD, we can denote the feasibility constraints as $[n+1]^m$, the set of possible types as $\mathbb{R}_+^m$, and the objective as FAIRNESS. Theorem 1 reduces BMeD($[n+1]^m, \mathbb{R}_+^m$, FAIRNESS) to GOOP($[n+1]^m, \mathbb{R}_+^m$, FAIRNESS), which can be interpreted as a fair allocation problem with costs (again, because FAIRNESS is allocation only) [7,9]. Specifically, GOOP($[n+1]^m, \mathbb{R}_+^m$, FAIRNESS) takes as input a value $v_{ij} \geq 0$ and monetary cost $c_{ij} \in \mathbb{R}$ for all children i and goods j. The goal is to find an allocation that maximizes the fairness minus cost. Formally, allocate the goods into disjoint sets S_i to maximize $\min_i\{\sum_{j\in S_i} v_{ij}\} - \sum_i \sum_j c_{ij}$. While it is NP-hard to approximate GOOP($[n+1]^m, \mathbb{R}_+^m$, FAIRNESS) within any finite factor, we develop poly-time $(1, m-n+1)$- and $(1/2, \tilde{O}(\sqrt{n}))$-approximation algorithms for fair allocation with costs, based on algorithms of Bezáková and Dani [2] and Asadpour and Saberi [1] for fair allocation (without costs).

Theorem 8 (Fair Allocation of Indivisible Goods) *There are poly-time $(1, m-n+1)$- and $(1/2, \tilde{O}(\sqrt{n}))$-approximation algorithms for* GOOP($[n+1]^m, \mathbb{R}_+^m$, FAIRNESS)*. Therefore, for all $\epsilon > 0$, there is a $(\epsilon, \min\{\tilde{O}(\sqrt{n}), m-n+1\})$-approximation algorithm for* BMeD($[n+1]^m, \mathbb{R}_+^m$, FAIRNESS)*. If ℓ is the length of the input to a* BMeD($[n+1]^m, \mathbb{R}_+^m$, FAIRNESS) *instance, the algorithm succeeds with probability $1 - \exp(-\text{poly}(\ell, 1/\epsilon))$ and terminates in time $\text{poly}(\ell, 1/\epsilon)$.*

Tools for Convex Optimization

We prove Theorems 1 and 2 by solving a linear program over the space of possible interim allocation rules and generalizations of interim allocation rules that we do not discuss here. In doing so, we also develop new tools applicable for general convex optimization that we discuss here. We omit full details of the approach and refer the reader to a series of papers by the authors [5–7, 9] for specifics of the linear program solved and why it addresses BMeD. Seminal works of Khachiyan [12], Grötschel, Lovász, and Schrijver [10], and Karp and Papadimitriou [11] study the problems of optimization and separation over a close, convex region $P \subseteq \mathbb{R}^d$ (Below, we denote by $\alpha P = \{\alpha \mathbf{x} | \mathbf{x} \in P\}$. Also, for simplicity of exposition, we only consider P that contain the origin, so that $\alpha P \subseteq P$ for all $\alpha \leq 1$, but our results extend to all closed, convex P. See [9] for our most general results.). Formally, these problems are:

Optimize(P):

INPUT*: A direction* $\mathbf{c} \in \mathbb{R}^d$.
OUTPUT*: A point* $\mathbf{x} \in P$.
GOAL*: Find* $\mathbf{x}^* \in \arg\max_{\mathbf{x}\in P}\{\mathbf{c}\cdot\mathbf{x}\}$.

Separate(P):

INPUT*: A point* $\mathbf{x} \in \mathbb{R}^d$.
OUTPUT*: "Yes," or a direction* $\mathbf{c} \in \mathbb{R}^d$.
GOAL*: If* $\mathbf{x} \in P$*, output "yes." Otherwise, output any* $\mathbf{c}$ *such that* $\mathbf{c}\cdot\mathbf{x} > \max_{\mathbf{y}\in P}\{\mathbf{c}\cdot\mathbf{y}\}$.

Khachiyan's Ellipsoid algorithm shows that if one can solve the problem Separate(P) in time $\text{poly}(d)$, then one can also solve Optimize(P) in

time poly(d). Grötschel, Lovász, and Schrijver and independently Karp and Papadimitriou show that the other direction holds as well: if one can solve Optimize(P) in time poly(d), then one can also solve Separate(P) in time poly(d). This is colloquially called "the equivalence of separation and optimization." While separation as a means for optimization has obvious uses, optimization as a means for separation is more subtle. Still, numerous applications exist (including our results) and we refer the reader to [10, 11] for several others, including the first poly-time algorithm for submodular minimization.

In order to provide our guarantees with respect to approximation, we develop further the equivalence of separation and optimization to accommodate approximation. Specifically, consider the following problems, further parameterized by some $\alpha < 1$:

α-Optimize(P):

INPUT: *A direction* $\mathbf{c} \in \mathbb{R}^d$.
OUTPUT: *A point* $\mathbf{x} \in P$.
GOAL: *Find* $\mathbf{x}$ *satisfying* $\mathbf{c} \cdot \mathbf{x} \geq \alpha \max_{\mathbf{y} \in P} \{\mathbf{c} \cdot \mathbf{y}\}$.

α-Separate(P):

INPUT: *A point* $\mathbf{x} \in \mathbb{R}^d$.
OUTPUT: *"Yes" and a proof that* $\mathbf{x} \in P$, *or a direction* $\mathbf{c} \in \mathbb{R}^d$ *(For formal details on exactly what constitutes a proof, we refer the reader to [6, 7, 9]. Roughly speaking,* $\mathbf{x}$ *is written as a convex combination of points* known *to be in* P*.).*
GOAL: *If* $\mathbf{x} \in \alpha P$, *output "yes" and a proof that* $\mathbf{x} \in P$. *If* $\mathbf{x} \notin P$, *output a direction* $\mathbf{c}$ *such that* $\mathbf{c} \cdot \mathbf{x} > \alpha \max_{\mathbf{y} \in P} \{\mathbf{c} \cdot \mathbf{y}\}$. *If* $\mathbf{x} \in P \backslash \alpha P$, *either is acceptable.*

Theorem 9 (Approximate Equivalence of Separation and Optimization) *For all* $\alpha \leq 1$, *the problems* α-Optimize(P) *and* α-Separate(P) *are computationally equivalent. That is, if one can solve one in time* poly(d), *one can solve the other in time* poly(d) *as well.*

We also extend these results to accommodate bi-criterion approximation, via the problems below, further parameterized by some $\beta > 1$ and subset $S \subseteq [d]$ of coordinates (Below, when we write $(\beta \mathbf{x}_S, \mathbf{x}_{-S})$, we mean to take $\mathbf{x}$ and multiply each $x_i, i \in S$ by β.).

(α, β, S)-Optimize(P):

INPUT: *A direction* $\mathbf{c} \in \mathbb{R}^d$.
OUTPUT: *A point* $\mathbf{x} \in P$.
GOAL: *Find* $\mathbf{x}$ *satisfying* $\mathbf{c} \cdot (\beta \mathbf{x}_S, \mathbf{x}_{-S}) \geq \alpha \max_{\mathbf{y} \in P} \{\mathbf{c} \cdot \mathbf{y}\}$.

(α, β, S)-Separate(P):

INPUT: *A point* $\mathbf{x} \in \mathbb{R}^d$.
OUTPUT: *"Yes" and a proof that* $\mathbf{x} \in P$, *or a direction* $\mathbf{c} \in \mathbb{R}^d$.
GOAL: *If* $(\beta \mathbf{x}_S, \mathbf{x}_{-S}) \in \alpha P$, *output "yes" and a proof that* $\mathbf{x} \in P$. *If* $\mathbf{x} \notin P$, *output a direction* $\mathbf{c}$ *such that* $\mathbf{c} \cdot (\beta \mathbf{x}_S, \mathbf{x}_{-S}) > \alpha \max_{\mathbf{y} \in P} \{\mathbf{c} \cdot \mathbf{y}\}$. *If* $(\beta \mathbf{x}_S, \mathbf{x}_{-S}) \notin \alpha P$ *and* $\mathbf{x} \in P$, *either is acceptable (An astute reader might worry that for some* α, β, S, P, *the problem* (α, β, S)*-Separate(*P*) is impossible, due to the existence of an* $\mathbf{x} \notin P$ *such that* $(\beta \mathbf{x}_S, \mathbf{x}_{-S}) \in \alpha P$. *For some* α, β, S, P, *this is indeed the case, but we show that* (α, β, S)*-Optimize(*P*) is impossible in these cases as well.).*

Theorem 10 (Bi-Criterion Approximate Equivalence of Separation and Optimization) *For all* $\alpha \leq 1, \beta \geq 1, S \subseteq [d]$, *the problems* (α, β, S)-Optimize(P) *and* (α, β, S)-Separate(P) *are computationally equivalent. That is, if one can solve one in time* poly(d), *one can solve the other in time* poly(d) *as well.*

More formal statements and how we apply these theorems to yield our main result can be found in [9]. Finally, the theorems hold for minimization as well as maximization and without the restriction that P contains the origin (but the theorem statements are more technical).

Open Problems

Our work provides a novel computational framework for solving Bayesian mechanism design problems. We have applied our framework to solve several specific important problems, such as computing revenue-optimal auctions in multi-item settings and approximately optimal BIC mechanisms for job scheduling, but numerous

R

important settings and objectives remain unresolved. Theorem 1 provides a concrete approach for tackling such problems, via the design of (α, β)-approximations for the purely algorithmic Generalized Objective Optimization Problem. Therefore, one important direction following our work is to apply our framework to novel settings and design algorithms for the resulting GOOP instances.

Recommended Reading

1. Asadpour A, Saberi A (2007) An approximation algorithm for max-min fair allocation of indivisible goods. In: The 39th annual ACM symposium on theory of computing (STOC), San Diego
2. Bezáková I, Dani V (2005) Allocating indivisible goods. SIGecom Exch 5(3):11–18
3. Cai Y, Daskalakis C (2011) Extreme-value theorems for optimal multidimensional pricing. In: The 52nd annual IEEE symposium on foundations of computer science (FOCS), Palm Springs
4. Cai Y, Daskalakis C, Matthew Weinberg S (2012) An algorithmic characterization of multi-dimensional mechanisms. In: The 44th annual ACM symposium on theory of computing (STOC), New York
5. Cai Y, Daskalakis C, Matthew Weinberg S (2012) Optimal multi-dimensional mechanism design: reducing revenue to welfare maximization. In: The 53rd annual IEEE symposium on foundations of computer science (FOCS), New Brunswick
6. Cai Y, Daskalakis C, Matthew Weinberg S (2013) Reducing revenue to welfare maximization: approximation algorithms and other generalizations. In: The 24th annual ACM-SIAM symposium on discrete algorithms (SODA), New Orleans
7. Cai Y, Daskalakis C, Matthew Weinberg S (2013) Understanding incentives: mechanism design becomes algorithm design. In: The 54th annual IEEE symposium on foundations of computer science (FOCS), Berkeley
8. Daskalakis C, Matthew Weinberg S (2012) Symmetries and optimal multi-dimensional mechanism design. In: The 13th ACM conference on electronic commerce (EC), Valencia
9. Daskalakis C, Matthew Weinberg S (2015) Bayesian truthful mechanisms for job scheduling from bi-criterion approximation algorithms. In: The 26th annual ACM-SIAM symposium on discrete algorithms (SODA), San Diego
10. Grötschel M, Lovász L, Schrijver A (1981) The ellipsoid method and its consequences in combinatorial optimization. Combinatorica 1(2):169–197
11. Karp RM, Papadimitriou CH (1980) On linear characterizations of combinatorial optimization problems. In: The 21st annual symposium on foundations of computer science (FOCS), Syracuse
12. Khachiyan LG (1979) A polynomial algorithm in linear programming. Sov Math Dokl 20(1):191–194
13. Lenstra JK, Shmoys DB, Tardos É (1990) Approximation algorithms for scheduling unrelated parallel machines. Math Program 46(1–3):259–271
14. Myerson RB (1981) Optimal auction design. Math Oper Res 6(1):58–73
15. Shmoys DB, Tardos É (1993) Scheduling unrelated machines with costs. In: The 4th symposium on discrete algorithms (SODA), Austin

Registers

Paul Vitányi
Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands

Keywords

Asynchronous communication hardware; Shared-memory (wait-free); Wait-free registers; Wait-free shared variables

Years and Authors of Summarized Original Work

1986; Lamport, Vitanyi, Awerbuch

Problem Definition

Consider a system of asynchronous processes that communicate among themselves by only executing read and write operations on a set of shared variables (also known as shared *registers*). The system has no global clock or other synchronization primitives. Every shared variable is associated with a process (called *owner*) which writes it and the other processes may read it. An execution of a write (read) operation on a shared variable will be referred to as a *Write* (*Read*) on that variable. A Write on a shared variable puts a value from a pre-determined finite domain into the variable, and a Read reports a value from the

domain. A process that writes (reads) a variable is called a *writer* (*reader*) of the variable.

The goal is to construct shared variables in which the following two properties hold. (1) Operation executions are not necessarily atomic, that is, they are not indivisible but rather consist of atomic sub-operations, and (2) every operation finishes its execution within a bounded number of its own steps, irrespective of the presence of other operation executions and their relative speeds. That is, operation executions are *wait-free*. These two properties give rise to a classification of shared variables, depending on their output characteristics. Lamport [8] distinguishes three categories for 1-writer shared variables, using a precedence relation on operation executions defined as follows: for operation executions A and B, A *precedes* B, denoted $A \longrightarrow B$, if A finishes before B starts; A and B *overlap* if neither A precedes B nor B precedes A. In 1-writer variables, all the Writes are totally ordered by "$\longrightarrow$". The three categories of 1-writer shared variables defined by Lamport are the following.

1. A *safe* variable is one in which a Read not overlapping any Write returns the most recently written value. A Read that overlaps a Write may return any value from the domain of the variable.
2. A *regular* variable is a safe variable in which a Read that overlaps one or more Writes returns either the value of the most recent Write preceding the Read or of one of the overlapping Writes.
3. An *atomic* variable is a regular variable in which the Reads and Writes behave as if they occur in some total order which is an extension of the precedence relation.

A shared variable is *boolean* (Boolean variables are referred to as *bits*.) or *multivalued* depending upon whether it can hold only one out of two or one out of more than two values. A *multiwriter* shared variable is one that can be written and read (concurrently) by many processes. If there is only one writer and more than one reader it is called a *multireader* variable.

Key Results

In a series of papers starting in 1974, for details see [4], Lamport explored various notions of concurrent reading and writing of shared variables culminating in the seminal 1986 paper [8]. It formulates the notion of wait-free implementation of an atomic multivalued shared variable – written by a single writer and read by (another) single reader – from safe 1-writer 1-reader 2-valued shared variables, being mathematical versions of physical *flip-flops*, later optimized in [13]. Lamport did not consider constructions of shared variables with more than one writer or reader.

Predating the Lamport paper, in 1983 Peterson [10] published an ingenious wait-free construction of an atomic 1-writer, n-reader m-valued atomic shared variable from $n + 2$ safe 1-writer n-reader m-valued registers, $2n$ 1-writer 1-reader 2-valued atomic shared variables, and 2 1-writer n-reader 2-valued atomic shared variables. He presented also a proper notion of the wait-freedom property. In his paper, Peterson didn't tell how to construct the n-reader boolean atomic variables from flip-flops, while Lamport mentioned the open problem of doing so, and, incidentally, uses a version of Peterson's construction to bridge the algorithmically demanding step from atomic shared bits to atomic shared multivalues. On the basis of this work, N. Lynch, motivated by concurrency control of multi-user data-bases, posed around 1985 the question of how to construct wait-free multiwriter atomic variables from 1-writer multireader atomic variables. Her student Bloom [1] found in 1985 an elegant 2-writer construction, which, however, has resisted generalization to multiwriter. Vitányi and Awerbuch [14] were the first to define and explore the complicated notion of wait-free constructions of general multiwriter atomic variables, in 1986. They presented a proof method, an unbounded solution from 1-writer 1-reader atomic variables, and a bounded solution from 1-writer n-reader atomic variables. The bounded solution turned out not to be atomic, but only achieved regularity ("Errata" in [14]). The paper introduced important notions and techniques

in the area, like (bounded) vector clocks, and identified open problems like the construction of atomic wait-free bounded multireader shared variables from flip-flops, and atomic wait-free bounded multiwriter shared variables from the multireader ones. Peterson who had been working on the multiwriter problem for a decade, together with Burns, tried in 1987 to eliminate the error in the unbounded construction of [14] retaining the idea of vector clocks, but replacing the obsolete-information tracking technique by repeated scanning as in [10]. The result [11] was found to be erroneous in the technical report (R. Schaffer, On the correctness of atomic multiwriter registers, Report MIT/LCS/TM-364, 1988). Neither the re-correction in Schaffer's Technical Report, nor the claimed re-correction by the authors of [11] has appeared in print. Also in 1987 there appeared at least five purported solutions for the implementation of 1-writer n-reader atomic shared variable from 1-writer 1-reader ones: [2, 7, 12] (for the others see [4]) of which [2] was shown to be incorrect (S. Haldar, K. Vidyasankar, *ACM Oper. Syst. Rev*, 26:1(1992), 87–88) and only [12] appeared in journal version. The paper [9], initially a 1987 Harvard Tech Report, resolved all multiuser constructions in one stroke: it constructs a bounded n-writer n-reader (multiwriter) atomic variable from $O(n^2)$ 1-writer 1-reader safe bits, which is optimal, and $O(n^2)$ bit-accesses per Read/Write operation which is optimal as well. It works by making the unbounded solution of [14] bounded, using a new technique, achieving a robust proof of correctness. "Projections" of the construction give specialized constructions for the implementation of 1-writer n-reader (multireader) atomic variables from $O(n^2)$ 1-writer 1-reader ones using $O(n)$ bit accesses per Read/Write operation, and for the implementation of n-writer n-reader (multiwriter) atomic variables from n 1-writer n-reader (multireader) ones. The first "projection" is optimal, while the last "projection" may not be optimal since it uses $O(n)$ control bits per writer while only a lower bound of $\Omega(\log n)$ was established. Taking up this challenge, the construction in [6] claims to achieve this lower bound.

Timestamp System

In a multiwriter shared variable it is only required that every process keeps track of which process wrote last. There arises the general question whether every process can keep track of the order of the last Writes by all processes. A. Israeli and M. Li were attracted to the area by the work in [14], and, in an important paper [5], they raised and solved the question of the more general and universally useful notion of a bounded timestamp system to track the order of events in a concurrent system. In a timestamp system every process owns an *object*, an abstraction of a set of shared variables. One of the requirements of the system is to determine the temporal order in which the objects are written. For this purpose, each object is given a *label* (also referred to as a *timestamp*) which indicates the latest (relative) time when it has been written by its owner process. The processes assign labels to their respective objects in such a way that the labels reflect the real-time order in which they are written to. These systems must support two operations, namely *labeling* and *scan*. A labeling operation execution (Labeling, in short) assigns a new label to an object, and a scan operation execution (Scan, in short) enables a process to determine the ordering in which all the objects are written, that is, it returns a set of labeled-objects ordered temporally. The concern is with those systems where operations can be executed *concurrently*, in an overlapped fashion. Moreover, operation executions must be *wait-free*, that is, each operation execution will take a bounded number of its own steps (the number of accesses to the shared space), irrespective of the presence of other operation executions and their relative speeds. Israeli and Li [5] constructed a bit-optimal bounded timestamp system for *sequential* operation executions. Their sequential timestamp system was published in the above journal reference, but the preliminary concurrent timestamp system in the conference proceedings, of which a more detailed version has been circulated in manuscript form, has not been published in final form. The first generally accepted solution of the *concurrent* case of the bounded timestamp system was from Dolev and Shavit [3]. Their construction is of the type presented in [5]

and uses shared variables of size $O(n)$, where n is the number of processes in the system. Each Labeling requires $O(n)$ steps, and each Scan $O(n^2 \log n)$ steps. (A 'step' accesses an $O(n)$ bit variable.) In [4] the unbounded construction of [14] is corrected and extended to obtain an efficient version of the more general notion of a bounded concurrent timestamp system.

Applications

Wait-free registers are, together with message-passing systems, the primary interprocess communication method in distributed computing theory. They form the basis of all constructions and protocols, as can be seen in the textbooks. Wait-free constructions of concurrent timestamp systems (CTSs, in short) have been shown to be a powerful tool for solving concurrency control problems such as various types of mutual exclusion, multiwriter multireader shared variables [14], and probabilistic consensus, by synthesizing a "wait-free clock" to sequence the actions in a concurrent system. For more details see [4].

Open Problems

There is a great deal of work in the direction of register constructions that use less constituent parts, or simpler parts, or parts that can tolerate more complex failures, than previous constructions referred to above. Only, of course, if the latter constructions were not yet optimal in the parameter concerned. Further directions are work on wait-free higher-typed objects, as mentioned above, hierarchies of such objects, and probabilistic constructions. This literature is too vast and diverse to be surveyed here.

Experimental Results

Register constructions, or related constructions for asynchronous interprocess communication, are used in current hardware and software.

Cross-References

- ▸ Asynchronous Consensus Impossibility
- ▸ Atomic Broadcast
- ▸ Causal Order, Logical Clocks, State Machine Replication
- ▸ Concurrent Programming, Mutual Exclusion
- ▸ Linearizability
- ▸ Renaming
- ▸ Self-Stabilization
- ▸ Snapshots in Shared Memory
- ▸ Synchronizers, Spanners
- ▸ Topology Approach in Distributed Computing

Recommended Reading

1. Bloom B (1988) Constructing two-writer atomic registers. IEEE Trans Comput 37(12):1506–1514
2. Burns JE, Peterson GL (1987) Constructing multi-reader atomic values from non-atomic values. In: Proceedings of the 6th ACM symposium principles of distributed computing, Vancouver, 10–12 Aug 1987, pp 222–231
3. Dolev D, Shavit N (1997) Bounded concurrent time-stamp systems are constructible. SIAM J Comput 26(2):418–455
4. Haldar S, Vitanyi P (2002) Bounded concurrent timestamp systems using vector clocks. J Assoc Comput Mach 49(1):101–126
5. Israeli A, Li M (1993) Bounded time-stamps. Distr Comput 6:205–209 (Preliminary, more extended, version in: proceedings of the 28th IEEE symposium on foundations of computer science, pp 371–382, 1987)
6. Israeli A, Shaham A (1992) Optimal multi-writer multireader atomic register. In: Proceedings of the 11th ACM symposium on principles distributed computing, Vancouver, 10–12 Aug 1992, pp 71–82
7. Kirousis LM, Kranakis E, Vitányi PMB (1987) Atomic multireader register. In: Proceedings of the workshop distributed algorithms. Lecture notes computer science, vol 312. Springer, Berlin, pp 278–296
8. Lamport L (1986) On interprocess communication – part I: basic formalism, part II: algorithms. Distrib Comput 1(2):77–101
9. Li M, Tromp J, Vitányi PMB (1996) How to share concurrent wait-free variables. J ACM 43(4):723–746 (Preliminary version: Li M, Vitányi PMB (1987) A very simple construction for atomic multiwriter register. Technical report TR-01–87, Computer Science Department, Harvard University, Nov 1987)
10. Peterson GL (1983) Concurrent reading while writing. ACM Trans Program Lang Syst 5(1):56–65
11. Peterson GL, Burns JE (1987) Concurrent reading while writing II: the multiwriter case. In: Proceed-

ings of the 28th IEEE symposium on foundations of computer science, Los Angeles, 27–29 Oct 1987, pp 383–392
12. Singh AK, Anderson JH, Gouda MG (1994) The elusive atomic register. J ACM 41(2):311–339 (Preliminary version in: proceedings of the 6th ACM symposium on principles distributed computing, 1987)
13. Tromp J (1989) How to construct an atomic variable. In: Proceedings of the workshop distributed algorithms. Lecture notes in computer science, vol 392. Springer, Berlin, pp 292–302
14. Vitányi PMB, Awerbuch B (1987) Atomic shared register access by asynchronous hardware. In: Proceedings of the 27th IEEE symposium on foundations of computer science, Los Angeles, 27–29 Oct 1987, pp 233–243 (Errata, Proceedings of the 28th IEEE symposium on foundations of computer science, Los Angeles, 27–29 Oct 1987, pp 487–487)

Regular Expression Matching

Lucian Ilie
Department of Computer Science, University of Western Ontario, London, ON, Canada

Keywords

Bit parallelism; Glushkov-McNaughton-Yamada automaton; Regular expression matching and searching

Years and Authors of Summarized Original Work

1956; Kleene
1968; Thompson
1992; Wu, Manber
2005; Navarro, Raffinot
2009; Bille, Thorup

Problem Definition

Given a *text string* T of length n and a *regular expression* R, the **regular expression matching** problem **(REM)** is to find all text positions at which an occurrence of a string in $L(R)$ ends (see below for definitions).

For an alphabet Σ, a *regular expression* R over Σ consists of elements of $\Sigma \cup \{\varepsilon\}$ (ε denotes the empty string) and operators $\cdot$ (concatenation), $|$ (union), and $*$ (iteration, i.e., repeated concatenation); the set of strings $L(R)$ represented by R is defined accordingly; see [7]. It is important to distinguish two measures for the size of a regular expression: the *size*, m, which is the total number of characters from $\Sigma \cup \{\cdot, |, *\}$, and Σ-*size*, m_Σ, which counts only the characters in Σ. As an example, for $R = \texttt{(A|T)((C|CG)*)}$, the set $L(R)$ contains all strings that start with an `A` or a `T` followed by zero or more strings in the set $\{\texttt{C}, \texttt{CG}\}$; the size of R is $m = 8$ and the Σ-size is $m_\Sigma = 5$. Any regular expression can be processed in linear time so that $m = \mathcal{O}(m_\Sigma)$ (with a small constant); the difference becomes important when the two sizes appear as exponents.

Key Results

Finite Automata

The classical solutions for the REM problem involve finite automata which are directed graphs with the edges labelled by symbols from $\Sigma \cup \{\varepsilon\}$; their nodes are called states; see [7] for details. Unrestricted automata are called *nondeterministic finite automata (NFA)*. *Deterministic finite automata (DFA)* have no ε-labels and require that no two outgoing edges of the same state have the same label. Regular expressions and DFAs are equivalent, that is, the sets of strings represented are the same, as shown by Kleene [11]. There are two classical ways of computing an NFA from a regular expression. Thompson's construction [17] builds an NFA with up to $2m$ states and up to $4m$ edges whereas Glushkov-McNaughton-Yamada's automaton [5, 12] has the minimum number of states, $m_\Sigma + 1$, and $\mathcal{O}(m_\Sigma^2)$ edges; see Fig. 1. Any NFA can be converted into an equivalent DFA by the *subset construction*: each subset of the set of states of the NFA becomes a state of the DFA. The problem is that the DFA can have exponentially more states than the NFA.

Classical Solutions

A regular expression is first converted into an NFA or DFA which is then simulated on the text. In order to be able to search for a match starting anywhere in the text, a loop labelled by all elements of Σ is added to the initial state; see Fig. 1.

Searching with an NFA requires linear space, but many states can be active at the same time, and to update them all we need, for Thompson's NFA, $\mathcal{O}(m)$ time for each letter of the text; this gives Theorem 1. On the other hand, DFAs allow searching time that is linear in n but require more space for the automaton. Theorem 2 uses the DFA obtained from Glushkov-McNaughton-Yamada's NFA.

Theorem 1 (Thompson [17]) *The REM problem can be solved with an NFA in $\mathcal{O}(mn)$ time and $\mathcal{O}(m)$ space.*

Theorem 2 (Kleene [11]) *The REM problem can be solved with a DFA in $\mathcal{O}(n + 2^{m\Sigma})$ time and $\mathcal{O}(2^{m\Sigma})$ space.*

Lazy Construction and Modules

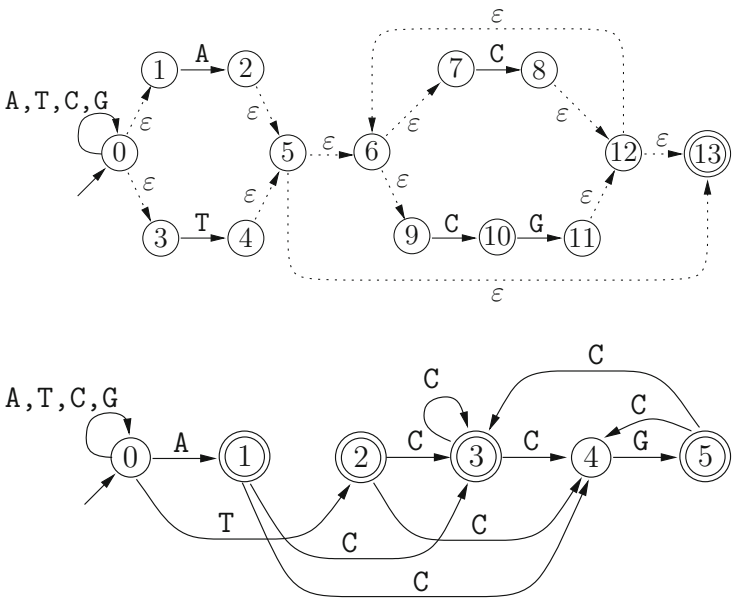
One heuristic to alleviate the exponential increase in the size of DFA is to build only the states reached while scanning the text, as implemented in *Gnu Grep*. Still, the space needed for the DFA remains a problem. A four-Russians approach was presented by Myers [13] where a trade-off between the NFA and DFA approaches is proposed. The syntax tree of the regular expression is divided into modules which are implemented as DFAs and are thereafter treated as leaf nodes in the syntax tree. The process continues until a single module is obtained. An $\mathcal{O}(mn/\log n)$ time and space algorithm is obtained. This bound was recently improved by Bille and Thorup [2].

Theorem 3 (Bille and Thorup [2]) *The REM problem can be solved in linear space and $\mathcal{O}\left(mn/(\log n)^{3/2}\right)$ time.*

The same authors showed in [3] that the length m of the regular expression can be essentially replaced in the complexity bounds by the number of strings (concatenations of characters) that appear in the regular expression.

Bit Parallelism

The simulation of the abovementioned modules is done by encoding all states as bits of a single computer word (called *bit mask*) so that all can be updated in a single operation. The method can be used without modules to simulate directly an NFA as done in [20] and implemented in the



Regular Expression Matching, Fig. 1 Thompson's NFA (*left*) and Glushkov-McNaughton-Yamada's NFA (*right*) for the regular expression `(A|T)((C|CG)*)`; the initial loops labelled `A,T,C,G` are not part of the construction; they are needed for REM

Agrep software [19]. Note that, in fact, the DFA is also simulated: a whole bit mask corresponds to a subset of states of the NFA, that is, one state of the DFA.

The bit-implementation of Wu and Manber [20] uses the property of Thompson's automaton that all Σ-labelled edges connect consecutive states, that is, they carry a bit 1 from position i to position $i+1$. This makes it easy to deal with the Σ-labelled edges, but the ε-labelled ones are more difficult. A table of size linear in the number of states of the DFA needs to be precomputed to account for the ε-closures (set of states reachable from a given state by ε-paths).

Note that in Theorems 1, 2, and 3, the space complexity is given in words. In Theorems 4 and 5 below, for a more practical analysis, the space is given in bits and the alphabet size is also taken into consideration. For comparison, the space in Theorem 2, given in bits, is $\mathcal{O}(|\Sigma| m_\Sigma 2^{m_\Sigma})$.

Theorem 4 (Wu and Manber [20]) *Thompson's automaton can be implemented using* $2m(2^{2m+1} + |\Sigma|)$ *bits.*

Glushkov-McNaughton-Yamada's automaton has different structural properties. First, it is ε-free, that is, there are no ε-labels on edges. Second, all edges incoming to a given state are labelled the same. These properties are exploited by Navarro and Raffinot [16] to construct a bit-parallel implementation that requires less space. The result is a simple algorithm for regular expression searching which uses less space and usually performs faster than any existing algorithm.

Theorem 5 (Navarro and Raffinot [16]) *Glushkov-McNaughton-Yamada's automaton can be implemented using* $(m_\Sigma + 1)(2^{m_\Sigma+1} + |\Sigma|)$ *bits.*

All algorithms in this category run in $\mathcal{O}(n)$ time, but smaller DFA representation implies more locality of reference and thus faster algorithms in practice. An improvement of any algorithm using Glushkov-McNaughton-Yamada's automaton can be done by reducing first the automaton by merging some of its states, as done by Ilie et al. [8, 9]. The reduction can be performed in such a way that all useful properties of the automaton are preserved. The search becomes faster due to the reduction in size.

Filtration

The above approaches examine every character in the text. In [18] a multipattern search algorithm is used to search for strings that must appear inside any occurrence of the regular expression. Another technique is used in *Gnu Grep*; it extracts the longest string that must appear in any match (it can be used only when such a string exists). In [16], bit-parallel techniques are combined with a reverse factor search approach to obtain a very fast character-skipping algorithm for regular expression searching.

Related Problems

Regular expressions with *backreference* have a feature that helps remembering what was matched to be used later; the matching problem becomes NP-complete; see [1]. *Extended* regular expressions involve adding two extra operators, intersection and complement, which do not change the expressive power. The corresponding matching problem can be solved in $\mathcal{O}((n+m)^4)$ time using dynamic programming; see [7, Exercise 3.23].

Concerning finite automata construction, recall that Thompson's NFA has $\mathcal{O}(m)$ edges, whereas the ε-free Glushkov-McNaughton-Yamada's NFA can have a quadratic number of edges. It has been shown in [4] that one can always build an ε-free NFA with $\mathcal{O}(m \log m)$ edges (for fixed alphabets). However, it is the number of states which is more important in the searching algorithms.

Applications

Regular expression matching is a powerful tool in text-based applications, such as text retrieval and text editing, and in computational biology to find various motifs in DNA and protein sequences. See [6] for more details.

Open Problems

The most important theoretical problem is whether linear time and linear space can be achieved simultaneously. Characterizing the regular expressions that can be searched for using a linear-size equivalent DFA is also of interest. The expressions consisting of a single string are included here – the algorithm of Knuth, Morris, and Pratt is based on this. Also, it is not clear how much we can reduce an NFA efficiently (as done by [8, 9]); the problem of finding a minimal NFA is PSPACE-complete; see [10]. Finally, for testing, it is not clear how to define random regular expressions.

Experimental Results

A disadvantage of the bit-parallel technique compared with the classical implementation of a DFA is that the former builds all possible subsets of states whereas the latter builds only the states that can be reached from the initial one (the other ones are useless). On the other hand, bit-parallel algorithms are simpler to code and more flexible (they allow also approximate matching), and there are techniques for reducing the space required. Among the bit-parallel versions, Glushkov-McNaughton-Yamada-based algorithms are better than Thompson-based ones. Modules obtain essentially the same complexity as bit-parallel ones but are more complicated to implement and slower in practice. As the number of computer words increases, bit-parallel algorithms slow down and modules may become attractive. Note also that technological progress has more impact on the bit-parallel algorithms, as opposed to classical ones, since the former depend very much on the machine word size. For details on comparison among various algorithms (including filtration based), see [15]; more recent comparisons are in [16], including the fastest algorithms to date.

URLs to Code and Data Sets

Many text editors and programming languages include regular expression search features. They are, as well, among the tools used in protein databases, such as PROSITE and SWISS-PROT, which can be found at www.expasy.org. The package *agrep* [20] can be downloaded from webglimpse.net and *nrgrep* [14] from www.dcc.uchile.cl/gnavarro/software.

Cross-References

- ▸ Approximate Regular Expression Matching is a more general problem where errors are allowed.

Recommended Reading

1. Aho A (1990) Algorithms for finding patterns in strings. In: van Leeuwen J (ed) Handbook of theoretical computer science. Algorithms and Complexity, vol A. MIT Press Cambridge, MA, pp 255–300
2. Bille P, Thorup M (2009) Faster regular expression matching. In: Albers S et al (eds) Automata, languages and programming. Springer, Berlin/Heidelberg, pp 171–182
3. Bille P, Thorup M (2010) Regular expression matching with multi-strings and intervals. In: Proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms, SIAM, Philadelphia, PA, pp 1297–1308
4. Geffert V (2003) Translation of binary regular expressions into nondeterministic ε-free automata with $\mathcal{O}(n \log n)$ transitions. J Comput Syst Sci 66(3):451–472
5. Glushkov V-M (1961) The abstract theory of automata. Russ Math Surv 16:1–53
6. Gusfield D (1997) Algorithms on strings, trees and sequences. Cambridge University Press, New York, N.Y.
7. Hopcroft J, Ullman J (1979) Introduction to automata, languages, and computation. Addison-Wesley, Reading, Mass
8. Ilie L, Navarro G, Yu S (2004) On NFA reductions. In: Karhumäki J et al (eds) Theory is forever. Lecture notes in computer science vol 3113. Springer-Verlag Berlin Heidelberg, pp 112–124
9. Ilie L, Solis-Oba R, Yu S (2005) Reducing the size of NFAs by using equivalences and preorders. In: Apostolico A, Crochemore M, Park K (eds) Combinatorial pattern matching. Springer, Berlin/Heidelberg, pp 310–321
10. Jiang T, Ravikumar B (1993) Minimal NFA problems are hard. SIAM J Comput 22(6):1117–1141
11. Kleene SC (1956) Representation of events in nerve sets. In: Shannon CE, McCarthy J (eds) Automata

studies. Princeton University Press, Princeton, N. J., pp 3–40
12. McNaughton R, Yamada H (1960) Regular expressions and state graphs for automata. IRE Trans Electron Comput 9(1):39–47
13. Myers E (1992) A four Russians algorithm for regular expression pattern matching. J ACM 39(2):430–448
14. Navarro G (2001) Nr-grep: a fast and flexible pattern matching tool. Softw Pract Experience 31:1265–1312
15. Navarro G, Raffinot M (2002) Flexible pattern matching in strings – practical on-line search algorithms for texts and biological sequences. Cambridge University Press, Cambridge, U.K.
16. Navarro G, Raffinot M (2005) New techniques for regular expression searching. Algorithmica 41(2):89–116
17. Thompson K (1968) Regular expression search algorithm. Commun ACM 11(6):419–422
18. Watson B (1995) Taxonomies and toolkits of regular language algorithms. PhD Dissertation, Eindhoven University of Technology, The Netherlands
19. Wu S, Manber U (1992) Agrep – a fast approximate patter-matching tool. In: Proceedings of the USENIX technical conference, San Francisco, California, pp 153–162
20. Wu S, Manber U (1992) Fast text searching allowing errors. Commun ACM 35(10):83–91

Reinforcement Learning

Eyal Even-Dar
Google, New York, NY, USA

Keywords

Neuro-dynamic programming

Years and Authors of Summarized Original Work

1992; Watkins

Problem Definition

Many sequential decision problems ranging from dynamic resource allocation to robotics can be formulated in terms of stochastic control and solved by methods of reinforcement learning. Therefore, reinforcement learning (a.k.a neuro-dynamic programming) has become one of the major approaches to tackling real-life problems.

In reinforcement learning, an agent wanders in an unknown environment and tries to maximize its long-term return by performing actions and receiving rewards. The most popular mathematical models to describe reinforcement learning problems are the Markov Decision Process (MDP) and its generalization, the partially observable MDP. In contrast to supervised learning, in reinforcement learning, the agent is learning through interaction with the environment and thus influences the "future." One of the challenges that arises in such cases is the exploration-exploitation dilemma. The agent can choose either to exploit its current knowledge and perhaps not learn anything new or to explore and risk missing considerable gains.

While reinforcement learning contains many problems, due to lack of space, this entry focuses on the basic ones. For a detailed history of the development of reinforcement learning, see [1, Chapter 1]. The focus of this entry is on Q-learning and Rmax.

Notation

Markov Decision Process

A Markov decision process (MDP) formalizes the following problem. An agent is in an environment, which is composed of different states. In each time step, the agent performs an action and as a result observes a signal. The signal is composed from the reward to the agent and the state it reaches in the next time step. More formally the MDP is defined as follows:

Definition 1 A Markov decision process (MDP) M is a 4-tuple (S,A,P,R), where S is a set of states, A is a set of actions, $P_{s,s'}{}^{a}$ is the transition probability from state s to state s' when performing action $a \in A$ in state s, and $R(s,a)$ is the reward distribution when performing action a in state s.

A strategy for an MDP assigns, at each time t, for each state s a probability for performing action $a \in A$, given a history $F_{t-1} =$

$\{s_1, a_1, r_1, \ldots, s_{t-1}, a_{t-1}, r_{t-1}\}$ which includes the states, actions, and rewards observed until time $t-1$. While executing a strategy π, an agent performs at time t action a_t in state s_t and observes a reward r_t (distributed according to $R(s_t, a_t)$), and a next state s_{t+1} (distributed according to $P_{s_t,\cdot}^{a_t}$). The sequence of rewards is combined into a single value called the *return*. The agent's goal is to maximize the return. There are several natural ways to define the return.

- *Finite horizon:* The return of policy π for a given horizon H is $\sum_{t=0}^{H} r_t$.
- *Discounted return:* For a discount parameter $\gamma \in (0,1)$, the discounted return of policy π is $\sum_{t=0}^{\infty} \gamma^t r_t$.
- *Undiscounted return:* The return of policy π is $\lim_{t\to\infty} \frac{1}{t+1} \sum_{i=0}^{t} r_i$.

Due to lack of space, only discounted return, which is the most popular approach mainly due to its mathematical simplicity, is considered. The value function for each state s, under policy π, is defined as $V^{\pi}(s) = E^{\pi}[\sum_{i=0}^{\infty} r_i \gamma^i]$, where the expectation is over a run of policy π starting at state s. The state-action value function for using action a in state s and then following π is defined as $Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s'} P_{s,s'}^{a} V^{\pi}(s')$.

There exists a stationary deterministic optimal policy, π^*, which maximizes the return from any start state [11]. This implies that for any policy π and any state s, $V^{\pi *}(s) \geq V^{\pi}(s)$, and $\pi^*(s) = \text{argmax}_a(Q^{\pi *}(s,a))$. A policy π is ε-optimal if $\|V^{\pi *} - V^{\pi}\|_{\infty} \leq \epsilon$.

Problems Formulation

The reinforcement learning problems are divided into two categories, planning and learning.

Planning

Given an MDP in its tabular form, compute the optimal policy. An MDP is given in its tabular form if the 4-tuple, (A, S, P, R) is given explicitly.

The standard methods for the planning problem in MDP are given below.

Value Iteration

Value iteration is defined as follows. Start with some initial value function, C_s, and then iterate using the Bellman operator, $TV(s) = \max_a R(s,a) + \gamma \sum_{s' \in S} P_{s,s'}^{a} V(s')$.

$$V_0(s) = C_s$$

$$V_{t+1}(s) = TV_t(s),$$

This method relies on the fact that the Bellman operator is contracting. Therefore, the distance between the optimal value function and current value function contracts by a factor of γ with respect to max norm (L_∞) in each iteration.

Policy Iteration

This algorithm starts with initial policy π_0 and iterates over polices. The algorithm has two phases for each iteration. In the first phase, the *value evaluation step*, a value function for π_t is calculated, by finding the fixed point of $T_{\pi t} V_{\pi t} = V_{\pi t}$, where $T_{\pi t} V = R(S, \pi_t(s)) + \gamma \sum_{s' \in S} P_{s,s'}^{\pi_t(s)} V(s')$. The second phase, *policy improvement step*, is taking the next policy π_{t+1} as a greedy policy with respect to $V_{\pi t}$. It is known that policy iteration converges with fewer iterations than value iteration. In practice the convergence of policy iteration is very fast.

Linear Programming

This approach formulates and solves an MDP as a linear program (LP). The LP variables are $V_1, \ldots, V_n$, where $V_i = V(s_i)$. The definition is:

Variables: $V_1, \ldots, V_n$
Minimize: $\sum_i V_i$
Subject to: $V_i \geq [R(s_i, a) + \gamma \sum_j P_{s_i, s_j}(a) V_j]$

$$\forall a \in A, s_i \in S.$$

Learning

Given the states and action identities, learn an (almost) optimal policy through interaction with

the environment. The methods are divided into two categories: model-free learning and model-based learning.

The widely used Q-learning [16] is a model-free algorithm. This algorithm belongs to the class of temporal difference algorithms [12]. Q-learning is an off-policy method, i.e., it does not depend on the underlying policy but, as can immediately be seen, depends on the trajectory and not on the policy generating the trajectory.

Q-Learning

The algorithm estimates the state-action value function (for discounted return) as follows:

$$Q_0(s,a) = 0$$
$$Q_{t+1}(s,a) = (1-\alpha_t(s,a))Q_t(s,a) + \alpha_t(s,a)(r_t(s,a) + \gamma V_t(s'))$$

where s' is the state reached from state s when performing action a at time t, and $V_t(s) = \max_a Q_t(s,a)$. Assume that $\alpha_t(s',a') = 0$ if at time t action a' was not performed at state s'. A learning rate α_t is *well behaved* if for every state action pair (s,a): (1) $\sum_{t=1}^{\infty} \alpha_t(s,a) = \infty$ and (2) $\sum_{t=1}^{\infty} \alpha_t^2(s,a) = \infty$. As will be seen, this is necessary for the convergence of the algorithm.

The model-based algorithms are very simple to describe; they simply build an empirical model and use any of the standard methods to find the optimal policy in the empirical (approximate) model. The main challenge in these methods is in balancing exploration and exploitation and having an appropriate stopping condition. Several algorithms give a nice solution for this [3, 7]. A version of these algorithms appearing in [6] is described below.

On an intuitive level, a state will become known when it was visited "enough" times and one can estimate with high probability its parameters with good accuracy. The modified empirical model is defined as follows. All states that are not in K are represented by a single absorbing state in which the reward is maximal (which causes exploration). The probability to move to the absorbing state from a state $s \in K$ is the empirical probability to move out of K from s and the probability to move between states in K is the empirical probability.

Algorithm 1: A model-based algorithm

```
Rmax
Set K = ∅;
if s ∈ K? then
   Execute π̂(s)
else
   Execute a random action;
   if s becomes known then
      K = K ∪ {s};
      Compute optimal policy, π̂ for
      the modified empirical model
   end
end
```

Sample complexity [6] measures how many samples an algorithm needs in order to learn. Note that the sample complexity translates into the time needed for the agent to wander in the MDP.

Key Results

The first Theorem shows that the planning problem is easy as long as the MDP is given in its tabular form, and one can use the algorithms presented in the previous section.

Theorem 1 ([10]) *Given an MDP, the planning problem is P-complete.*

The learning problem can be done also efficiently using the $R_{\max}$ algorithm as is shown below.

Theorem 2 ([3, 7]) *$R_{\max}$ computes an ε-optimal policy from state s with probability at least $1-\delta$ with sample complexity polynomial in $|A|, |S|, \frac{1}{\epsilon}$ and $\log \frac{1}{\delta}$, where s is the state in which the algorithm halts. Also the algorithm's computational complexity is polynomial in $|A|$ and $|S|$.*

The fact that Q-learning converges in the limit to the optimal Q function (which guarantees that the greedy policy with respect to the Q function will be optimal) is now shown.

Theorem 3 ([17]) *If every state-action is visited infinitely often and the learning rate is well behaved, then Q_t converges to Q^* with probability one.*

The last statement is regarding the convergence rate of Q-learning. This statement must take into consideration some properties of the underlying policy, and assume that this policy covers the entire state space in reasonable time. The next theorem shows that the convergence rate of Q-learning can vary according to the tuning of the algorithm parameters.

Theorem 4 ([4]) *Let L be the time needed for the underlying policy to visit every state action with probability 1/2. Let T be the time until $||Q^* - Q_T|| \leq \epsilon$ with probability at least $1 - \delta$ and $\#(s,a,t)$ be the number of times action a was performed at state s until time t. Then if $\alpha_t(s,a) = 1/\#(s,a,t)$, then T is polynomial in $L, \frac{1}{\epsilon}, \log\frac{1}{\delta}$ and exponential in $\frac{1}{1-\gamma}$. If $\alpha_t(s,a) = 1/\#(s,a,t)^{\omega}$ for $\omega \in (1/2, 1)$, then T is polynomial $L, \frac{1}{\epsilon}, \log\frac{1}{\delta}$ and $\frac{1}{1-\gamma}$.*

Applications

The biggest successes of reinforcement learning so far are mentioned here. For a list of successful applications of reinforcement learning, see http://neuromancer.eecs.umich.edu/cgi-bin/twiki/view/Main/SuccessesOfRL.

Backgammon Tesauro [14] used temporal difference learning combined with neural networks to design a player that learned to play backgammon by playing itself and resulted in a player at the level of the world's top players.

Helicopter control Ng et al. [9] used inverse reinforcement learning for autonomous helicopter flight.

Open Problems

While in this entry only MDPs given in their tabular form were discussed, much current research is dedicated to two major directions: large state space and partially observable environments.

In many real-world applications, such as robotics, the agent cannot observe the state she is in and can only observe a signal which is correlated with it. In such scenarios, the MDP framework is no longer suitable, and another model is in order. The most popular reinforcement learning for such environments is the partially observable MDP. Unfortunately, for POMDP even the planning problems are intractable (and not only for the optimal policy which is not stationary but even for the optimal stationary policy); the learning contains even more obstacles as the agent cannot repeat the same state twice with certainty, and thus, it is not obvious how she can learn. An interesting open problem is trying to characterize when a POMDP is "solvable" and when it is hard to solve according to some structure.

In most applications, the assumption that the MDP can be represented in its tabular form is not realistic and approximate methods are in order. Unfortunately not much theoretically is known under such conditions. Here are a few of the prominent directions to tackle large state space.

Function Approximation

The term "function approximation" is due to the fact that this approach takes examples from a desired function (e.g., a value function) and constructs an approximation of the entire function. Function approximation is an instance of supervised learning, which is studied in machine learning and other fields. In contrast to the tabular representation, this time a parameter vector Θ represents the value function. The challenge will be to learn the optimal vector parameter in the sense of minimum square error, i.e.,

$$\min_{\Theta} \sum_{s \in S} (V^{\pi}(s) - V(s,\Theta))^2,$$

where $V(s,\Theta)$ is the approximation function. One of the most important function approximations is the linear function approximation,

$$V_t(s,\Theta) = \sum_{i=1}^{T} \phi_s(i)\Theta_t(i),$$

where each state has a set of vector features, ϕ_s. A feature-based function approximation was analyzed and demonstrated in [2, 15]. The main goal here is designing algorithms which converge to almost optimal polices under realistic assumptions.

Factored Markov Decision Process

In an FMDP, the set of states is described via a set of random variables $X = \{X_1, \ldots, X_n\}$, where each X_i takes values in some finite domain $Dom(X_i)$. A state s defines a value $x_i \in Dom(X_i)$ for each variable X_i. The transition model is encoded using a dynamic Bayesian network. Although the representation is efficient, not only is finding an ε-optimal policy intractable [8], but it cannot be represented succinctly [1]. However, under assumptions on the FMDP structure, there exist algorithms such as [5] that have both theoretical guarantees and nice empirical results.

Cross-References

- ▸ Attribute-Efficient Learning
- ▸ Learning Automata
- ▸ Learning Constant-Depth Circuits
- ▸ PAC Learning

Recommended Reading

1. Allender E, Arora S, Kearns M, Moore C, Russell A (2002) Note on the representational incompatibility of function approximation and factored dynamics. In: Becker S, Thrun S, Obermayer K (eds) Advances in neural information processing systems 15. MIT, Cambridge
2. Bertsekas DP, Tsitsiklis JN (1996) Neuro-dynamic programming. Athena Scientific, Belmont
3. Brafman R, Tennenholtz M (2002) R-max – a general polynomial time algorithm for near optimal reinforcement learning. J Mach Learn Res 3:213–231
4. Even-Dar E, Mansour Y (2003) Learning rates for q-learning. J Mach Learn Res 5:1–25
5. Guestrin C, Koller D, Parr R, Venkataraman S (2003) Efficient solution algorithms for factored MDPs. J Artif Intell Res 19:399–468
6. Kakade S (2003) On the sample complexity of reinforcement learning. Ph.D. thesis, University College London
7. Kearns M, Singh S (2002) Near-optimal reinforcement learning in polynomial time. Mach Learn 49(2–3):209–232
8. Lusena C, Goldsmith J, Mundhenk M (2001) Nonapproximability results for partially observable Markov decision processes. J Artif Intell Res 14:83–103
9. Ng AY, Coates A, Diel M, Ganapathi V, Schulte J, Tse B, Berger E, Liang E (2006) Inverted autonomous helicopter flight via reinforcement learning. In: Ang MH Jr, Khatib O (eds) International symposium on experimental robotics. Springer tracts in advanced robotics 21. Springer, Berlin/New York
10. Papadimitriou CH, Tsitsiklis JN (1987) The complexity of Markov decision processes. Math Oper Res 12(3):441–450
11. Puterman M (1994) Markov decision processes. Wiley-Interscience, New York
12. Sutton R (1988) Learning to predict by the methods of temporal differences. Mach Learn 3:9–44
13. Sutton R, Barto A (1998) Reinforcement learning. An introduction. MIT, Cambridge
14. Tesauro GJ (1996) TD-gammon, a self-teaching backgammon program, achieves a master-level play. Neural Comput 6:215–219
15. Tsitsiklis JN, Van Roy B (1996) Feature-based methods for large scale dynamic programming. Mach Learn 22:59–94
16. Watkins C (1989) Learning from delayed rewards. Ph.D. thesis, Cambridge University
17. Watkins C, Dyan P (1992) Qlearning. Mach Learn 8(3/4):279–292

Renaming

Maurice Herlihy
Department of Computer Science, Brown University, Providence, RI, USA

Keywords

Wait-free renaming

Years and Authors of Summarized Original Work

1990; Attiya, Bar-Noy, Dolev, Peleg, Reischuk

Problem Definition

Consider a system in which $n + 1$ processes $P_0, \ldots, P_n$ communicate either by message-

passing or by reading and writing a shared memory. Processes are *asynchronous*: there is no upper or lower bounds on their speeds, and up to t of them may fail undetectably by halting. In the *renaming task* proposed by Attiya, Bar-Noy, Dolev, Peleg, and Reischuk [1], each process is given a unique *input name* taken from a range $0, \ldots, N$, and chooses a unique *output name* taken from a strictly smaller range $0, \ldots, K$. To rule out trivial solutions, a process's decision function must depend only on input names, not its preassigned identifier (so that P_i cannot simply choose output name i). Attiya et al. showed that the task has no solution when $K = n$, but does have a solution when $K = N + t$. In 1993, Herlihy and Shavit [2] showed that the task has no solution when $K < N + t$.

Vertexes, simplexes, and complexes model decision tasks.(See the companion article entitled ▸ Topology Approach in Distributed Computing). A process's state at the start or end of a task is represented as a vertex $\mathbf{v}$ labeled with that process's identifier, and a value, either input or output: $\mathbf{v} = \langle P, v_i \rangle$. Two such vertexes are *compatible* if (1) they have distinct process identifiers, and (2) those process can be assigned those values together. For example, in the renaming task, input values are required to be distinct, so two input vertexes are compatible only if they are labeled with distinct process identifiers and distinct input values.

Figure 1 shows the output complex for the three-process renaming task using four names. Notice that the two edges marked A are identical, as are the two edges marked B. By identifying these edges, this task defines a simplicial complex that is topologically equivalent to a torus. Of course, after changing the number of processes or the number of names, this complex is no longer a torus.

Key Results

Theorem 1 *Let S^n be an n-simplex, and S^m a face of S^n. Let S be the complex consisting of all faces of S^m, and $\dot{S}$ the complex consisting of all proper faces of S^m (the boundary complex of S). If $\sigma(\dot{S})$ is a subdivision of $\dot{S}$, and $\phi: \sigma(\dot{S}) \to \mathcal{F}(S)$ a simplicial map, then there exists a subdivision $\tau(S)$ and a simplicial map $\psi: \tau(S) \to \mathcal{F}(S)$ such that $\tau(\dot{S}) = \sigma(\dot{S})$, and ϕ and ψ agree on $\sigma(\dot{S})$.*

Informally, any simplicial map of an m-sphere to $\mathcal{F}$ can be "filled in" to a simplicial map of the $(m+1)$-disk. A *span* for $\mathcal{F}(S^n)$ is a subdivision σ of the input simplex S^n together with a simplicial map $\phi: \sigma(S^n) \to \mathcal{F}(S^n)$ such that for every face S^m of S^n, $\phi: \sigma(S^m) \to \mathcal{F}(S^m)$. Spans are constructed one dimension at a time. For each $\mathbf{s} = \langle P_i, v_i \rangle \in S^n, \phi$ carries $\mathbf{s}$ to the solo execution by P_i with input $\mathbf{v}_i$. For each $S^1 = (\mathbf{s}_0, \mathbf{s}_1)$, Theorem 1 implies that $\phi(\mathbf{s}_0)$ and $\phi(\mathbf{s}_1)$ can be joined by a path in $\mathcal{F}(S^1)$. For each $S^2 = (\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2)$, the inductively constructed spans define each face of the boundary complex $\phi: \sigma(S^1_{ij}) \to \mathcal{F}(S^1)_{ij}$, for $i, j \in \{0, 1, 2\}$. Theorem 1 implies that one can "fill in" this map, extending the subdivision from the boundary complex to the entire complex.

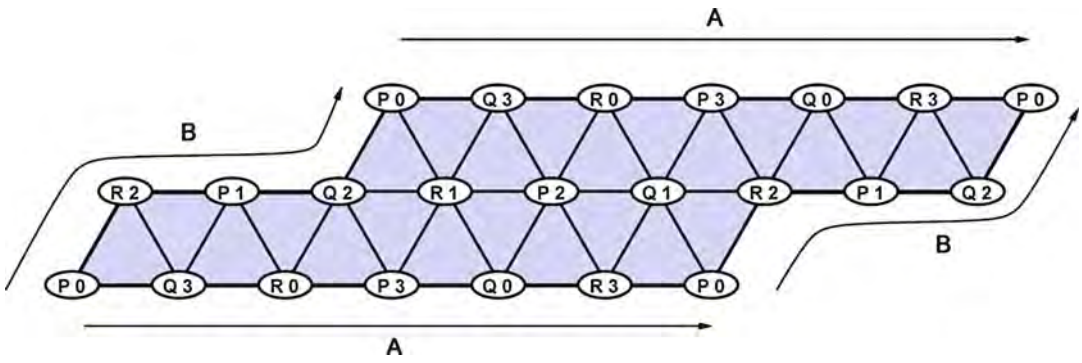
Theorem 2 *If a decision task has a protocol in asynchronous read/write memory, then each input simplex has a span.*

One can restrict attention to protocols that have the property that any process chooses the same name in a solo execution.

Definition 1 A protocol is *comparison-based* if the only operations a process can perform on processor identifiers is to test for equality and order; that is, given two P and Q, a process can test for $P = Q, P \leq Q$, and $P \geq Q$, but cannot examine the structure of the identifiers in any more detail.

Lemma 1 *If a wait-free renaming protocol for K names exists, then a comparison-based protocol exists.*

Proof Attiya et al. [1] give a simple comparison-based wait-free renaming protocol that uses $2n + 1$ output names. Use this algorithm to assign each



Renaming, Fig. 1 Output complex for 3-process renaming with 4 names

process an *intermediate* name, and use that intermediate name as input to the K-name protocol.□

Comparison-based algorithms are *symmetric* on the boundary of the span. Let S^n be an input simplex, $\phi: \sigma(S^n) \to \mathcal{F}(S^n)$ a span, and $\mathcal{R}$ the output complex for $2n$ names. Composing the span map ϕ and the decision map δ yields a map $\sigma(S^n) \to \mathcal{R}$. This map can be simplified by replacing each output name by its parity, replacing the complex $\mathcal{R}$ with the binary n-sphere $\mathcal{B}^n$.

$$\mu: \sigma(S^n) \to \mathcal{B}^n \ . \tag{1}$$

Denote the simplex of $\mathcal{B}^n$ whose values are all zero by 0^n, and all one by 1^n.

Lemma 2 $\mu^{-1}(0^n) = \mu^{-1}(1^n) = \emptyset$.

Proof The range $0, \ldots, 2n-1$ does not contain $n+1$ distinct even names or $n+1$ distinct odd names. □

The *n-cylinder* C^n is the binary n-sphere without 0^n and 1^n. Informally, the rest of the argument proceeds by showing that the boundary of the span is "wrapped around" the hole in C^n a non-zero number of times.

The span $\sigma(S^n)$ (indeed any any subdivided n-simplex) is a (combinatorial) *manifold with boundary*: each $(n-1)$-simplex is a face of either one or two n-simplexes. If it is a face of two, then the simplex is an *internal simplex*, and otherwise it is a *boundary* simplex. An orientation of S^n induces an orientation on each n-simplex of $\sigma(S^n)$ so that each internal $(n-1)$-simplex inherits opposite orientations. Summing these oriented simplexes yields a chain, denoted $\sigma_*(S^n)$, such that

$$\partial\sigma_*(S^n) = \sum_{i=0}^{n} (-1)^i \sigma_*(face_i(S^n)) \ .$$

The following is a standard result about the homology of spheres.

Theorem 3 *Let the chain 0^n be the simplex 0^n oriented like S^n. (1) For $0 < m < n$, any two m-cycles are homologous, and (2) every n-cycle C^n is homologous to $k \cdot \partial 0^n$, for some integer k. C^n is a boundary if and only if $k = 0$.*

Let S^m be the face of S^n spanned by solo executions of $P_0, \ldots, P_m$. Let 0^m denote some m-simplex of C^n whose values are all zero. Which one will be clear from context.

Lemma 3 *For every proper face S^{m-1} of S^n, there is an m-chain $\alpha(S^{m-1})$ such that*

$$\mu_*(\sigma_*(S^m)) - 0^m - \sum_{i=0}^{m} (-1)^i \alpha(face_i(S^m))$$

is a cycle.

Proof By induction on m. When $m=1$, $ids(S^1) = \{i, j\}$. 0^1 and $\mu_*(\sigma_*(S^1))$ are

1-chains with a common boundary $\langle P_i, 0\rangle - \langle P_j, 0\rangle$, so $\mu_*(\sigma_*(S^1)) - 0^1$ is a cycle, and $\alpha(\langle P_i, 0\rangle) = \emptyset$.

Assume the claim for $m, 1 \geq m < n-1$. By Theorem 3, every m-cycle is a boundary (for $m < n-1$), so there exists an $(m+1)$-chain $\alpha(S^m)$ such that

$$\mu_*(\sigma_*(S^m)) - 0^m - \sum_{i=0}^{m} (-1)^i \alpha(face_i(S^m)) = \partial\alpha(S^m) .$$

Taking the alternating sum over the faces of S^{m+1}, the $\alpha(face_i(S^m))$ cancel out, yielding

$$\mu_*(\partial\sigma_*(S^{m+1})) - \partial 0^{m+1} = \sum_{i=0}^{m+1} (-1)^i \partial\alpha(face_i(S^{m+1})) .$$

Rearranging terms yields

$$\partial \left(\mu_*(\sigma_*(S^{m+1})) - 0^{m+1} - \sum_{i=0}^{m+1} (-1)^i \alpha(face_i(S^{m+1})) \right) = 0 ,$$

implying that

$$\mu_*(\sigma_*(S^{m+1})) - 0^{m+1} - \sum_{i=0}^{m+1} (-1)^i \alpha(face_i(S^{m+1}))$$

is an $(m+1)$-cycle. □

Theorem 4 *There is no wait-free renaming protocol for $(n+1)$ processes using $2n$ output names.*

Proof Because

$$\mu_*(\sigma_*(S^{n-1})) - 0^{n-1} - \sum_{i=0}^{n} (-1)^i \alpha(face_i(S^{n-1}))$$

is a cycle, Theorem 3 implies that it is homologous to $k \cdot \partial 0^n$, for some integer k. Because μ is symmetric on the boundary of $\sigma(S^n)$, the alternating sum over the $(n-1)$-dimensional faces of S^n yields:

$$\mu_*(\partial\sigma_*(S^n)) - \partial 0^n \sim (n+1)k \cdot \partial 0^n$$

or

$$\mu_*(\partial\sigma_*(S^n)) \sim (1 + (n+1)k) \cdot \partial 0^n .$$

Since there is no value of k for which $(1 + (n+1)k)$ is zero, the cycle $\mu_*(\partial\sigma_*(S^n))$ is not a boundary, a contradiction. □

Applications

The renaming problem is a key tool for understanding the power of various asynchronous models of computation.

Open Problems

Characterizing the full power of the topological approach to proving lower bounds remains an open problem.

Cross-References

- Asynchronous Consensus Impossibility
- Set Agreement
- Topology Approach in Distributed Computing

Recommended Reading

1. Attiya H, Bar-Noy A, Dolev D, Peleg D, Reischuk R (1990) Renaming in an asynchronous environment. J ACM 37(3):524–548
2. Herlihy MP, Shavit N (1993) The asynchronous computability theorem for t-resilient tasks. In: Proceedings 25th annual ACM symposium on theory of computing, pp 111–120

Revenue Monotone Auctions

Gagan Goel[1] and Mohammad Reza Khani[2]
[1]Google Inc., New York, NY, USA
[2]University of Maryland, College Park, MD, USA

Keywords

Algorithmic Game Theory; Approximation; Electronic Commerce; Incentive compatibility; Mechanism Design; Price of revenue monotonicity; Revenue monotonicity; Social welfare

Introduction

Fueled by the growth of Internet and advancements in online advertising techniques, today more and more online firms rely on advertising revenue for their business. Some of these firms include news agencies, media outlets, search engines, social and professional networks, etc. Much of this online advertising business is moving to what's called *programmatic* buying where an advertiser bids for each single impression, sometimes in real time, depending on how he values the ad opportunity. This work is motivated by the need of a desired property in the auction mechanisms that are used in these bid-based advertising systems.

A standard mechanism for most auction scenarios is the famous Vickrey-Clarke-Groves (VCG) mechanism. VCG is *incentive compatible* (IC) and maximizes *social welfare*. Incentive compatibility guarantees that the best response for each advertiser is to report its true valuation. This makes the mechanism transparent and removes the load from the advertisers to calculate the best response. *Social welfare* is the sum of the valuations of the winners. This value is treated as a proxy for how much all the participants gain from the transaction. What makes VCG mechanism versatile is that it reduces the mechanism design problem into an optimization problem for any scenario.

Even though this versatility of VCG mechanism makes it a popular choice mechanism, however, it doesn't satisfy an important property, namely, that of *revenue monotonicity*. Revenue monotonicity says that if one increases the bid values or add new bidders, the total revenue should not go down. To see that VCG is not revenue monotone, consider a simple example of two items and three bidders (A, B, and C). Say, bidder A wants only the first item and has a bid of 2. Similarly bidder B wants only the second item and has a bid of 2. Bidder C wants both the items or nothing and has a bid of 2. Now if only bidders A and B participate in the auction, then VCG gives a revenue of 2; however, if all the three bidders participate, then the revenue goes down to 0.

This lack of revenue monotonicity (which has been noted several times in the literature) is one of the serious practical drawbacks of the celebrated VCG mechanism. To think of it, an online firm that depends on advertising revenue puts significant resources in its sales efforts to attract more bidders as the general belief is that more bidders imply more competition which should lead to higher prices. Now to tell this firm that their revenue can go down if they get more bidders can be strategically very confusing for them. To see this from another perspective, say, in a search engine firm, there is a team which makes a UI change that increases the click-through probability (CTR) of the search ads. These changes are thought of as good changes in the firm as they increase the effective bid of the bidders (the effective bid of a bidder in search advertising is a function of its cost-per-click bid and the CTR of its ad). Now if after making the change, the revenue goes down, what was supposed to be a good change may seem like a bad change. The point we are trying to make is that there are many teams in a firm, and for these teams to function properly, it is important that the auction mechanisms satisfy *revenue monotonicity*.

In this entry, with a focus on auctions arising in advertising scenarios, we seek to understand mechanisms that satisfy this additional property

of revenue monotonicity (RM). It is well known that for various settings (including ours), no mechanism can satisfy both IC and RM properties while attaining optimal social welfare. In fact it is known that one cannot even hope to get Pareto-optimality in social welfare while attaining both IC and RM [10]. Thus to overcome this bottleneck and develop an understanding of RM mechanisms, we relax the requirement of attaining full social welfare and define the notion of *price of revenue monotonicity* (PoRM). Price of revenue monotonicity of an IC and RM mechanism M is the ratio of optimal social welfare to the social welfare attained by the mechanism M. The goal is to design mechanisms that satisfy IC and RM properties and at the same time achieve low price of revenue monotonicity. To the best of our knowledge, this is the first work that defines and studies this notion of price of revenue monotonicity.

We study two different advertising settings in this entry. The first setting we study is the *image-text* auction. In image-text auction there is a special box designated for advertising in a publisher's website which can be filled by either k text ads or a single image ad. The second setting is the *video-pod* auction where an advertising break of a certain duration in a video content can be filled with multiple video ads of possibly different durations.

We note that revenue monotonicity is an *across-instance* constraint as it requires total revenue to behave in a certain manner across different instances, where a single instance is defined by fixing the *type* of the buyers. Note that incentive compatibility is also an across-instance constraint. A lot of research effort has gone into understanding incentive compatibility, which has resulted in useful tools for designing incentive-compatible mechanisms. Surprisingly, hardly any work has gone into understanding and building tools for designing mechanisms which satisfy the desired property of revenue monotonicity. We believe that understanding revenue monotonicity will shed new fundamental insights into the design of mechanisms for many practical scenarios.

Related Work

Ausubel and Milgrom [1] show that VCG satisfies RM if bidders' valuations satisfy *bidder submodularity*. Bidders' valuations satisfy bidders submodularity if and only if for any bidder i and any two sets of bidders S, S' with $S \subseteq S'$ we have $\text{WF}(S \cup \{i\}) - \text{WF}(S) \geq \text{WF}(S' \cup \{i\}) - \text{WF}(S')$, where $\text{WF}(S)$ is the maximum social welfare achievable using only S. Note that this is a general tool one can use to design revenue-monotone mechanisms – restrict the range of the possible allocations such that we get bidder submodularity when we run VCG on this range. However, we can show that this general tool is not so powerful by showing that for our auction scenarios, it is not possible to get a mechanism with PoRM better than $\Omega(k)$ by using the above tool.

Ausubel and Milgrom [1] also show that bidder submodularity is guaranteed when the goods are substitutes, i.e., the valuation function of each bidder is submodular over the goods. However, for many practical scenarios, including ours, the valuation function of the bidders is not submodular. Ausubel and Milgrom [1] design mechanisms which select allocations that are in the core of the exchange economy for combinatorial auctions. Here an allocation is in the core if there is no coalition of bidders and the seller to trade with each other in a way which is preferred by all the members of the coalition to the allocation. Day and Milgrom [3] show that core-selecting mechanisms that choose a core allocation which minimizes the seller's revenue satisfy RM given bidders follow so-called *best-response truncation strategy*. Therefore the core-selecting mechanism designed by [1] satisfies RM if the participants play such best-response strategy, although this mechanism is not incentive compatible.

Rastegari et al. [10] prove that no mechanism for general combinatorial auctions which satisfies IC and RM can achieve weakly maximal social welfare. An allocation is weakly maximal if it cannot be modified to make at least one participant better off without hurting anyone else. In another work [9] they design a randomized mech-

anism for combinatorial auctions which achieves weak maximality and expected revenue monotonicity.

Another related work is around the characterization of mechanisms that achieve the IC property. The classic result of Roberts [11] states that affine maximizers are the only social choice functions that can be implemented using IC mechanisms when bidders have unrestricted quasi-linear valuations. Subsequent works study the restricted cases [2, 6, 12, 13].

There is also an extensive body of research around designing mechanisms with good bounds on the revenue. Myerson [7] designs a mechanism which achieves the optimal expected revenue in the single parameter Bayesian setting. Goldbert et al. consider optimizing revenue in prior-free settings (see [8] for a survey on this).

Our Results

As mentioned earlier, we study two settings: (1) image-text auction and (2) video-pod auction. Both these settings can be described using the following abstract model. Say, there is a seller selling k identical items to n participants/buyers. Participant i wants either d_i items or nothing and has a valuation of v_i if it gets d_i items or 0 otherwise. Demand d_i is assumed to be public knowledge, and valuation v_i is assumed to be the private information of the participant i. We want to design a mechanism that is incentive compatible, individually rational (IR), and revenue monotone and maximizes social welfare.

For the image-text auction, the demand $d_i \in \{1, k\}$, i.e., each participant wants either 1 item (text ads) or k items (image ad). For the video-pod auction, an item corresponds to a unit time interval (say, one second), and the demand d_i could be any number between 1 and k, i.e., $d_i \in [k]$.

The first result of this entry is the following theorem.

Theorem 1 *We design a deterministic mechanism for image-text auction (*MITA*) which satisfies individual rationality (IR), IC, and RM with* PoRM *of at most* $\sum_{i=1}^{k} \frac{1}{i} \simeq \ln(k)$*, i.e., the ratio of* MITA*'s welfare over the optimal welfare is at most* $\ln(k)$.

The proof of Theorem 1 appears in section "Image-Text Auctions." We outline our mechanism over here: Let $v_1 \geq \ldots \geq v_{n_1}$ be the valuations of text participants and V_1 be the maximum valuation of the image participants. If $\max_{j \in [k]} j \cdot v_j$ is less than V_1, MITA gives all the items to the image participant who has valuation V_1; otherwise MITA picks the highest j^* text participants as the winners where j^* is the maximum number in $[k]$ such that $j^* \cdot v_{j^*} \geq V_1$. Note that the j that maximizes $j \cdot v_j$ might be less than the $j*$ which is the largest j such that $j \cdot v_j \geq V_1$. Also note that MITA sometimes picks less than k text ads as the winner (even if there are k or more text ads). VCG always picks the maximum number of text ads (if it decides to allocate the slot to text ads); this is one of the reasons why VCG fails to satisfy RM. When we allow lesser number of text ads to be declared as winners, intuitively, this increases the competition which boosts the revenue and thus helps in achieving RM, although this comes with a loss in social welfare.

Surprisingly, we can also show that the above mechanism achieves the optimal PoRM for the image-text auction by proving a matching lower bound. We show that a mechanism that satisfies IR, IC, RM, and two additional mild assumptions of anonymity (AM) and independence of irrelevant alternatives (IIA) cannot achieve a PoRM better than $\sum_{i=1}^{k} \frac{1}{i}$. Anonymity means that the auction mechanism doesn't depend on the identities of the participants (a formal definition appears in section "Image-Text Auctions"). IIA means that decreasing the bid of a losing participant shouldn't hurt any winner. Note that our mechanism satisfies both AM and IIA as well. Formally, we prove the following theorem whose proof appears in section "Image-Text Auctions."

Theorem 2 *There is no deterministic mechanism which satisfies IR, IC, RM, AM, and IIA and has* PoRM *less than* $\sum_{i=1}^{k} \frac{1}{i}$.

Finally we prove the following theorem for video-pod auctions.

Theorem 3 *We design a mechanism for video-pod auction (*MVPA*) which satisfies IR, IC, and RM with* PoRM *of at most* $(\lfloor \log k \rfloor + 1) \cdot (2 + \ln k)$.

We give the formal proof of Theorem 3 in section "Video-Pod Auctions" and outline the mechanism here. MVPA partitions the participants into $(\lfloor \log k \rfloor + 1)$ groups where each group $g \in [\log k]$ contains only the participants whose demands are in the range $[2^{g-1}, 2^g)$. MVPA selects winners only from one group. We round up the size of each participant in group g to 2^g; thus we can have at most $\frac{k}{2^g}$ number of winners from the group g. Let $v_1^{(g)} \geq \ldots \geq v_p^{(g)}$ be the sorted valuations of all the participants in group g. We define the max possible revenue of group g (MPRG(g)) to be

$$\text{MPRG}(g) = \max_{j \in [k/2^g]} j \cdot v_j^{(g)}.$$

As the name of MPRG(g) suggests, its value captures the maximum revenue we can truthfully obtain from group g without violating revenue monotonicity. Let g^* be the group with the highest MPRG value and group g' be the group whose MPRG is the second highest. The set of winners are the first j participants from group g^* where j is the largest number in $[k/2^g]$ such that $j \cdot v_j^{(g^*)}$ is greater than or equal MPRG(g'). We show that PoRM of MVPA is $(\lfloor \log k \rfloor + 1) \cdot (2 + \ln k)$.

Preliminaries

Let $N = \{1, \ldots, n\}$ be the set of all participants and k be the number of identical items. We denote the type of participant i by $\theta_i = (d_i, v_i) \in [k] \times \mathbb{R}^+$, where d_i is the number of items participant i demands and v_i is her valuation for getting d_i items. Note that the valuation of player i for getting less than d_i items is 0. Now in the image-text auction, participants have demand of either 1 or k. In the video-pod auction, participants can have arbitrary demands in $\{1, \ldots, k\}$. Let's denote the set of all possible types $[k] \times \mathbb{R}^+$ by Θ and the set of all type profiles of n participants by $\Theta^n = \underbrace{\Theta \times \ldots \times \Theta}_{n}$.

A deterministic mechanism $\mathcal{M}$ consists of an allocation rule $x : \Theta^n \to 2^n$ which maps each type profile to a subset of participants as the winners and payment rule $p : \Theta^n \to \left(\mathbb{R}^+\right)^n$ which maps each type profile to the payments of each participant.

Let $\theta = (\theta_1, \theta_2, \ldots, \theta_n) \in \Theta^n$ be a specific type profile. Also let $\mathcal{A}_\theta$ be the set of all feasible solutions, i.e.,

$$\mathcal{A}_\theta = \left\{ S \subseteq N \middle| \sum_{i \in S} d_i \leq k \right\}.$$

For each feasible solution $A \in \mathcal{A}_\theta$, the social welfare of A (denoted by WF(A)) is equal to $\sum_{\theta_i \in A} v_i$. To evaluate the social welfare of a mechanism $\mathcal{M}$ on a type profile θ, we compare the welfare of its solution to the optimal solution.

Definition 1 The welfare ratio of mechanism $\mathcal{M} = (x, p)$ on type profile $\theta \in \Theta^n$ (denoted by WFR$(\mathcal{M}, \theta)$) is the following:

$$\text{WFR}(\mathcal{M}, \theta) = \frac{\max_{A \in \mathcal{A}_\theta} \text{WF}(A)}{\text{WF}(x(\theta))}$$

To capture the worst-case loss in social welfare across all type profiles, we define the notion of *price of revenue monotonicity*.

Definition 2 The Price of Revenue Monotonicity of a mechanism $\mathcal{M}$ (denoted by PoRM$(\mathcal{M})$) is defined as follows:

$$\text{PoRM}(\mathcal{M}) = \max_{\theta \in \Theta^n} \text{WFR}(\mathcal{M}, \theta)$$

The desired goal is to design mechanisms which have low PoRM value, where the best possible value is 1.

Note that since we are interested in mechanisms with bounded PoRM, we restrict ourselves to mechanisms that satisfy consumer sovereignty. Consumer sovereignty says that any participant can be a winner as long as he bids high enough.

Now we will define a weakly monotone allocation rule which is used in the characterization of deterministic IC mechanisms. Let function $x_i : \Theta^n \to \{0, 1\}$ be the restriction of function x to participant i. Here $x_i(.)$ is one if participant i is a winner and zero otherwise.

Definition 3 We call allocation function x is weakly monotone if for any type profile $\theta \in \Theta^n$ and any participant $i \in [n]$ with demand

R

d_i, function $x_i((d_i, v_i), \theta_{-i})$ is a non-decreasing function in v_i.

Note that if a deterministic mechanism $\mathcal{M}$ satisfies consumer sovereignty and has a weakly monotone allocation function, then function $x_i((d_i, v_i), \theta_{-i})$ is a single-step function. The value at which the function $x_i((d_i, v_i), \theta_{-i})$ jumps from zero to one, i.e., the smallest value at which the participant i becomes a winner, is called *critical value*.

Definition 4 Let $\mathcal{M} = (x, p)$ be a deterministic mechanism that satisfies consumer sovereignty and has a weakly monotone allocation function; the critical value of participant i in type profile θ is $v_i^* = sup\{v_i | x_i((d_i, v_i), \theta_{-i}) = 0\}$.

The following lemma characterizes deterministic IC mechanisms (first given by [7]). We provide a proof sketch for the sake of completeness (for a complete proof, see, e.g., [8]).

Lemma 1 *Let $\mathcal{M} = (x, p)$ be a mechanism which satisfies IR. Mechanism $\mathcal{M}$ is truthful (IC) if and only if the following hold:*

1. *x is weakly monotone.*
2. *If participant i is a winner, then its payment is its critical value (v_i^*).*

Proof First we prove that if $\mathcal{M}$ is truthful, then it satisfies both conditions 1 and 2. We prove the first condition by contradiction. If x is not monotone, then there exist participant i, type profile θ, and two values $v_i^{(1)} > v_i^{(2)}$ such that i wins in type profile $\left((d_i, v_i^{(2)}), \theta_{-i}\right)$ but loses in type profile $\left((d_i, v_i^{(1)}), \theta_{-i}\right)$. This makes incentive for participant i to lie for type profile $\left((d_i, v_i^{(1)}), \theta_{-i}\right)$ and announce its valuation as $v_i^{(2)}$.

Consider an arbitrary participant i who is a winner; now we prove that the payment of participant i is its critical value. Assume for contradiction that mechanism $\mathcal{M}$ charges participant i amount c_i where $c_i < v_i^*$ in a type profile $((d_i, v_i), \theta_{-i})$. In this case, if participant i had type $(d_i, \hat{v_i})$ where $c_i < \hat{v_i} < v_i^*$, then i is not a winner in $((d_i, \hat{v_i}), \theta_{-i})$ as v_i^* is the critical value. Therefore, if the real type of participant i is $(d_i, \hat{v_i})$, she has incentive to lie her type as (d_i, v_i), become a winner, and pay c_i. Hence, the payment cannot be less than v_i^*. Now suppose that there exists value v_i for which mechanism $\mathcal{M}$ charges i amount c_i which is more than v_i^*. In this case, if participant i had type $(d_i, \hat{v_i})$ where $v_i^* < \hat{v_i} < c_i$, then i is still a winner (as v_i^* is the critical value) and pays at most $\hat{v_i}$ (as $\mathcal{M}$ satisfies IR). Therefore, she has an incentive to lie her type as $(d_i, \hat{v_i})$, become a winner, and pay at most $\hat{v_i}$. Hence, the payment cannot be more than v_i^* for any winning valuation v_i.

For the other direction, it is easy to check that any IR mechanism that satisfies conditions 1 and 2 is truthful. □

Image-Text Auctions

In this section we give our *mechanism for image-text auction* (MITA) which satisfies IR, IC, RM, and PoRM(MITA) $\leq \ln k$. Recall that in the image-text auction we have k identical items to sell and there are two groups of participants: the ones who want all the k items which we call *image participants* and the ones who want only one item which we refer to as *text participants*. As a result there are also two possible types of outcome: MITA gives all the items to an image participant; or it gives an item to each member of a subset of the text participants.

We start with explaining why VCG fails to satisfy RM and how we address this issue in MITA. Consider the type profile where we have one image participant with type $(k, 1)$ and one text participant with type $(1, 1)$. In this case either of the participants can be the winner. The payment of the winner in VCG is her critical value which is one. However if we add one more text participant with the same type $(1, 1)$, the two text participants win and each of them pays zero. The reason for the payment drop is that VCG always selects k winners from the text participants. This decreases the critical value of each text participant as the valuation of the other text participants helps her to win against image participants. In our mechanism we overcome this issue by not guaranteeing that the maximal number of text

participants can win an item. In other words, in our mechanism it is possible that less than k text participants win an item even if there are more than k text participants. This way, intuitively, even if the number of text participants increases, it potentially creates more competition and hence increases the payments.

Let θ be an arbitrary-type profile where there are n_1 text participants with types $(1, v_1), \ldots, (1, v_{n_1})$ and n_2 image participants with types $(k, V_1), \ldots, (k, V_{n_2})$. We define mechanism $\text{MITA} = (x^{\text{MITA}}, p^{\text{MITA}})$ by giving allocation function x^{MITA} which is weakly monotone. Given the allocation function, we obtain payment function p^{MITA} using the critical values defined in Lemma 1 which makes the mechanism truthful.

Allocation rule of MITA. Without loss of generality, we assume that $v_1 \geq v_2 \geq \ldots \geq v_{n_1}$ and $V_1 \geq V_2 \geq \ldots \geq V_{n_2}$. Also, we assume that $n_1 \geq k$; if not, we add fake text participants with value 0. For each $j \in [k]$, we consider value $j \cdot v_j$. Let candidate set C_θ contain all the values $j \in [k]$ such that $j \cdot v_j$ is greater than or equal to V_1, i.e., $C_\theta = \{j \in [k] | j \cdot v_j \geq V_1\}$. If C_θ is empty, the image participant with type (k, V_1) wins. If C_θ is nonempty, then let j^* be the maximum member of C_θ, i.e., $j^* = \max_{j \in C_\theta} j$. In this case the first j^* text participants win.

Observation 1 *Allocation function x^{MITA} is weakly monotone.*

Proof Recall from Definition 3, in order to prove that x^{MITA} is weakly monotone, we have to show that for any participant $i \in [n]$ with demand d_i, function $x_i\left((d_i, v_i), \theta_{-i}\right)$ is a non-decreasing function in v_i.

If i is an image participant, then i wins if its valuation is larger than $\max(W, \max_{j \in [k]} j \cdot v_j)$ where W is the largest valuation of the image participants in θ_{-i}. Moreover, bidder i loses for any value smaller than or equal to $\max(W, \max_{j \in [k]} j \cdot v_j)$. Therefore x_i is weakly monotone.

If i is a text participant, then let $v'_1 \geq v'_2 \geq \ldots$ be the sorted valuations of the text participants and V_1 be the largest valuation of image participants in θ_{-i}. Let t be the smallest value such that there exist $j \in [k-1]$ where $v'_{j+1} \leq t \leq v'_j$ and $(j+1) \cdot t$ is greater than or equal to V_1. If the valuation of bidder i is larger than or equal to t, then she wins since $(j+1) \cdot t \geq V_1$; otherwise she does not win since t is the smallest value for which there exist $j \in [k-1]$ such that $(j+1) \cdot t \geq V_1$. Therefore x_i is weakly monotone. □

In the following lemma we obtain the critical value (or truthful payments) of the winners in x^{MITA} using Lemma 1. The lemma also gives an intuition to why we select j^* text participants to win, which is the maximum j such that $j \cdot v_j \geq V_1$.

Lemma 2 *If C_θ, where $C_\theta = \{j \in [k] | j \cdot v_j \geq V_1\}$, is empty, then the first image participant wins all the items with critical value $\max(V_2, \max_{j \in [k]} j \cdot v_j)$. If C_θ is not empty, the first j^* text participants win the items where $j^* = max_{j \in C_\theta} j$ and all of them have critical value $\max(v_{k+1}, \frac{V_1}{j^*})$.*

Proof We find the critical value (Definition 4) of a winner by showing that if she has any valuation larger than the critical value she wins and for any valuation less than the critical value she doesn't.

If C_θ is empty, then the first image participant (with type (k, V_1)) wins all the items. As long as V_1 is larger than $\max(V_2, \max_{j \in [k]} j \cdot v_j)$, participant (k, V_1) wins. If V_1 is less than $\max(V_2, \max_{j \in [k]} j \cdot v_j)$, then she loses to the image participant (k, V_2) if $\max(V_2, \max_{j \in [k]} j \cdot v_j) = V_2$ or loses to the text participants if $\max(V_2, \max_{j \in [k]} j \cdot v_j) = \max_{j \in [k]} j \cdot v_j$. This means that the critical value of the first image participant is $\max(V_2, \max_{j \in [k]} j \cdot v_j)$ if she is the winner.

If C_θ is nonempty, then the first j^* text participants win. Let $i \in [j^*]$ be an arbitrary winner. First we observe that for any valuation v'_i greater than or equal to $\max\left(v_{k+1}, \frac{V_1}{j^*}\right)$, participant i remains as a winner in type profile $\theta' = ((1, v'_i), \theta_{-i})$. This is because for any such change in valuation of participant i number j^* remains in set $C_{\theta'}$. Moreover, this change does not add any new number j' to $C_{\theta'}$ such that $j' > j^*$ because the valuations of the text participants with index greater than j^* are not changed in θ'.

R

In order to prove that for any valuation v_i' less than critical value $\max\left(v_{k+1}, \frac{V_1}{j^*}\right)$, participant i is not a winner we consider two cases: (A) when the critical value is equal to $\frac{V_1}{j^*}$ and (B) when the critical value is equal to v_{k+1}.

Case (A): We prove this case by contradiction. Let v_i' be a valuation less than $\frac{V_1}{j^*}$ for which participant i is in the set of winners in type profile $\theta' = ((1, v_i'), \theta_{-i})$. Because v_i' is less than $\frac{V_1}{j^*}$, the number of winners which contains participant i cannot be less than or equal to j^* in type profile θ'. Let $j' \in [k]$ which is greater than j^* be the number of winners in θ'. This means that there are at least j' participants whose valuation is larger than $\frac{V_1}{j'}$ in θ'. Note that all the valuations in θ are the same as θ' except v_i which is decreased to v_i'; therefore, there are also at least j' participants whose valuation is larger than $\frac{V_1}{j'}$ in θ and hence j' is in set C_θ. This contradicts with the fact that j^* is the largest member of C_θ.

Case (B): In case (B) we have $\max\left(v_{k+1}, \frac{V_1}{j^*}\right) = v_{k+1}$ which implies that $k \cdot v_{k+1}$ is larger than V_1 as $j^* \in [k]$. Therefore Case (B) can only happen when $j^* = k$. Now consider participant i decreases its valuation to value v_i' that is less than v_{k+1}; then it cannot be a winner as there are k other participants whose valuations are more than v_i' while we have only k items. □

The payment function of MITA is set to the critical values of the winners as specified in Lemma 2 which by using Observation 1 and Lemma 1 implies MITA satisfies IC. Moreover, as the payments are always less than the participants' bid, IR property of MITA follows. Finally in the following lemma, we show that MITA is revenue monotone.

Lemma 3 *Let θ' be the type profile obtained by either increasing the valuation of a participant or adding a new participant to the type profile θ; then we have* $\text{REVENUE}(\text{MITA}, \theta') \geq \text{REVENUE}(\text{MITA}, \theta)$.

Proof Let $v_1 \geq v_2 \geq \ldots$ be the valuations of text participants and $V_1 \geq V_2 \geq \ldots$ be the valuations of image participants in θ. Similarly let $v_1' \geq v_2' \geq \ldots$ be the valuations of text participants and $V_1' \geq V_2' \geq \ldots$ be the valuations of image participants in θ'. Note that for any i we have $v_i \leq v_i'$ and $V_i \leq V_i'$ as we have onemore participant or a higher valuation in θ'. Let x be the new added participant or the participant which has higher valuation in θ'.

We prove this lemma by considering the value of $\text{REVENUE}(\text{MITA}, \theta)$ for the case when text participants win and the case when an image participant wins. If an image participant wins, then it means that $V_1 > \max_{j\in[k]} j \cdot v_j$ and she pays $\max(V_2, \max_{j\in[k]} j \cdot v_j)$ which is the total revenue.

If text participants win, then it means $V_1 \leq \max_{j\in[k]} j \cdot v_j$ and there are j^* winners where each of them pays $\max(v_{k+1}, \frac{V_1}{j^*})$. If $\max(v_{k+1}, \frac{V_1}{j^*}) = \frac{V_1}{j^*}$, then the total revenue is V_1. If $\max(v_{k+1}, \frac{V_1}{j^*}) = v_{k+1}$, it implies that $k \cdot v_{k+1}$ is larger than V_1. Remember that $C_\theta = \{j \in [k] | j \cdot v_j \geq V_1\}$ and $j^* = \max_{j\in C_\theta} j$; therefore $j^* = k$ and hence the total payment of the winners is $k \cdot v_{k+1}$.

In summary the total revenue for type profile θ is the following:

$$\text{REVENUE}(\text{MITA}, \theta) = \begin{cases} \max(V_2, \max_{j\in[k]} j \cdot v_j) & V_1 > \max_{j\in[k]} j \cdot v_j \ (A) \\ \max(V_1, k \cdot v_{k+1}) & V_1 \leq \max_{j\in[k]} j \cdot v_j \ (B) \end{cases}$$

Similarly the total revenue for type profile θ' is the following:

$$\text{REVENUE}(\text{MITA}, \theta') = \begin{cases} \max(V_2', \max_{j\in[k]} j \cdot v_j') & V_1' > \max_{j\in[k]} j \cdot v_j' \ (A) \\ \max(V_1', k \cdot v_{k+1}') & V_1' \leq \max_{j\in[k]} j \cdot v_j' \ (B) \end{cases}$$

Note that because for any i we have $v_i \leq v'_i$ and $V_i \leq V'_i$ the following inequalities are straightforward:

$$V_1 \leq V'_1 \tag{1}$$

$$V_2 \leq V'_2 \tag{2}$$

$$\max_{j\in[k]} j \cdot v_j \leq \max_{j\in[k]} j \cdot v'_j \tag{3}$$

$$k \cdot v_{k+1} \leq k \cdot v'_{k+1} \tag{4}$$

If both REVENUE(MITA, θ) and REVENUE(MITA, θ') take their value from Case (A), then the proof of the lemma follows from Eqs. (2) and (3). Similarly if both REVENUE(MITA, θ) and REVENUE(MITA, θ') take their value from Case (B), then the proof of the lemma follows from Eqs. (1) and (4).

If REVENUE(MITA, θ) takes its value from Case (A) and REVENUE(MITA, θ') takes from Case (B), then it means that participant x is a text participant which causes $\max_{j\in[k]} j \cdot v'_j$ to be larger than V'_1. The following proves the theorem for this case:

REVENUE(MITA, θ)	
$= \max(V_2, \max_{j\in[k]} j \cdot v_j)$	
$< V_1$	REVENUE(MITA, θ) takes its value from Case (A)
$= V'_1$	participant x is a text-participant
$\leq \max(V'_1, k \cdot v'_{k+1})$	
$=$ REVENUE(MITA, θ')	

If REVENUE(MITA, θ) takes its value from Case (B) and REVENUE(MITA, θ') takes from Case (A), then it means that participant x is an image participant. The following proves the theorem for this case:

REVENUE(MITA, θ)	
$= \max(V_1, k \cdot v_{k+1})$	
$< \max_{j\in[k]} j \cdot v_j$	REVENUE(MITA, θ) takes its value from Case (B) and the fact that $v_k \geq v_{k+1}$
$= \max_{j\in[k]} j \cdot v'_j$	x is an image participant
$\leq \max(V'_2, \max_{j\in[k]} j \cdot v'_j)$	
$=$ REVENUE(MITA, θ')	

□

In the above we proved that MITA satisfies IR, IC, and RM. In the following theorem we bound the PoRM of MITA and finish this section.

Theorem 4 $\text{PoRM}(\text{MITA}) \leq \ln k$.

Proof Let A be the set of winner(s) which realizes the maximum social welfare in type profile θ. If A contains only one image participant with valuation V_1, then we also have $V_1 \geq \max_{j\in[k]} j \cdot v_j$. Mechanism MITA also selects an image participant with the same valuation if $V_1 > \max_{j\in[k]} j \cdot v_j$ and hence PoRM(MITA) is 1. Otherwise we have $V_1 = \max_{j\in[k]} j \cdot v_j$ where MITA selects a set of text participants which overall gives social welfare V_1 and hence again the PoRM(MITA) is 1.

Now we consider the case when A contains text participants. By adding enough dummy participants with value zero, and without loss of generality, we assume that set A contains the first k text participants with highest valuations $v_1 \geq v_2 \geq \ldots \geq v_k$. Mechanism MITA selects either the first j^* text participants with highest valuations ($v_1 \geq v_2 \geq \ldots \geq v_{j^*}$) or selects an image participant with valuation V_1. Remember that j^* is the greatest number in set $C_\theta = \{j \,|\, j \in [k] \wedge j \cdot v_j \geq V_1\}$ which implies the following:

$$\forall j' \in \{j^*+1, \ldots, k\} \quad v_{j'} < \frac{V_1}{j'} \tag{5}$$

Note that if MITA selects an image participant, then Eq. (5) holds for $j^* = 0$.

Now we consider the following two cases to prove the theorem:

If MITA selects an image participant, then we have the following:

$$\begin{aligned}\text{PoRM}(\text{MITA}) &= \frac{\sum_{j\in[k]} v_j}{V_1} \\ &\leq \frac{\sum_{j\in[k]} V_1/j}{V_1} \qquad \text{Eq. (5)} \\ &\leq \ln k\end{aligned}$$

If MITA selects the first j^* text participants, then we have the following:

$$\begin{aligned}\text{PoRM}(\text{MITA}) &= \frac{\sum_{j\in[k]} v_j}{\sum_{j\in[j^*]} v_j} \\ &\leq \frac{\sum_{j\in[j^*]} v_j + \sum_{j=j^*+1}^{k} v_j}{\sum_{j\in[j^*]} v_j} \\ &\leq \frac{\sum_{j\in[j^*]} v_j + \sum_{j=j^*+1}^{k} V_1/j}{\sum_{j\in[j^*]} v_j} \\ &\qquad \text{Eq. (5)} \\ &\leq \frac{\sum_{j\in[j^*]} v_j + \sum_{j=j^*+1}^{k} \left(\sum_{j\in[j^*]} v_j\right)/j}{\sum_{j\in[j^*]} v_j} \\ &\qquad \text{because } V_1 \leq \sum_{j\in[j^*]} v_j \\ &\leq \ln k\end{aligned}$$

□

Video-Pod Auctions

In this section we design a mechanism for video-pod auction (MVPA) which satisfies IR, IC, and RM whose PoRM is at most $(\lfloor \log k \rfloor + 1) \cdot (2 + \ln k)$. Note that all the log functions are in base 2. Let $\theta = ((d_1, v_1), \ldots, (d_n, v_n)) \in \Theta^n$ be an arbitrary-type profile of n participants. We define the allocation and payment function of MVPA for this type profile.

Mechanism MVPA partitions the participants into $\lfloor \log k \rfloor + 1$ groups $G^{(1)}, \ldots, G^{(\lfloor \log k \rfloor + 1)}$ where group $G^{(g)}$ contains all the participants whose demand is in the range $[2^{g-1}, 2^g)$. Mechanism MVPA selects winners only from one group $G^{(g)}$.

Definition 5 Let $M^{(g)}$ be equal to $\max\left(\lfloor \frac{k}{2^g} \rfloor, 1\right)$ which is the maximum number of winners MVPA selects from group $G^{(g)}$.

Note that we can select at least $\lfloor \frac{k}{2^g} \rfloor$ winners from $G^{(g)}$ since there are k items and the demand of each participant is at most 2^g. Moreover, from the last group $G^{(\lfloor \log k \rfloor + 1)}$ we can select at least one winner although $\lfloor \frac{k}{2^{(\lfloor \log k \rfloor + 1)}} \rfloor = 0$, since we assume the demand of all the participants is from set $[k]$.

Let $\left(d_1^{(g)}, v_1^{(g)}\right), \ldots, \left(d_p^{(g)}, v_p^{(g)}\right)$ be the types of all the participants in group g where $p = |G^{(g)}|$. Here by adding enough dummy participants, we assume p is always larger than $M^{(g)}$. Also, without loss of generality we assume $v_1^{(g)} \geq v_2^{(g)} \geq \ldots \geq v_p^{(g)}$. We define the max possible revenue of group g (MPRG(g)) to be the following:

$$\text{MPRG}(g) = \max_{j \in [M^{(g)}]} j \cdot v_j^{(g)}$$

As the name MPRG suggests, we will see that its value captures the maximum revenue can be truthfully obtained from group g. Let $G^{(g^*)}$ be a group with the maximum MPRG and $G^{(g')}$ be a group with the second maximum MPRG breaking the ties arbitrarily.

The set of winners selected by MVPA is

$$\left\{\left(d_1^{(g^*)}, v_1^{(g^*)}\right), \ldots, \left(d_j^{(g^*)}, v_j^{(g^*)}\right)\right\}$$

where j is the largest number in $[M^{(g^*)}]$ for which $j \cdot v_j^{(g^*)}$ is larger than or equal to $\text{MPRG}(g')$. In other words, the number of winners (j) is the largest number in $[M^{(g^*)}]$ for which $j \cdot v_j^{(g^*)} \geq \text{MPRG}(g')$.

Now we use Lemma 1 to show that MVPA is truthful and obtain the payments of winners.

Observation 2 *Allocation function x^{MVPA} is weakly monotone.*

Proof Note that MVPA sorts the participants according to their valuation and selects the first j participants. Therefore if any participant i increases its valuation, it only helps her to enter the winning set. Hence, the observation follows: □

In the rest of this section, we drop the group identifier of $M^{(g^*)}$ and simply use M unless it is about another group.

In the following lemma we find the critical value of each winner i which is actually equal to its payment $\left(p_i^{\text{MVPA}}\right)$.

Lemma 4 *Let set of winners $x^{\text{MVPA}}(\theta)$ contain the first j participants with highest valuations from $G^{(g^*)}$ and $v_{M+1}^{(g^*)}$ be the $(M+1)$th highest valuation in group $G^{(g^*)}$ which is zero if it does not exist. Then, the payment of participant i is the following:*

$$p_i^{\text{MVPA}}(\theta) = \begin{cases} \max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right) & i \in x^{\text{MVPA}}(\theta) \\ 0 & i \notin x^{\text{MVPA}}(\theta) \end{cases}$$

R

Proof If participant i is not a winner, then its payment is zero. When participant i is a winner, then we prove that its payment is equal to its critical value (Definition 4). In order to prove that value $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right)$ is the critical value of participant i, we show that for any value larger than $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right)$ participant i still wins and for any value less than it she loses.

Remember that $v_1^{(g^*)} \geq v_2^{(g^*)} \geq \ldots \geq v_p^{(g^*)}$ are the valuations of participants in group $G^{(g^*)}$ and $v_1^{(g^*)}, v_2^{(g^*)}, \ldots, v_j^{(g^*)}$ are the valuations of the winners. Because group $G^{(g^*)}$ is the group with the maximum MPRG, we have $v_j^{(g^*)} \geq \frac{\text{MPRG}(g')}{j}$. As there can be at most M winners from group $G^{(g^*)}$, we have $v_j^{(g^*)} \geq v_{M+1}^{(g^*)}$. Therefore we have

$$v_j^{(g^*)} \geq \max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right). \quad (6)$$

Let participant i with type profile $\left(d_i^{(g^*)}, v_i^{(g^*)}\right)$ be the ith winner in group g^* where $i \in [j]$. We show that for any valuation greater than or equal to $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right)$ participant i remains in the winning set. Equation (6) implies that there are j participants in group $G^{(g^*)}$ whose valuations are larger than $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right)$. If we decrease the valuation of participant i to $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right)$, we still have j participants in group $G^{(g^*)}$ with valuations at least $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right)$. Therefore, the value $\text{MPRG}(g^*)$ will be at least $\text{MPRG}(g')$ and group $G^{(g^*)}$ remains the winning group: hence participant i remains in the winning set.

Now we prove that if the valuation of participant i is less than the $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right)$, she cannot be in the winning set. In order to prove this, we consider two cases: (A) when $\max(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)})$ is equal to $v_{M+1}^{(g^*)}$ and (B) when $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right)$ is equal to $\frac{\text{MPRG}(g')}{j}$.

Case (A): If $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right) = v_{M+1}^{(g^*)}$ and the valuation of participant i is less than $v_{M+1}^{(g^*)}$, then it means that there are M participants who have valuations greater than the valuation of participant i. As there can be at most M winners from group $G^{(g^*)}$, participant i cannot be a winner.

Case (B): We prove this case by contradiction. Suppose $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right) = \frac{\text{MPRG}(g')}{j}$ and $\theta' = \left((d_i^{(g^*)}, v_i^{(g^*)'}), \theta_{-i}\right)$ be a type profile in which the valuation of participant i is less than $\frac{\text{MPRG}(g')}{j}$ while she is still winner. Because the valuation of participant i $\left(v_i^{(g^*)'}\right)$ is less than $\frac{\text{MPRG}(g')}{j}$ and i is in the winning set, in order for $\text{MPRG}(g^*)$ to be larger than $\text{MPRG}(g')$, there has to be more than j winners. Let $j' > j$ be the number of winners in θ'. Having j' winners in θ' and in order for $G^{(g^*)}$ to be the group with the highest MPRG, we conclude that there are j' participants with valuation greater than $\frac{\text{MPRG}(g')}{j'}$. Note that the only difference between θ and θ' is that the valuation of participant i is higher in θ. Therefore, there are also at least j' participants with valuation greater than $\frac{\text{MPRG}(g')}{j'}$ in θ. This contradicts with the way we select the number of winners (j) in θ which is the maximum number for which $j \cdot v_j^{(g^*)}$ is larger than $\text{MPRG}(g')$. □

The allocation function x^{MVPA} is weakly monotone (Observation 2) and the payments of the winners are their critical values (Lemma 4); therefore by Lemma 1 we conclude that MVPA satisfies IC.

In the rest of this section, first we prove that MVPA satisfies RM and then bounds its PoRM.

Proposition 1 *The total revenue of mechanism* MVPA *for type profile* θ *(*$\text{REVENUE}(\text{MVPA}, \theta)$*) is the following:*

$$\text{REVENUE}(\text{MVPA}, \theta) = \max\left(\text{MPRG}(g'), M \cdot v_{M+1}^{(g^*)}\right)$$

where g' is a group with the second highest MPRG.

Proof From Lemma 4 we know that there are j winners and each of them pays $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right)$. Therefore the sum of payments or the revenue of MVPA is $j \cdot \max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right)$. The proof of the proposition follows if we show that when $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right)$ is equal to $v_{M+1}^{(g^*)}$, then the number of winners (j) is equal to M.

If $\max\left(\frac{\text{MPRG}(g')}{j}, v_{M+1}^{(g^*)}\right)$ is equal to $v_{M+1}^{(g^*)}$, then as $v_{M+1}^{(g^*)} \leq v_M^{(g^*)}$, we have $M \cdot v_M^{(g^*)} \geq \frac{\text{MPRG}(g')}{j}$. Remember that j is the maximum number in the set $[M]$ for which $j \cdot v_j^{(g^*)}$ is larger than $\text{MPRG}(g')$. Therefore j is equal to M. □

Lemma 5 *Let θ' be the type profile obtained by either adding a new participant or increasing the valuation of a participant in θ. Then,*

$$\text{REVENUE}(\text{MVPA}, \theta') \geq \text{REVENUE}(\text{MVPA}, \theta).$$

Proof Let x be the new added participant or the participant which has the increased valuation in θ'. Throughout the proof we show MPRG of each group g in type profile θ by $\text{MPRG}_\theta(g)$ and in type profile θ' by $\text{MPRG}_{\theta'}(g)$. Similarly, we show the jth highest valuation of the participants of group g by $v_j^{(g^*,\theta)}$ in type profile θ and by $v_j^{(g^*,\theta')}$ in type profile θ'.

As the jth highest valuation of the participants of each group can only increase by adding participant x, we conclude

$$\forall g, \forall j \quad v_j^{(g,\theta')} \geq v_j^{(g,\theta)}. \tag{7}$$

Remember that MPRG_θ of each group g is $\max_{j \in [M^{(g)}]} j \cdot v_j^{(g,\theta)}$ and using Eq. (7) we get

$$\forall g \quad \text{MPRG}_{\theta'}(g) \geq \text{MPRG}_\theta(g). \tag{8}$$

In order to prove this lemma we consider two cases: (A) adding participant x does not change the winning group $G^{(g^*)}$ and (B) adding x changes the winning group.

Case (A): Let g'' be a group with the second highest MPRG in θ'; it is possible that g' is equal to g''.

$$\begin{aligned}
\text{REVENUE}(\text{MVPA}, \theta') &= \max\left(\text{MPRG}_{\theta'}(g''), M \cdot v_{M+1}^{(g^*,\theta')}\right) \\
&\quad \text{Proposition 1} \\
&\geq \max\left(\text{MPRG}_{\theta'}(g'), M \cdot v_{M+1}^{(g^*,\theta')}\right) \\
&\quad \text{definition of } g'' \\
&\geq \max\left(\text{MPRG}_{\theta}(g'), M \cdot v_{M+1}^{(g^*,\theta)}\right) \\
&\quad \text{Eqs. (7) and (8)} \\
&= \text{REVENUE}(\text{MVPA}, \theta)
\end{aligned}$$

Case (B): Let $G^{(g'')}$ be a group with the highest MPRG in θ'. We have

$$\text{MPRG}_\theta(g^*) \geq \text{MPRG}_\theta(g') \tag{9}$$

as g^* has the highest and g' has the second highest MPRG in θ.

$$\begin{aligned} \text{MPRG}_\theta(g^*) &\geq M \cdot v_M^{(g^*,\theta)} && \text{As } \text{MPRG}_\theta(g^*) = \max_{j\in[M]} j \cdot v_j^{(g^*,\theta)} \\ &\geq M \cdot v_{M+1}^{(g^*,\theta)} && \text{As } v_M^{(g^*,\theta)} \geq v_{M+1}^{(g^*,\theta)} \end{aligned} \tag{10}$$

Let $\hat{g}$ be the group with second highest MPRG in θ'. Because g^* is no longer the winning group in θ', it can be a candidate for the group with the second highest MPRG in θ' and hence we have the following:

$$\text{MPRG}_\theta(g^*) \leq \text{MPRG}_{\theta'}(g^*) \leq \text{MPRG}_{\theta'}(\hat{g}) \tag{11}$$

The following equations conclude the proof of this case:

$$\begin{aligned} \text{REVENUE}(\text{MVPA}, \theta) &= \max\left(\text{MPRG}_\theta(g'), M \cdot v_{M+1}^{(g^*,\theta)}\right) \\ &\leq \text{MPRG}_\theta(g^*) \\ &\quad \text{by Eqs. (9) and (10)} \\ &\leq \text{MPRG}_{\theta'}(\hat{g}) \\ &\quad \text{by Eq. (11)} \\ &\leq \max\left(\text{MPRG}_{\theta'}(\hat{g}), M^{(g'')} \cdot v_{M^{(g'')}+1}^{(g'',\theta)}\right) \\ &= \text{REVENUE}(\text{MVPA}, \theta') \end{aligned}$$

□

The following lemma which bounds PoRM of MVPA finishes this section:

Theorem 5 $\text{PoRM}(\text{MITA}) \leq (\lfloor \log k \rfloor + 1) \cdot (2 + \ln k)$

Proof Let $\text{WF}(g)$ to be the maximum social welfare achievable if we select the winners only from group $G^{(g)}$. Let A be a set of winner(s) which realizes the maximum welfare in type profile θ. Note that as there are $\lfloor \log k \rfloor + 1$ groups, one group $(\hat{g})$ has a subset of participants from A whose social welfare is at least $\frac{\text{WF}(A)}{\lfloor \log k \rfloor + 1}$ and hence the following:

$$\text{WF}(\hat{g}) \geq \frac{\text{WF}(A)}{\lfloor \log k \rfloor + 1} \tag{12}$$

Now we prove the following claim about $\text{MPRG}(\hat{g})$:

Claim 1 $\text{MPRG}(\hat{g}) \geq \frac{\text{WF}(\hat{g})}{2+\ln k}$

Proof Let B be the set of participants from group $G^{(\hat{g})}$ which give the maximum social welfare. Because the demands of all the participants of $G^{(\hat{g})}$ are in range $[2^{\hat{g}-1}, 2^{\hat{g}})$, size of B is at most $\lfloor k/2^{\hat{g}-1} \rfloor$. Remember from Definition 5 that $M^{(\hat{g})} = \max\left(\lfloor k/2^{(\hat{g})} \rfloor, 1\right)$ is the maximum number of winners that MVPA potentially selects from group $G^{(\hat{g})}$. Therefore, we have $|B| \leq 2 \cdot M^{(\hat{g})} + 1$.

Throughout the proof, we drop the superscript from $M^{(\hat{g})}$ and simply refer to it as M.

Let $v_1 \geq v_2 \geq \ldots \geq v_{2\cdot M+1}$ be the valuations of the participants in B; if B has less than $2\cdot M+1$ participants, we add enough dummy participants with valuations zero. Remember that $\text{MPRG}(\hat{g}) = \max_{j\in[M]} j \cdot v_j^{(\hat{g})}$ where M is at least 1 (see Definition 5) which implies

$$v_i \leq \frac{\text{MPRG}(\hat{g})}{i} \quad \forall i \in [M] \tag{13}$$

The following equations conclude the proof of the claim:

$$\begin{aligned}
\text{WF}(\hat{g}) &= \sum_{i=1}^{2\cdot M+1} v_i \\
&= \sum_{i=1}^{M} v_i + \sum_{i=M+1}^{2\cdot M+1} v_i \\
&\le \sum_{i=1}^{M} v_i + \sum_{i=M+1}^{2\cdot M+1} v_M \\
&\quad \text{replacing } v_i \text{ with } v_M \text{ for } i > M \\
&\le \sum_{i=1}^{M} \frac{\text{MPRG}(\hat{g})}{i} + \sum_{i=M+1}^{2\cdot M+1} \frac{\text{MPRG}(\hat{g})}{M} \\
&\quad \text{by Eq. (13)} \\
&\le (2 + \ln k)\text{MPRG}(\hat{g})
\end{aligned}$$

□

Remember $G^{(g^*)}$ is the group with maximum MPRG value. Let j be the number for which $\text{MPRG}(g^*)$ is equal to $j \cdot v_j^{(g^*)}$. Allocation function x^{MVPA} selects the first j^* participants from group $G^{(g^*)}$ where j^* is the maximum number for which $j^* \cdot v_{j^*}^{(g^*)}$ is larger than $\text{MPRG}(g')$. Therefore we can conclude that $j \le j^*$ and hence

$$\text{WF}\left(x^{\text{MVPA}}(\theta)\right) \ge \text{MPRG}(g^*). \tag{14}$$

The following equations conclude the proof of the theorem:

$$\begin{aligned}
\text{WF}\left(x^{\text{MVPA}}(\theta)\right) &\ge \text{MPRG}(g^*) \\
&\quad \text{by Eq. (14)} \\
&\ge \text{MPRG}(\hat{g}) \\
&\quad G^{(g^*)} \text{ has the highest MPRG} \\
&\ge \frac{\text{WF}(\hat{g})}{2 + \ln k} \\
&\quad \text{Claim 1} \\
&\ge \frac{\text{WF}(A)}{(\lfloor \log k \rfloor + 1) \cdot (2 + \ln k)} \\
&\quad \text{by Eq. (12)}
\end{aligned}$$

□

Lower Bound

In this section we prove Theorem 2. As mentioned earlier we need two additional mild assumptions of *anonymity* and *independence of irrelevant alternatives* (which we define below) on the class of mechanisms for which we prove our lower bound.

Definition 6 A mechanism $(\mathcal{M} = (x, p))$ is anonymous (AM) if the following holds: Suppose $\theta_1, \theta_2 \in \Theta^n$ are two type profiles which are permutations of each other (i.e., the set of type profiles are same just that the identities of participants to whom those types belongs are different). Say, $\theta_2 = \pi(\theta_1)$. Also say $x(\theta_1) = S_1$ and $x(\theta_2) = S_2$. Then $S_2 = \pi(S_1)$.

Definition 7 Let $\theta \in \Theta^n$ be an arbitrary-type profile and $i \in N$ be an arbitrary participant with type $\theta_i = (d_i, v_i)$. A mechanism $(\mathcal{M} = (x, p))$ satisfies independence of irrelevant alternatives (IIA) that if we decrease the bid of a losing participant, say, participant i, to $\hat{v}_i < v_i$, then the new set of winners is a super set of the previous one, i.e., $x(\theta) \subseteq x((d_i, \hat{v}_i), \theta_{-i})$. In other words, decreasing the bid of a losing participant does not hurt any winner.

The proof outline of Theorem 2 is the following. Let $\mathcal{M}^* = (x^*, p^*)$ be a mechanism which satisfies all the five properties and has the optimal PoRM OPT (i.e., $\text{OPT} = \text{PoRM}(\mathcal{M}^*)$). We study the behavior of $\mathcal{M}^*$ in a few type profiles. Let ϵ be an arbitrary small positive real value. First we show that when there are only two participants with types $(k, 1)$ and $(k, 1 + \epsilon)$, $\mathcal{M}^*$ gives all the k items to the participant with type $(k, 1 + \epsilon)$. The revenue of $\mathcal{M}^*$ from these two participants is 1. Then, we add k

more participants to create type profile $\theta = \big((1, 1-\epsilon), (1, \frac{1}{2}-\epsilon), \ldots, (1, \frac{1}{k}-\epsilon), (k, 1), (k, 1+\epsilon)\big)$. The RM property requires $\mathcal{M}^*$ to make at least the same revenue for θ. From this constraint we are able to show that $\mathcal{M}^*$ assigns all the items to participant $k+2$ with type $(k, 1+\epsilon)$ and hence gets social welfare $1+\epsilon$. Note that the maximum social welfare happens when the set of winners is $\{1, \ldots, k\}$ which implies $\text{WFR}(\mathcal{M}^*, \theta) \geq \sum_{i=1}^{k} \frac{1}{i} - k \cdot \epsilon$ (see Definition 1). Because $\text{PoRM}(\mathcal{M}^*) \geq \text{WFR}(\mathcal{M}^*, \theta)$ for any $\theta \in \Theta^n$, we conclude that $\text{OPT} \geq \sum_{i=1}^{k} \frac{1}{i}$.

First we study the behavior of $\mathcal{M}^*$ when we have only two participants with types $(k, 1)$ and $(k, 1+\epsilon)$.

Lemma 6 *Mechanism $\mathcal{M}^*$ in type profile $((k, 1), (k, 1+\epsilon))$ gives all k items to the second participant and make one unit of revenue, i.e., $x^*\left((k, 1), (k, 1+\epsilon)\right) = \{2\}$ and $p^*\left((k, 1), (k, 1+\epsilon)\right) = (0, 1)$.*

Proof First we study type profile $((k, v_1), (k, v_2))$ for general values $v_1, v_2 \in \mathbb{R}^+$ where $v_1 < v_2$. We prove that $\mathcal{M}^*$ gives all the items to the second participant.

Claim 2 $x^*\left((k, v_1), (k, v_2)\right) = \{2\}$ for any $v_1, v_2 \in \mathbb{R}^+$ where $v_1 < v_2$.

Proof First note that M^* has to have a winner for this type profile because otherwise its social welfare will be zero while the maximum social welfare is v_2. This makes the social welfare ratio of $\mathcal{M}^*$ to be undefined.

Now we prove that if $x^*\left((k, v_1), (k, v_2)\right) = \{1\}$, then $\mathcal{M}^*$ either violates IC or AM. Let call type profile $((k, v_1), (k, v_2))$ by $\theta^{(1)}$ and suppose for the sake of contradiction $x^*(\theta^{(1)}) = \{1\}$. From Lemma 1 we know that if participant 1 increases his bid to v_2, she still wins; hence $x^*(\theta^{(2)}) = \{1\}$ where $\theta^{(2)} = ((k, v_2), (k, v_2))$. Now if in type profile $\theta^{(2)}$ participant 2 decrease his bid to v_1, again from Lemma 1 we conclude that she cannot win, i.e., $x^*(\theta^{(3)}) = \{1\}$ where $\theta^{(3)} = ((k, v_2), (k, v_1))$. Type profile $\theta^{(1)}$ is $\theta^{(3)}$ with participant 1 swapped with participant 2 but in both of them the first participant wins which contradicts with AM. □

Claim 2 directly proves that the winner in type profile $((k, 1), (k, 1+\epsilon))$ is the second participant. The only thing remains is to show that her payment (p_2) is 1. Note that payment p_2 cannot be less than one because otherwise by Lemma 1 participant 2 wins all the items in type profile $((k, 1), (k, p_2))$ which contradicts with Claim 2. Payment p_2 cannot be larger than one because otherwise for any value $1 < v_2 < p_2$ participant 2 wins all the items in type profile $((k, 1), (k, v_2))$. This contradicts with Lemma 1 which states that the payment p_2 is the smallest value for which participant 2 wins the items. □

Now we add k more participants, each of which wants only one item. In the following lemma we prove that RM forces $\mathcal{M}^*$ to assign all of the items to one of the participants who want all the items.

Lemma 7 *For the set of $k+2$ participants with type profile $\theta^{(0)} = \big((1, 1-\epsilon), (1, \frac{1}{2}-\epsilon), \ldots, (1, \frac{1}{k}-\epsilon), (k, 1), (k, 1+\epsilon)\big)$, mechanism $\mathcal{M}^*$ assigns all the k items to either participant $k+1$ or participant $k+2$, i.e., $x^*\left(\theta^{(0)}\right) = \{k+1\}$ or $x^*\left(\theta^{(0)}\right) = \{k+2\}$.*

Proof We prove the lemma by contradiction that if $\mathcal{M}^*$ assigns the items to a subset of the first k participants, it satisfies RM. We consider a class of k type profiles $(\theta^{(1)}, \ldots, \theta^{(k)})$ where $\theta^{(i)}$ is built from $\theta^{(i-1)}$. The only possible difference between $\theta^{(i)}$ and $\theta^{(i-1)}$ is in the valuation of participant i. If participant i is a winner in $\theta^{(i-1)}$, then we obtain $\theta^{(i)}$ by increasing the valuation of the ith participant from $\frac{1}{i} - \epsilon$ to $1 - \epsilon$. Note that the payment of participant i in $\theta^{(i-1)}$ is at most her valuation which is $\frac{1}{i} - \epsilon$ and in $\theta^{(i)}$ it remains the same by Lemma 1. If participant i is not a winner in $\theta^{(i-1)}$, then we obtain $\theta^{(i)}$ by decreasing his valuation to zero. Note that by IIA, no winner turns to a loser in $\theta^{(i)}$.

Let $j \in \{1, \ldots, k\}$ be the largest number for which participant j is a winner in $\theta^{(j-1)}$ and we increase his valuation to $1-\epsilon$ in $\theta^{(j)}$. Note that at the start in type profile $\theta^{(0)}$, the set of winners is a nonempty subset of $\{1, \ldots, k\}$. Therefore there is at least one such j for which participant j is a

winner in $\theta^{(j)}$ since decreasing the non-winners valuation does not reduce the size of the winners.

Now we prove that there is no winner in the set of participants $\{j+1,\ldots,k\}$ in type profile $\theta^{(j)}$. Assume otherwise and let $p \in \{j+1,\ldots,k\}$ be the smallest number for which participant p is a winner in $\theta^{(j)}$. Note that when we decrease the valuation of each participant $j < p' < p$ to zero to obtain $\theta^{(p')}$, participant p remains as a winner in all of them by IIA. Therefore, participant p is a winner in type profile $\theta^{(p-1)}$ and we increase his valuation in $\theta^{(p)}$ which contradicts with the fact that j is the largest number for which participant j is a winner in $\theta^{(j-1)}$.

The payment of participant j in $\theta^{(j-1)}$ is at most its valuation which is $\frac{1}{j}-\epsilon$. When we increase his bid to $1-\epsilon$ in type profile $\theta^{(j)}$, its payment remains the same by Lemma 1. Note that by construction of $\theta^{(j)}$, the valuation of all participants in $\{1,\ldots,j\}$ is either zero or $1-\epsilon$. If the valuation of them is $1-\epsilon$ and they are winner, by AM their payment is $\frac{1}{j}-\epsilon$. Therefore the total payments or revenue of $\mathcal{M}^*$ in $\theta^{(j)}$ is at most $j\cdot(\frac{1}{j}-\epsilon) = 1-j\cdot\epsilon$ since there is no other winner in set of participants $\{j+1,\ldots,k\}$ in type profile $\theta^{(j)}$.

Note that type profile $\theta^{(j)}$ is obtained from type profile $((k,1),(k,1+\epsilon))$ by adding k more participants. However the revenue of $\theta^{(j)}$ is $1-j\cdot\epsilon$ that is strictly less than 1 which is the revenue of $((k,1),(k,1+\epsilon))$ by Lemma 6. This contradicts with the RM property of $\mathcal{M}^*$; hence $\mathcal{M}^*$ has to assign the items to either participant $k+1$ or $k+2$. □

Now we show how from Lemma 7 we can derive Theorem 2. Note that the maximum welfare for type profile $\theta^{(0)} = \big((1,1-\epsilon),(1,\frac{1}{2}-\epsilon),\ldots,(1,\frac{1}{k}-\epsilon),(k,1),(k,1+\epsilon)\big)$ realized when we give one item to each of the first k participants for which we get the total social welfare $\sum_{i=1}^{k}\frac{1}{i}-k\cdot\epsilon$, i.e., the nominator of Definition 1 for this type profile is $\sum_{i=1}^{k}\frac{1}{i}-k\cdot\epsilon$. The denominator of Definition 1 is at most $1+\epsilon$ by Lemma 7. Therefore the ratio of the welfare for this type profile is at least $\frac{\sum_{i=1}^{k}1/i-k\cdot\epsilon}{1+\epsilon}$. Because OPT is the maximum ratio over all type profiles (see Definition 2), we have $\text{OPT} \geq \frac{\sum_{i=1}^{k}1/i-k\cdot\epsilon}{1+\epsilon}$ which results in $\text{OPT} \geq \sum_{i=1}^{k}\frac{1}{i}-\epsilon'$ where $\epsilon' = \frac{\epsilon(k-\sum_{i=1}^{k}1/i)}{1+\epsilon}$.

Note that the value ϵ' can be made arbitrarily small by selecting a sufficiently small value for ϵ. Therefore we prove that for any positive small real value ϵ', we have $\text{OPT} \geq \sum_{i=1}^{k}\frac{1}{i}-\epsilon'$ which implies Theorem 2.

Acknowledgments The first author would like to thank Vasilis Gkatzelis for some initial fruitful discussions on this topic.

Recommended Reading

1. Ausubel LM, Milgrom P (2002) Ascending auctions with package bidding. Front Theor Econ 1(1):1–42
2. Bikhchandani S, Chatterji S, Lavi R, Mu'alem A, Nisan N, Sen A (2006) Weak monotonicity characterizes deterministic dominant-strategy implementation. Econometrica 74(4):1109–1132
3. Day R, Milgrom P (2008) Core-selecting package auctions. Int J Game Theory 36(3–4):393–407
4. Goldberg AV, Hartline JD, Karlin AR, Saks M, Wright A (2006) Competitive auctions. Games Econ Behav 55(2):242–269
5. Goldberg AV, Hartline JD, Wright A (2001) Competitive auctions and digital goods. In: Symposium on discrete algorithms, Washington, DC, pp 735–744
6. Lavi R, Mu'Alem A, Nisan N (2003) Towards a characterization of truthful combinatorial auctions. In: Foundations of computer science, Cambridge, pp 574–583
7. Myerson RB (1981) Optimal auction design. Math Oper Res 6(1):58–73
8. Nisan N, Roughgarden T, Tardos E, Vazirani VV (2007) Algorithmic game theory. Cambridge University Press, Cambridge
9. Rastegari B, Condon A, Leyton-Brown K (2009) Stepwise randomized combinatorial auctions achieve revenue monotonicity. In: Symposium on discrete algorithms, New York, pp 738–747
10. Rastegari B, Condon A, Leyton-Brown K (2011) Revenue monotonicity in deterministic, dominant-strategy combinatorial auctions. Artif Intell 175(2):441–456
11. Roberts K (1979) The characterization of implementable choice rules. Aggreg Revel Prefer 12(2):321–348
12. Rochet J-C (1987) A necessary and sufficient condition for rationalizability in a quasi-linear context. J Math Econ 16(2):191–200
13. Saks M, Yu L (2005) Weak monotonicity suffices for truthfulness on convex domains. In: Electronic commerce, Vancouver, pp 286–293

Reverse Search; Enumeration Algorithms

Masashi Kiyomi
International College of Arts and Sciences, Yokohama City University, Yokohama, Kanagawa, Japan

Keywords

Enumeration; Reverse search; Vertices of a polytope

Years and Authors of Summarized Original Work

1996; Avis, Fukuda
2003; Uno
2004; Nakano, Uno

Problem Definition

We will consider enumeration problems, i.e., we want to list all the objects that satisfy given conditions (e.g., vertices of a polytope $\{\boldsymbol{x} \mid A\boldsymbol{x} \geq \boldsymbol{b}\}$ or maximal cliques in a given graph). One object should not be listed twice or more.

Introduction

In this entry, we consider an enumeration scheme called *reverse search* developed by Avis and Fukuda [1]. The scheme was originally developed to enumerate all the vertices of a given polytope represented by the intersection of half spaces [1]. The scheme is very powerful, and quite many kinds of objects such as arrangements in a hyperplane, triangulations of a polygon, bases of a matroid, spanning trees, trees, or maximal cliques in a graph, plane graphs of given number of vertices, etc., can be enumerated with it [1–4, 6].

Think of a problem to enumerate (or visit) all the vertices of a given connected graph G. Most of the readers may use depth-first search or breadth-first search algorithms. The two algorithms dynamically find tree structures in G and traverse them. Given an enumeration problem, reverse search scheme also finds some kind of tree structure on objects to enumerate dynamically and traverse it. To execute depth-first search or breadth-first search, a graph G should be given explicitly, and we have to remember which vertices have been visited. However, in most of enumeration problems, the objects that we want to enumerate are, of course, not explicitly given, nor we cannot remember all the objects that we have already output in the execution of an algorithm. For example, if we want to enumerate all the vertices of a polytope represented by the intersection of half spaces, the vertices are not given explicitly. If we want to enumerate plane graphs of 100 vertices, the number is quite large, and we do not want to remember every obtained graph. So, the scheme is designed to treat implicitly given objects and run with small amount of memory.

Key Results

When we develop an algorithm for enumerating some objects with reverse search scheme, we first think of an implicit connected graph G_{rs} of the objects. For example, when enumerating all the vertices of a given polytope, we think of a graph whose vertices correspond to the vertices of the polytope and whose edges correspond to the edges of the polytope. When enumerating spanning trees in a graph G, we think of a graph G_{rs} whose vertices correspond to the spanning trees of G, and $\{i, j\} \in E(G_{\mathrm{rs}})$ if and only if spanning tree T_j of G corresponding to vertex j of G_{rs} can be obtained from spanning tree T_i of G corresponding to vertex i of G_{rs} by removing an edge and adding an edge. Of course, we cannot make such a graph G_{rs} explicitly without enumerating the vertices of the polytope or the spanning trees of G. However, given an object x, we can easily generate every object whose corresponding vertex is adjacent to x's corresponding vertex in G_{rs}. To put it the other way around, we require G_{rs} this property. In the above examples, G_{rs} are undirected. However, G_{rs} is sometimes di-

rected. When enumerating (not necessarily maximal) cliques in a graph, $(i, j) \in E(G_{\mathrm{rs}})$ if and only if the clique corresponding to j is a proper subset of the clique corresponding to i.

Now we have a connected graph G_{rs} of objects to enumerate. Then for every vertex $v \in V(G_{\mathrm{rs}})$ except for a special vertex r called *root*, we define a *parent* vertex u of v such that u is adjacent to v, and no vertex of G_{rs} is a proper ancestor of itself, i.e., by iteratively moving from a vertex v to the parent of v, to the parent of the parent of v, and so on, we never come to the start vertex v again. Then we easily come to the following lemma.

Lemma 1 *For every vertex v of G_{rs}, v is a descendant of r.*

Proof Since every vertex of G_{rs} cannot be an ancestor of itself and the number of vertices in G_{rs} is bounded, every vertex of G_{rs} has its oldest ancestor. Since a vertex of G_{rs} except for r has its parent, it cannot be the oldest ancestor. Therefore, r is the oldest ancestor of every vertex of G_{rs}. □

By the lemma above, edges in G_{rs} corresponding to the "parent-child" relations clearly induce a spanning tree (or arborescence) T_{rs} of G_{rs}. Therefore, we can enumerate every object by traversing T_{rs} from r in the depth-first manner. The whole scheme is shown below.

```
Procedure ENUMERATE_SUBTREE(v)
  output v
  for all w satisfying (w, v) ∈ E(G_rs) do
    if v is the parent of w then
        ENUMERATE_SUBTREE(w)
    end if
  end for
end procedure
```

```
Procedure ENUMERATE
  find r
  ENUMERATE_SUBTREE(r)
end procedure
```

If the depth of T_{rs} is very deep, using a recursion needs a big amount of memory. However, since we can find the parent of each vertex of G_{rs}, we actually do not need to use a recursion. Even if we do not remember the previously visited vertices in G_{rs}, we can go back in the tree search by finding the parents. If the time complexity for finding the parent is relatively high, the total time complexity gets high. Therefore, there is a time-space trade-off.

Examples

For enumerating all the vertices of a given polytope $P = \{A\boldsymbol{x} \geq \boldsymbol{b}\}$, we use G_{rs} described in the previous section. For the sake of simplicity, we assume that P is not degenerated. First, we find a vertex $\boldsymbol{x}^*$ of P by the simplex method or the interior point method. Then, find an objective vector $\boldsymbol{c}$ such that the unique optimal solution of the linear programming problem $\min \boldsymbol{c}^\top \boldsymbol{x}$, s. t. $A\boldsymbol{x} \geq \boldsymbol{b}$ is $\boldsymbol{x}^*$. We define a parent vertex $\overline{\boldsymbol{x}}$ of a vertex $\boldsymbol{x}$ as the vertex corresponding to the basis of P obtained from the basis corresponding to $\boldsymbol{x}$ by a single pivot in the simplex method minimizing $\boldsymbol{c}^\top \boldsymbol{x}$ with Bland's pivot rule. The root vertex is $\boldsymbol{x}^*$. We can easily find every vertex $\boldsymbol{x}'$ satisfying $\{\boldsymbol{x}', \boldsymbol{x}\} \in E(G_{\mathrm{rs}})$ by swapping a basic variable and a nonbasic variable from the basis corresponding to $\boldsymbol{x}'$. Of course, we can easily check if vertex $\boldsymbol{x}$ is the parent of vertex $\boldsymbol{x}'$ by running the simplex method by one step from $\boldsymbol{x}'$.

For enumerating (not necessarily maximal) cliques in a graph G, we also use (the directed graph) G_{rs} in the previous section. We define the parent of vertex v corresponding to clique C_v in G as the vertex u corresponding to clique C_u such that C_u is obtained from C_v by removing the vertex of the smallest index. The root is the empty set. Since a vertex w satisfying $(w, v) \in E(G_{\mathrm{rs}})$ corresponds to a clique obtained by adding a vertex to C_v, we can find it easily. Clearly we can check if v is the parent of u easily, too.

Note that the algorithms introduced in this section are for an easy explanation. One can develop faster algorithms for the problems.

Avoiding Long Delays

A naive implementation of the reverse search scheme sometimes causes a long delay between successive outputs of two objects. Consider the

case that the depth of T_{rs} is very deep and one has to return from a leaf to the root. In order to avoid this kind of long delays, a smart method is known [5, 7]. At the odd level of recursion, we output the objects before making the recursive calls, and at the even level of the recursion, we output after the termination of the recursive calls. In this way, at least one of three iterations outputs an object when the algorithm ascends or descends the search tree T_{rs}. The algorithm is shown below.

```
Procedure ENUMERATE_SUBTREE(v, parity)
  if parity = 0 then
    output v
  end if
  for all w satisfying (w, v) ∈ E(G_rs) do
    if v is the parent of w then
        ENUMERATE_SUBTREE(w,parity ⊕ 1)
    end if
  end for
  if parity = 1 then
    output v
  end if
end procedure
```

```
Procedure ENUMERATE
  find r
  ENUMERATE_SUBTREE(r, 0)
end procedure
```

Note

For the sake of easy understanding, we introduced G_{rs}. However, most of results using the reverse search type algorithms do not treat G_{rs}. It is easy to understand that we can develop reverse search algorithms only by good definitions of parent-child relations and fast algorithms to enumerate children of given objects. If one can develop a fast children enumeration algorithm which enumerates all the children of an object in time T and if the degrees of some vertices in G_{rs} are quite large compared with T, the resulting enumeration algorithm is faster than the naive implementation described in this entry. Such examples will appear in other entries in this book.

Recommended Reading

1. Avis D, Fukuda K (1996) Reverse search for enumeration. Discret Appl Math 65:21–46
2. Makino K, Uno T (2004) New algorithms for enumerating all maximal cliques. Lect Notes Comput Sci 3111:260–272
3. Nakano S (2001) Enumerating floorplans with n rooms. Lect Notes Comput Sci 2223:107–115
4. Nakano S (2004) Efficient generation of triconnected plane triangulations. Comput Geom Theory Appl 27(2):109–122
5. Nakano S, Uno T (2004) Constant time generation of trees with specified diameter. Lect Notes Comput Sci 3353:33–45
6. Shioura A, Tamura A, Uno T (1997) An optimal algorithm for scanning all spanning trees of undirected graphs. SIAM J Comput 26(3):678–692
7. Uno T (2003) Two general methods to reduce delay and change of enumeration algorithms. National Institute of Informatics (in Japan), Technical repor

RNA Secondary Structure Boltzmann Distribution

Rune B. Lyngsø
Department of Statistics, Oxford University, Oxford, UK
Winton Capital Management, Oxford, UK

Keywords

Full partition function

Years and Authors of Summarized Original Work

2005; Miklós, Meyer, Nagy

Problem Definition

This problem is concerned with computing features of the Boltzmann distribution over RNA secondary structures in the context of the standard Gibbs free energy model used for RNA Secondary Structure Prediction by Minimum Free Energy (cf. corresponding entry). Thermodynam-

ics state that for a system with configuration space Ω and free energy given by $E: \Omega \mapsto \mathbf{R}$, the probability of the system being in state $\omega \in \Omega$ is proportional to $e^{-E(\omega)/RT}$ where R is the universal gas constant and T the absolute temperature of the system. The normalizing factor

$$Z = \sum_{\omega \in \Omega} e^{-E(\omega)/RT} \tag{1}$$

is called the *full partition function* of the system.

Over the past several decades, a model approximating the free energy of a structured RNA molecule by independent contributions of its secondary structure components has been developed and refined. The main purpose of this work has been to assess the stability of individual secondarystructures. However, it immediately translates into a distribution over all secondary structures. Early work focused on computing the pairing probability for all pairs of bases, i.e., the sum of the probabilities of all secondary structures containing that base pair. Recent work has extended methods to compute probabilities of base pairing probabilities for RNA heterodimers [2], i.e., interacting RNA molecules, and expectation, variance and higher moments of the Boltzmann distribution.

Notation

Let $s \in \{A, C, G, U\}^*$ denote the sequence of bases of an RNA molecule. Use $X \cdot Y$ where $X, Y \in \{A, C, G, U\}$ to denote a base pair between bases of type X and Y, and $i \cdot j$ where $1 \leq i < j \leq |s|$ to denote a base pair between bases $s[i]$ and $s[j]$.

Definition 1 (RNA Secondary Structure) A secondary structure for an RNA sequence s is a set of base pairs $S = \{i \cdot j \mid 1 \leq i < j \leq |s| \wedge i < j - 3\}$. For $i \cdot j, i' \cdot j' \in S$ with $i \cdot j \neq i' \cdot j'$

- $\{i, j\} \cap \{i', j'\} = \emptyset$ (each base pairs with at most one other base)
- $\{s[i], s[j]\} \in \{\{A, U\}, \{C, G\}, \{G, U\}\}$ (only Watson-Crick and G, U wobble base pairs)
- $i < i' < j \Rightarrow j' < j$ (base pairs are either nested or juxtaposed but not overlapping)

The second requirement, that only canonical base pairs are allowed, is standard but not consequential in solutions to the problem. The third requirement states that the structure does not contain pseudoknots. This restriction is crucial for the results listed in this entry.

Energy Model

The model of Gibbs free energy applied, usually referred to as the nearest-neighbor model, was originally proposed by Tinoco et al. [10, 11]. It approximates the free energy by postulating that the energy of the full three dimensional structure only depends on the secondary structure, and that this in turn can be broken into a sum of independent contributions from each loop in the secondary structure.

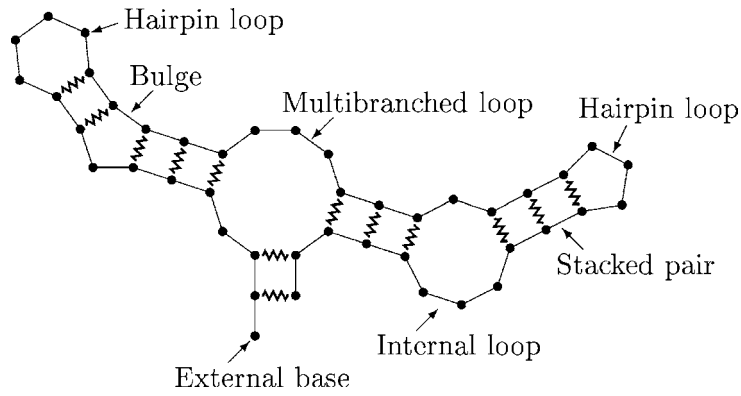
Definition 2 (Loops) For $i \cdot j \in S$, base k is *accessible* from $i \cdot j$ iff $i < k < j$ and $\neg \exists i' \cdot j' \in S: i < i' < k < j' < j$. The *loop closed by* $i \cdot j, \ell_{i \cdot j}$, consists of $i \cdot j$ and all the bases accessible from $i \cdot j$. If $i' \cdot j' \in S$ and i' and j' are accessible from $i \cdot j$, then $i' \cdot j'$ is an interior base pair in the loop closed by $i \cdot j$.

Loops are classified by the number of interior base pairs they contain:

- hairpin loops have no interior base pairs
- stacked pairs, bulges, and internal loops have one interior base pair that is separated from the closing base pair on neither side, on one side, or on both sides, respectively
- multibranched loops have two or more interior base pairs

Bases not accessible from any base pair are called external. This is illustrated in Fig. 1. The free energy of structure S is

$$\Delta G(S) = \sum_{i \cdot j \in S} \Delta G(\ell_{i \cdot j}) \tag{2}$$



RNA Secondary Structure Boltzmann Distribution, Fig. 1 A hypothetical RNA structure illustrating the different loop types. Bases are represented by *circles*, the RNA backbone by *straight lines*, and base pairs by *zigzagged lines*

where $\Delta G(\ell_{i\cdot j})$ is the free energy contribution from the loop closed by $i \cdot j$. The contribution of S to the full partition function is

$$e^{-\Delta G(S)/RT} = e^{-\sum_{i\cdot j \in S} \Delta G(\ell_{i\cdot j})/RT} = \prod_{\ell_{i\cdot j} \in S} e^{-\Delta G(\ell_{i\cdot j})/RT}. \quad (3)$$

Problem 1 (RNA Secondary Structure Distribution)
INPUT: RNA sequence s, absolute temperature T and specification of ΔG at T for all loops.
OUTPUT: $\sum_S e^{-\Delta G(S)/RT}$, where the sum is over all secondary structures for s.

Key Results

Solutions are based on recursions similar to those for RNA Secondary Structure Prediction by Minimum Free Energy, replacing sum and minimization with multiplication and sum (or more generally with a *merge function* and a *choice function* [8]). The key difference is that recursions are required to be non-redundant, i.e., any particular secondary structure only contributes through one path through the recursions.

Theorem 1 *Using the standard thermodynamic model for RNA secondary structures, the partition function can be computed in time $O(|s^3)$ and space $O(|s^2)$. Moreover, the computation can build data structures that allow $O(1)$ queries of the pairing probability of $i \cdot j$ for any $1 \le i < j \le |s|$ [5, 6, 7].*

Theorem 2 *Using the standard thermodynamic model for RNA secondary structures, the expectation and variance of free energy over the Boltzmann distribution can be computed in time $O(|s^3)$ and space $O(|s^2)$. More generally, the kth moment*

$$E_{\text{Boltzmann}}[\Delta G] = 1/Z \sum_S e^{-\Delta G(S)/RT} \Delta G^k(S), \quad (4)$$

where $Z = \sum_S e^{-\Delta G(S)/RT}$ is the full partition function and the sums are over all secondary structures for s, can be computed in time $O(k^2|s^3)$ and space $O(ks^2)$ [8].

In Theorem 2 the free energy does not hold a special place. The theorem holds for any function Φ defined by an independent contribution from each loop,

$$\Phi(S) = \sum_{i\cdot j \in S} \phi\left(\ell_{i\cdot j}\right), \quad (5)$$

provided each loop contribution can be handled with the same efficiency as the free energy contributions. Hence, moments over the Boltzmann distribution of e.g., number of base pairs, unpaired bases, or loops can also be efficiently computed by applying appropriately chosen indicator functions.

Applications

The original use of partition function computations was for discriminating between well defined and less well defined regions of a secondary structure. Minimum free energy predictions will always return a structure. Base pairing probabilities help identify regions where the prediction is uncertain, either due to the approximations of the model or that the real structure indeed does fluctuate between several low energy alternatives. Moments of Boltzmann distributions are used in identifying how biological RNA molecules deviates from random RNA sequences.

The data structures computed in Theorem 1 can also be used to efficiently sample secondary structures from the Boltzmann distribution. This has been used for probabilistic methods for secondary structure prediction, where the centroid of the most likely cluster of sampled structures is returned rather than the most likely, i.e., minimum free energy, structure [3]. This approach better accounts for the entropic effects of large neighborhoods of structurally and energetically very similar structures. As a simple illustration of this effect, consider twice flipping a coin with probability $p > 0.5$ for heads. The probability p^2 of heads in both flips is larger than the probability $p(1-p)$ of heads followed by tails or tails followed by heads (which again is larger than the probability $(1-p)^2$ of tails in both flips). However, if the order of the flips is ignored the probability of one heads and one tails is $2p(1-p)$. The probability of two heads remains p^2 which is smaller than $2p(1-p)$ when $p < \frac{2}{3}$. Similarly a large set of structures with fairly low free energy may be more likely, when viewed as a set, than a small set of structures with very low free energy.

Open Problems

As for RNA Secondary Structure Prediction by Minimum Free Energy, improvements in time and space complexity are always relevant. This may be more difficult for computing distributions, as the more efficient dynamic programming techniques of [9] cannot be applied. In the context of genome scans, the fact that the start and end positions of encoded RNA molecule is unknown has recently been considered [1].

Also the problem of including structures with pseudoknots, i.e., structures violating the last requirement in Definition 1, in the configuration space is an active area of research. It can be expected that all the methods of Theorems 3 through 6 in the entry on RNA Secondary Structure Prediction Including Pseudoknots can be modified to computation of distributions without affecting complexities. This may require some further bookkeeping to ensure non-redundancy of recursions, and only in [4] has this actively been considered.

Though the moments of functions that are defined as sums over independent loop contributions can be computed efficiently, it is unknown whether the same holds for functions with more complex definitions. One such function that has traditionally been used for statistics on RNA secondary structure [12] is the *order* of a secondary structure which refers to the nesting depth of multibranched loops.

URL to Code

Software for partition function computation and a range of related problems is available from www.bioinfo.rpi.edu/applications/hybrid/download.php and www.tbi.univie.ac.at/~ivo/RNA/. Software including a restricted class of structures with pseudoknots [4] is available at www.nupack.org.

Cross-References

- RNA Secondary Structure Prediction Including Pseudoknots
- RNA Secondary Structure Prediction by Minimum Free Energy

Recommended Reading

1. Bernhart S, Hofacker IL, Stadler P (2006) Local RNA base pairing probabilities in large sequences. Bioinformatics 22:614–615
2. Bernhart SH, Tafer H, Mückstein U, Flamm C, Stadler PF, Hofacker IL (2006) Partition function and base pairing probabilities of RNA heterodimers. Algorithms Mol Biol 1:3
3. Ding Y, Chan CY, Lawrence CE (2005) RNA secondary structure prediction by centroids in a Boltzmann weighted ensemble. RNA 11:1157–1166
4. Dirks RM, Pierce NA (2003) A partition function algorithm for nucleic acid secondary structure including pseudoknots. J Comput Chem 24:1664–1677
5. Hofacker IL, Stadler PF (2006) Memory efficient folding algorithms for circular RNA secondary structures. Bioinformatics 22:1172–1176
6. Lyngsø, RB, Zuker M, Pedersen CNS (1999) Fast evaluation of internal loops in RNA secondary structure prediction. Bioinformatics 15:440–445
7. McCaskill JS (1990) The equilibrium partition function and base pair binding probabilities for RNA secondary structure. Biopolymers 29:1105–1119
8. Miklós I, Meyer IM, Nagy B (2005) Moments of the Boltzmann distribution for RNA secondary structures. Bull Math Biol 67:1031–1047
9. Ogurtsov AY, Shabalina SA, Kondrashov AS, Roytberg MA (2006) Analysis of internal loops within the RNA secondary structure in almost quadratic time. Bioinformatics 22:1317–1324
10. Tinoco I, Borer PN, Dengler B, Levine MD, Uhlenbeck OC, Crothers DM, Gralla J (1973) Improved estimation of secondary structure in ribonucleic acids. Nat New Biol 246:40–41
11. Tinoco I, Uhlenbeck OC, Levine MD (1971) Estimation of secondary structure in ribonucleic acids. Nature 230:362–367
12. Waterman MS (1978) Secondary structure of single-stranded nucleic acids. Adv Math Suppl Stud 1:167–212

RNA Secondary Structure Prediction by Minimum Free Energy

Rune B. Lyngsø
Department of Statistics, Oxford University, Oxford, UK
Winton Capital Management, Oxford, UK

Keywords

RNA folding

Years and Authors of Summarized Original Work

2006; Ogurtsov, Shabalina, Kondrashov, Roytberg

Problem Definition

This problem is concerned with predicting the set of base pairs formed in the native structure of an RNA molecule. The main motivation stems from structure being crucial for function and the growing appreciation of the importance of RNA molecules in biological processes. Base pairing is the single most important factor determining structure formation. Knowledge of the secondary structure alone also provides information about stretches of unpaired bases that are likely candidates for active sites. Early work [7] focused on finding structures maximizing the number of base pairs. With the work of Zuker and Stiegler [17], focus shifted to energy minimization in a model approximating the Gibbs free energy of structures.

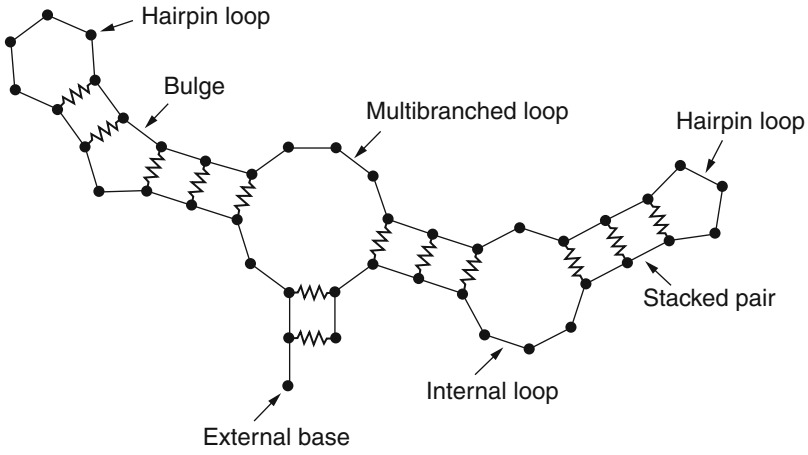
Notation

Let $s \in \{A, C, G, U\}^*$ denote the sequence of bases of an RNA molecule. Use $X \cdot Y$ where $X, Y \in \{A, C, G, U\}$ to denote a base pair between bases of type X and Y and $i \cdot j$ where $1 \leq i < j \leq |s|$ to denote a base pair between bases $s[i]$ and $s[j]$.

Definition 1 (RNA Secondary Structure) A secondary structure for an RNA sequence s is a set of base pairs $\mathcal{S} = \{i \cdot j \,|\, 1 \leq i < j \leq |s| \wedge i < j - 3\}$. For $i \cdot j, i' \cdot j' \in \mathcal{S}$ with $i \cdot j \neq i' \cdot j'$:

- $\{i, j\} \cap \{i', j'\} = \emptyset$ (each base pair with at most one other base)
- $\{s[i], s[j]\} \in \{\{A, U\}, \{C, G\}, \{G, U\}\}$ (only Watson-Crick and G, U wobble base pairs)
- $i < i' < j \Rightarrow j' < j$ (base pairs are either nested or juxtaposed but not overlapping)

The second requirement that only canonical base pairs are allowed is standard but not consequen-



RNA Secondary Structure Prediction by Minimum Free Energy, Fig. 1 A hypothetical RNA structure illustrating the different loop types. Bases are represented by *circles*, the RNA backbone by *straight lines*, and base pairs by *zigzagged lines*

tial in solutions to the problem. The third requirement states that the structure does not contain pseudoknots. This restriction is crucial for the results listed in this entry.

Energy Model

The model of Gibbs free energy applied, usually referred to as the nearest-neighbor model, was originally proposed by Tinoco et al. [10, 11]. It approximates the free energy by postulating that the energy of the full three-dimensional structure only depends on the secondary structure and that this in turn can be broken into a sum of independent contributions from each loop in the secondary structure.

Definition 2 (Loops) For $i \cdot j \in \mathcal{S}$, base k is *accessible* from $i \cdot j$ iff $i < k < j$ and $\neg\exists i' \cdot j' \in \mathcal{S} : i < i' < k < j' < j$. The *loop closed by* $i \cdot j, \ell_{i \cdot j}$, consists of $i \cdot j$ and all the bases accessible from $i \cdot j$. If $i' \cdot j' \in \mathcal{S}$ and i' and j' are accessible from $i \cdot j$, then $i' \cdot j'$ is an interior base pair in the loop closed by $i \cdot j$.

Loops are classified by the number of interior base pairs they contain:

- Hairpin loops have no interior base pairs.
- Stacked pairs, bulges, and internal loops have one interior base pair that is separated from the closing base pair on neither side, on one side, or on both sides, respectively.
- Multibranched loops have two or more interior base pairs.

Bases not accessible from any base pair are called external. This is illustrated in Fig. 1. The free energy of structure $\mathcal{S}$ is

$$\Delta G(\mathcal{S}) = \sum_{i \cdot j \in \mathcal{S}} \Delta G(\ell_{i \cdot j}), \qquad (1)$$

where $\Delta G(\ell_{i \cdot j})$ is the free energy contribution from the loop closed by $i \cdot j$.

Problem 1 (Minimum Free Energy Structure)

INPUT: RNA sequence s and specification of ΔG for all loops

$$\arg\min_{\mathcal{S}} \{\Delta G(\mathcal{S}) | \mathcal{S} \text{ secondary structure for } s\}.$$

OUTPUT: A secondary structure achieving the minimum of free energies, taken over all possible secondary structures

Key Results

Solutions are based on using dynamic programming to solve the general recursion

R

$$V[i,j] = \min_{k\geq 0; i<i_1<j_1<\cdots<i_k<j_k<j} \left\{ \Delta G(\ell_{i\cdot j; i_1\cdot j_1,\ldots,i_k\cdot j_k}) + \sum_{l=1}^{k} V[i_l, j_l] \right\}$$

$$W[i] = \min \left\{ W[i-1], \min_{0<k<i} \{W[k-1] + V[k,i]\} \right\},$$

where $\Delta G(\ell_{i\cdot j; i_1\cdot j_1,\ldots,i_k,j_k})$ is the free energy of the loop closed by $i \cdot j$ and interior base pairs $i_1 \cdot j_1, \ldots, i_k \cdot j_k$ and with initial condition $W[0] = 0$. In the following, it is assumed that all loop energies can be computed in time $O(1)$.

Theorem 1 *If the free energy of multibranched loops is a sum of:*

- *An affine function of the number of interior base pairs and unpaired bases*
- *Contributions for each base pair from stacking with either neighboring unpaired bases in the loop or with a neighboring base pair in the loop, whichever is more favorable*

a minimum free energy structure can be computed in time $O(|s|^4)$ and space $O(|s|^2)$ [17].

With these assumptions, the time required to handle the multibranched loop parts of the recursion reduces to $O(|s|^3)$. Hence, handling the $O(|s|^4)$ possible internal loops becomes the bottleneck.

Theorem 2 *If furthermore the free energy of internal loops is a sum of:*

- *A function of the total size of the loop, i.e., the number of unpaired bases in the loop*
- *A function of the asymmetry of the loop, i.e., the difference in number of unpaired bases on the two sides of the loop*
- *Contributions from the closing and interior base pairs stacking with the neighboring unpaired bases in the loop*

a minimum free energy structure can be computed in time $O(|s|^3)$ and space $O(|s|^2)$ [5].

Under these assumptions, the time required to handle internal loops reduces to $O(|s|^3)$. With further assumptions on the free energy contributions of internal loops, this can be reduced even further, again making the handling of multibranched loops the bottleneck of the computation.

Theorem 3 *If furthermore the size dependency is concave and the asymmetry dependency is constant for all but $O(1)$ values, a multibranched loop free minimum free energy structure can be computed in time $O(|s|^2 \log^2 |s|)$ and space $O(|s|^2)$ [8].*

The above assumptions are all based on the nature of current loop energies [6]. These energies have to a large part been developed without consideration of computational expediency and parameters determined experimentally, although understanding of the precise behavior of larger loops is limited. For multibranched loops, some theoretical considerations [4] would suggest that a logarithmic dependency would be more appropriate.

Theorem 4 *If the restriction on the dependency on number of interior base pairs and unpaired bases in Theorem 1 is weakened to any function that depends only on the number of interior base pairs, the number of unpaired bases, or the total number of bases in the loop, a minimum free energy structure can be computed in time $O(n^4)$ and space $O(n^3)$ [13].*

Theorem 5 *All the above theorems can be modified to compute a data structure that for any $1 \leq i < j \leq |s|$ allows us to compute the minimum free energy of any structure containing $i \cdot j$ in time $O(1)$ [15].*

Applications

Naturally, the key application of these algorithms is for predicting the secondary structure of

RNA molecules. This holds in particular for sequences with no homologues with common structure, e.g., functional analysis based on mutational effects and to some extent analysis of RNA aptamers. With access to structurally conserved homologues, prediction accuracy is significantly improved by incorporating comparative information [2].

Incorporating comparative information seems to be crucial when using secondary structure prediction as the basis of RNA gene finding. As it turns out, the minimum free energy of known RNA genes is not sufficiently different from the minimum free energy of comparable random sequences to reliably separate the two [9,14]. However, minimum free energy calculations are at the core of one successful comparative RNA gene finder [12].

Open Problems

Most current research is focused on refinement of the energy parametrization. The limiting factor of sequence lengths for which secondary structure prediction by the methods described here is still feasible is adequacy of the nearest-neighbor approximation rather than computation time and space. Still, improvements on time and space complexities are useful as biosequence analyses are invariably used in genome scans. In particular, improvements on Theorem 4, possibly for dependencies restricted to be logarithmic or concave, would allow for more advanced scoring of multibranched loops. A more esoteric open problem is to establish the complexity of computing the minimum free energy under the general formulation of (1), with no restrictions on loop energies except that they are computable in time polynomial in $|s|$.

Experimental Results

With the release of the most recent energy parameters [6], secondary structure prediction by finding a minimum free energy structure was found to recover approximately 73 % of the base pairs in a benchmark data set of RNA sequences with known secondary structure. Another independent assessment [1] put the recovery percentage somewhat lower at around 56 %. This discrepancy is discussed and explained in [1].

Data Sets

Families of homologous RNA sequences aligned and annotated with secondary structure are available from the Rfam database at www.sanger.ac.uk/Software/Rfam/. Three-dimensional structures are available from the Nucleic Acid Database at ndbserver.rutgers.edu/. An extensive list of this and other databases is available at www.imb-jena.de/RNA.html.

URL to Code

Software for RNA folding and a range of related problems is available at www.bioinfo.rpi.edu/applications/hybrid/download.php and www.tbi.univie.ac.at/~ivo/RNA/. Software implementing the efficient handling of internal loops of [8] is available at ftp.ncbi.nlm.nih.gov/pub/ogurtsov/Afold.

Cross-References

- ▸ RNA Secondary Structure Boltzmann Distribution
- ▸ RNA Secondary Structure Prediction Including Pseudoknots

Recommended Reading

1. Dowell R, Eddy SR (2004) Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. BMC Bioinform 5:71
2. Gardner PP, Giegerich R (2004) A comprehensive comparison of comparative RNA structure prediction approaches. BMC Bioinform 30:140
3. Hofacker IL, Stadler PF (2006) Memory efficient folding algorithms for circular RNA secondary structures. Bioinformatics 22:1172–1176

4. Jacobson H, Stockmayer WH (1950) Intramolecular reaction in polycondensations. I. The theory of linear systems. J Chem Phys 18:1600–1606
5. Lyngsø RB, Zuker M, Pedersen CNS (1999) Fast evaluation of internal loops in RNA secondary structure prediction. Bioinformatics 15:440–445
6. Mathews DH, Sabina J, Zuker M, Turner DH (1999) Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. J Mol Biol 288:911–940
7. Nussinov R, Jacobson AB (1980) Fast algorithm for predicting the secondary structure of single-stranded RNA. Proc Natl Acad Sci USA 77:6309–6313
8. Ogurtsov AY, Shabalina SA, Kondrashov AS, Roytberg MA (2006) Analysis of internal loops within the RNA secondary structure in almost quadratic time. Bioinformatics 22:1317–1324
9. Rivas E, Eddy SR (2000) Secondary structure alone is generally not statistically significant for the detection of noncoding RNAs. Bioinformatics 16:583–605
10. Tinoco I, Borer PN, Dengler B, Levine MD, Uhlenbeck OC, Crothers DM, Gralla J (1973) Improved estimation of secondary structure in ribonucleic acids. Nat New Biol 246:40–41
11. Tinoco I, Uhlenbeck OC, Levine MD (1971) Estimation of secondary structure in ribonucleic acids. Nature 230:362–367
12. Washietl S, Hofacker IL, Stadler PF (2005) Fast and reliable prediction of noncoding RNA. Proc Natl Acad Sci USA 102:2454–2459
13. Waterman MS, Smith TF (1986) Rapid dynamic programming methods for RNA secondary structure. Adv Appl Math 7:455–464
14. Workman C, Krogh A (1999) No evidence that mRNAs have lower folding free energies than random sequences with the same dinucleotide distribution. Nucleic Acids Res 27:4816–4822
15. Zuker M (1989) On finding all suboptimal foldings of an RNA molecule. Science 244:48–52
16. Zuker M (2000) Calculating nucleic acid secondary structure. Curr Opin Struct Biol 10:303–310
17. Zuker M, Stiegler P (1981) Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Res 9:133–148

RNA Secondary Structure Prediction Including Pseudoknots

Rune B. Lyngsø
Department of Statistics, Oxford University, Oxford, UK
Winton Capital Management, Oxford, UK

Keywords

Abbreviated as *pseudoknot prediction*

Years and Authors of Summarized Original Work

2004; Lyngsø

Problem Definition

This problem is concerned with predicting the set of base pairs formed in the native structure of an RNA molecule, including overlapping base pairs also known as pseudoknots. Standard approaches to RNA secondary structure prediction only allow sets of base pairs that are hierarchically nested. Though few known real structures require the removal of more than a small percentage of their base pairs to meet these criteria, a significant percentage of known real structures contain at least a few base pairs overlapping other base pairs. Pseudoknot substructures are known to be crucial for biological function in several contexts. One of the more complex known pseudoknot structures is illustrated in Fig. 1.

Notation

Let $s \in \{A, C, G, U\}^*$ denote the sequence of bases of an RNA molecule. Use $X \cdot Y$ where $X, Y \in \{A, C, G, U\}$ to denote a base pair between bases of type X and Y and $i \cdot j$ where $1 \leq i < j \leq |s|$ to denote a base pair between bases $s[i]$ and $s[j]$.

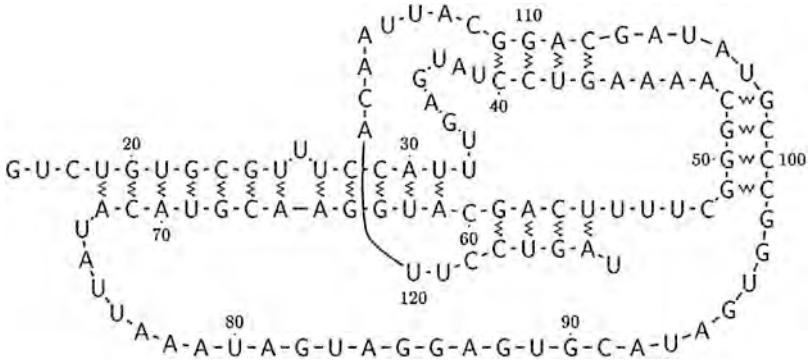
Definition 1 (RNA Secondary Structure) A secondary structure for an RNA sequence s is a set of base pairs $\mathcal{S} = \{i \cdot j | 1 \leq i \leq j \leq |s| \wedge i < j - 3\}$. For $i \cdot j, i' \cdot j' \in \mathcal{S}$ with $i \cdot j \neq i' \cdot j'$:

- $\{i, j\} \cap \{i', j'\} = \emptyset$ (each base pair with at most one other base)
- $\{s[i], s[j]\} \in \{\{A, U\}, \{C, G\}, \{G, U\}\}$ (only Watson-Crick and G, U wobble base pairs)

The second requirement that only canonical base pairs are allowed is standard but not consequential in solutions to the problem.

Scoring Schemes

Structures are usually assessed by extending the model of Gibbs free energy used for ▸ RNA Secondary Structure Prediction by Minimum Free Energy (cf. corresponding entry) with ad hoc



RNA Secondary Structure Prediction Including Pseudoknots, Fig. 1 Secondary structure of the *Escherichia coli* α operon mRNA from position 16 to position 127, cf. [5], Figure 1. The backbone of the RNA molecule is drawn as *straight lines*, while base pairings are shown with *zigzagged lines*

extrapolation of multibranched loop energies to pseudoknot substructures [11] or by summing independent contributions, e.g., obtained from base pair restricted minimum free energy structures from each base pair [13]. To investigate the complexity of pseudoknot prediction, the following three simple scoring schemes will also be considered:

Number of base pairs	$\#BP(\mathcal{S}) = \lvert\mathcal{S}\rvert$
Number of stacking base pairs	$\#SBP(\mathcal{S}) = \lvert\{i \cdot j \in \mathcal{S} \lvert i+1 \cdot j-1 \in \mathcal{S} \vee i-1 \cdot j+1 \in \mathcal{S}\}\rvert$
Number of base pair stackings	$\#BPS(\mathcal{S}) = \lvert\{i \cdot j \in \mathcal{S} \lvert i+1 \cdot j-1 \in \mathcal{S}\}\rvert$

These scoring schemes are inspired by the fact that stacked pairs are essentially the only loops having a stabilizing contribution in the Gibbs free energy model.

Problem 2 (Pseudoknot Prediction)

INPUT: RNA sequence s and an appropriately specified scoring scheme

OUTPUT: A secondary structure $\mathcal{S}$ for s that is optimal under the scoring scheme specified

Key Results

Theorem 1 *The complexities of pseudoknot prediction under the three simplified scoring schemes can be classified as follows, where* Σ *denotes the alphabet.*

Theorem 2 *If structures are restricted to be planar, i.e., the graph with the bases of the sequence as nodes and base pairs and backbone links of consecutive bases as edges is required to be planar, pseudoknot prediction under the #BPS scoring scheme is NP-hard for an alphabet of size 4. Conversely, a 1/2-approximation can be found in time* $O(|s|^3)$ *and space* $O(|s|^2)$ *by observing that an optimal pseudoknot free structure is a 1/2-approximation [6].*

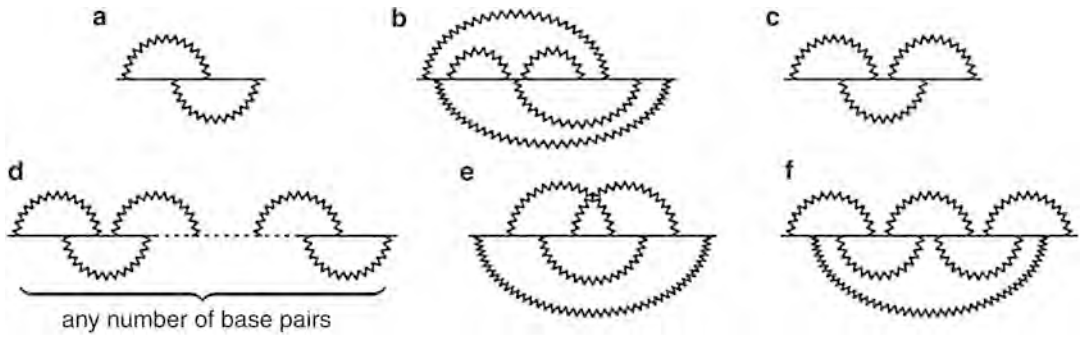
There are no steric reasons that RNA secondary structures should be planar, and the structure in Fig. 1 is actually nonplanar. Nevertheless, known real structures have relatively simple overlapping base pair patterns with very few nonplanar structures known. Hence, planarity has been used as a defining restriction on pseudoknotted structures [2, 15]. Similar reasoning has led to the development of several algorithms for finding an optimal structure from restricted classes of structures. These algorithms tend to use more realistic scoring schemes, e.g., extensions of the Gibbs free energy model, than the three simple scoring schemes considered above.

Theorem 3 *Pseudoknot prediction for a restricted class of structures including Fig. 2a–e, but not Fig. 2f, can be done in time* $O(|s|^6)$ *and space* $O(|s|^4)$ *[11].*

Theorem 4 *Pseudoknot prediction for a restricted class of planar structures including Fig. 2a–c, but not Fig. 2d–f, can be done in time* $O(|s|^5)$ *and space* $O(|s|^4)$ *[14].*

Theorem 5 *Pseudoknot prediction for a restricted class of planar structures including Fig. 2a, b, but not Fig. 2c–f, can be done in time* $O(|s|^5)$ *and space* $O(|s|^4)$ *or* $O(|s|^3)$ *[1, 4] (methods differ in generality of scoring schemes that can be used).*

	Fixed alphabet	Unbounded alphabet
#BP [13]	Time $O(\|s\|^3)$, space $O(\|s\|^2)$	Time $O(\|s\|^3)$, space $O(\|s\|^2)$
#SBP [7]	Time $O(\|s\|^{\|\Sigma\|^2+\|\Sigma\|^3})$, space $O(\|s\|^{\|\Sigma\|^2+\|\Sigma\|^3})$	NP hard
#BPS	NP hard for $\|\Sigma = 2\|$, PTAS [7] 1/3-approximation in time $O(\|s\|)$ [6]	NP hard [7], 1/3-approximation in time and space $O(\|s\|^2)$ [6]



RNA Secondary Structure Prediction Including Pseudoknots, Fig. 2 RNA secondary structures illustrating restrictions of pseudoknot prediction algorithms. Backbone is drawn as a *straight line*, while base pairings are shown with *zigzagged arcs*

Theorem 6 *Pseudoknot prediction for a restricted class of planar structures including Fig. 2a, but not Fig. 2b–f, can be done in time* $O(|s|^4)$ *and space* $O(|s|^2)$ *[1, 8].*

Theorem 7 *Recognition of structures belonging to the restricted classes of Theorems 3, 5, and 6 and enumeration of all irreducible cycles (i.e., loops) in such structures can be done in time* $O(|s|)$ *[3, 9].*

Applications

As for the prediction of RNA secondary structures without pseudoknots, the key application of these algorithms is for predicting the secondary structure of individual RNA molecules. Due to the steep complexities of the algorithms of Theorems 3–6, these are less well suited for genome scans than prediction without pseudoknots.

Enumerating all loops of a structure in linear time also allows scoring a structure in linear time, as long as the scoring scheme allows the score of a loop to be computed in time proportional to its size. This has practical applications in heuristic searches for good structures containing pseudoknots.

Open Problems

Efficient algorithms for prediction based on restricted classes of structures with pseudoknots that still contain a significant fraction of all known structures are an active area of research. Even using the more theoretical simple *#SBP* scoring scheme, developing, e.g., an $O(|s|^{|\Sigma|})$ algorithm for this problem would be of practical significance. From a theoretical point of view, the complexity of planar structures is the least well understood, with results for only the *#BPS* scoring scheme.

Classification of realistic energy models for RNA secondary structures with pseudoknots is much less developed than for RNA secondary structures without pseudoknots. Several recent papers have been addressing this gap [3, 9, 12].

Data Sets

PseudoBase at http://biology.leidenuniv.nl/~batenburg/PKB.html is a repository of representatives of most known RNA structures with pseudoknots.

URL to Code

The method of Theorem 3 is available at http://selab.janelia.org/software.html#pknots and of one of the methods of Theorem 5 at http://www.nupack.org, and an implementation applying a slight heuristic reduction of the class of structures considered by the method of Theorem 6 is available at http://bibiserv.techfak.uni-bielefeld.de/pknotsrg/ [10].

Cross-References

- RNA Secondary Structure Prediction by Minimum Free Energy

Recommended Reading

1. Akutsu T (2000) Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. Discret Appl Math 104:45–62
2. Brown M, Wilson C (1996) RNA pseudoknot modeling using intersections of stochastic context free grammars with applications to database search. In: Hunter L, Klein T (eds) Proceedings of the 1st Pacific symposium on biocomputing, Big Island of Hawaii, pp 109–125
3. Condon A, Davy B, Rastegari B, Tarrant F, Zhao S (2004) Classifying RNA pseudoknotted structures. Theor Comput Sci 320:35–50
4. Dirks RM, Pierce NA (2003) A partition function algorithm for nucleic acid secondary structure including pseudoknots. J Comput Chem 24:1664–1677
5. Gluick TC, Draper DE (1994) Thermodynamics of folding a pseudoknotted mRNA fragment. J Mol Biol 241:246–262
6. Ieong S, Kao M-Y, Lam T-W, Sung W-K, Yiu S-M (2001) Predicting RNA secondary structures with arbitrary pseudoknots by maximizing the number of stacking pairs. In: Proceedings of the 2nd symposium on bioinformatics and bioengineering, Bethesda, pp 183–190
7. Lyngsø RB (2004) Complexity of pseudoknot prediction in simple models. In: Proceedings of the 31th international colloquium on automata, languages and programming (ICALP), Turku, pp 919–931
8. Lyngsø RB, Pedersen CNS (2000) RNA pseudoknot prediction in energy based models. J Comput Biol 7:409–428
9. Rastegari B, Condon A (2007) Parsing nucleic acid pseudoknotted secondary structure: algorithm and applications. J Comput Biol 14(1):16–32
10. Reeder J, Giegerich R (2004) Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. BMC Bioinform 5:104
11. Rivas E, Eddy S (1999) A dynamic programming algorithm for RNA structure prediction including pseudoknots. J Mol Biol 285:2053–2068
12. Rødland EA (2006) Pseudoknots in RNA secondary structure: representation, enumeration, and prevalence. J Comput Biol 13:1197–1213
13. Tabaska JE, Cary RB, Gabow HN, Stormo GD (1998) An RNA folding method capable of identifying pseudoknots and base triples. Bioinformatics 14:691–699
14. Uemura Y, Hasegawa A, Kobayashi S, Yokomori T (1999) Tree adjoining grammars for RNA structure prediction. Theor Comput Sci 210:277–303
15. Witwer C, Hofacker IL, Stadler PF (2004) Prediction of consensus RNA secondary structures including pseudoknots. IEEE Trans Comput Biol Bioinform 1:66–77

Robotics

Elmar Langetepe
Department of Computer Science, University of Bonn, Bonn, Germany

Keywords

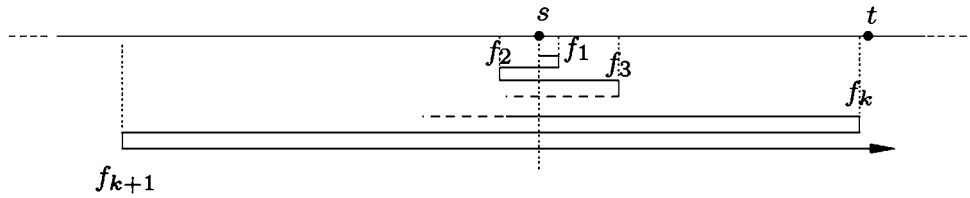
Competitive analysis; Exploration; Motion planning; Navigation; Online algorithms; Searching

Years and Authors of Summarized Original Work

1997; (Navigation) Blum, Raghavan, Schieber
1998; (Exploration) Deng, Kameda, Papadimitriou
2008; (Searching and Exploration) Fleischer, Kamphans, Klein, Langetepe, Trippen

Problem Definition

Since ancient history mankind has been fascinated by the problem of orienting itself in unknown environments. Problems like *escaping*



Robotics, Fig. 1 Any reasonable strategy for searching for a point on a line can be expressed as a sequence $X = f_1, f_2, f_3, \ldots$ of the search depths of the strategy

from a labyrinth or *searching for target objects* have been considered and discussed intensively. Such problems can be easily modeled in a geometric setting.

For example, let us assume that an agent is searching for an unknown door along a wall. We assume that the door is detected, if the agent exactly hits the door. Geometrically one is searching for an unknown point along a line as depicted in Fig. 1. Although the location of the point is not known and is only detected by a visit, it should be found without too much detour. This classical problem in online navigation was discussed by [2] in the early 1990s.

Since the distance and the location of the goal is not known, obviously any reasonable (deterministic) strategy should move in the two directions alternatingly and with increasing depths, f_i, until the goal is detected.

A classical result of [13] shows that it is optimal to use a search strategy that doubles the search distance in every step, i.e., $f_i = 2^i$. It can be shown that the resulting path to an arbitrary target point t is never greater than 9 times the shortest path to the target t, regardless of the position of t. There is no deterministic strategy that attains a smaller factor; see also [2].

Navigation and Exploration

We categorize three fundamental tasks: navigation, exploration, and localization. Navigation (or search) means to find a way to a predescribed location in an unknown environment as shown above. Exploration means to draw a complete map of an unknown environment or to detect or visit *all* possible targets. Localization means to determine the currently unknown position on a known and given map. In many settings, the environment is modeled geometrically as a simple polygon with or without holes. To distinguish the underlying combinatorial problems from the geometric problems, an environment may also be modeled as a graph. For an overview of online searching and exploration problems, see [25] or the more recent survey of Gal [14].

Performance Measure, Competitive Analysis

A general concept for evaluating the efficiency of an online strategy is the so-called *competitive analysis*. Formally, for a class of problems Π and any instance $P \in \Pi$, the cost, $\mathrm{OnlAlg}(P)$, of the online algorithm is compared to the cost, $\mathrm{OfflOpt}(P)$, of the optimal offline algorithm. If there are constants C and A, so that

$$\mathrm{OnlAlg}(P) \leq C \times \mathrm{OfflOpt}(P) + A$$

holds for any $P \in \Pi$, the online algorithm is called C *competitive*. In the case of exploration and navigation, the robot should minimize its travel distance. Therefore, the *competitive ratio* C measures the length of the detour compared to the optimal shortest tour computed under full information. An overview of efficient computations of optimal offline solutions for shortest paths problems can be found in the survey of Mitchell [23]. Many online motion planning problems were classified by the competitive analysis; see the surveys [4, 10, 25].

A randomized online algorithm against an oblivious adversary uses randomization on a fixed predetermined input (which is unknown to the online algorithm). In this case, the competitive ratio is a random variable, and it is maximized over all possible inputs. For

example, an optimal randomized strategy for the introductory *point-on-a-line* problem given by Kao et al. [17] achieves an optimal competitive ratio of 4.5911 . . ..

Different Models

The robot can be equipped with a vision system or with a local touch sensor, only. The impact of a compass is of some interest. One can consider continuous geometric settings such as (a collection of) simple polygons or a concatenation of corridors. On the other hand, the geometric environment might be given by a discrete concatenation of single cells (i.e., a grid graph environment) or is modeled by a general graph. Furthermore, we can consider a single robot or a set of k agents which are working together and exchange information to some extent. Additionally the size of the memory of the agents can be limited. Tasks for a huge set of agents with very limited abilities are related to swarm behavior which is not the topic of this overview.

Key Results

Navigation

Blum et al. [6] studied the problem of a blind robot trying to reach a goal t from a start position s (*point-to-point navigation*) in a two-dimensional scene of n non-overlapping axis-parallel rectangles of width at least one. In the *wall problem*, t is an infinite vertical line. In the *room problem*, the obstacles are within a square room with entry door s and the target t lies on the outer boundary. $O(\sqrt{n})$ competitive online algorithms have been developed. A lower bound on the competitive ratio of $\Omega(\sqrt{n})$ for the wall problem was given by [24], and for the room problem optimal $\Theta(\log n)$ competitive algorithms have been presented by [3]. For randomized strategies and point-to-point navigation, there is an $\Omega(\log \log n)$ lower bound for the model of an oblivious adversary from [18], and [5] presented randomized $O(\log n)$ competitive algorithms for the same problem and also for the wall problem.

The introductory search problem for the line (or 2-rays) was extended to m concurrent rays where an optimal competitive ratio of $1 + 2m^m/(m-1)^{m-1}$ was shown; see [2,13]. An optimal strategy visits the m rays alternatingly with search depth $f_i = (m/(m-1))^i$. For p agents on m rays working in parallel and exchanging information, an optimal ratio of $1+2(m/p-1)(m/(m-p))^{m/p}$ can be achieved; see [21]. In a natural extension in dimension 2, the robot scans the area with a radar connected to the starting point. Gal [13] introduced this two-dimensional search problem and conjectures that a logarithmic spiral (i.e., the natural continuous extension of the doubling heuristic) gives an optimal strategy. The best logarithmic spiral attains a competitive ratio of 17.289 . . .; finally a proof for the optimality of spiral search is given in [20].

Exploration

Deng et al. [7] introduced the online gallery route problem. We consider a simple room modeled by a simple polygon and an agent equipped with a visibility system. The task of computing the shortest roundtrip so that the agent *sees* all points in the polygon is denoted as the shortest watchman route (SWR) problem. In the case of a rectilinear simple polygon and with L_1-metric, there is an optimal (i.e., 1-competitive) online algorithm which gives a $\sqrt{2}$-approximation of the SWR for the L_2-metric in the rectilinear problem.

For general simple polygons, the problem was first solved by Icking et al. [16] with a proven competitive ratio of 26.5, whereas the greatest known lower bound is given by 1.28; see [15].

For the exploration of a geometric environment with k rectilinear obstacles, there is an $\Omega(\sqrt{k})$ lower bound on the competitive ratio for deterministic and randomized strategies; see [1].

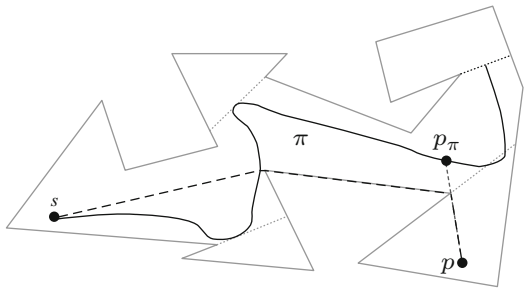
Online graph exploration by a set of k agents means that every vertex of an unknown graph has to be visited. In some configuration, additionally all edges have to be traversed. Assume that full communication among the agents is given. Finding the optimal makespan (finishing time)

algorithm for k agents is an NP-hard problem even for a given tree, and there is an $O(k/\log k)$ competitive algorithm for the online exploration (edges and vertices) version; see [12]. On the other hand, a special tree construction in [9] gives an $\Omega(\log k/\log\log k)$ lower bound on the competitive ratio. For cell environments (grid graph with or without holes), optimal competitive strategies exist.

Dependency Between Searching and Exploration

From the m-concurrent rays result above, one can easily deduce that there is no constant competitive online strategy for searching a point in simple polygons with n edges where n can grow. Nevertheless in a fixed polygon, any search strategy that sees all points defines a ratio for any point and has a worst case ratio; see Fig. 2. So there have to be some optimal *search path* for any fixed polygon. We want to find a general strategy that approximates the best search path for any polygon within a constant factor, i.e., within a constant *search ratio*.

The search ratio definition was first given by Koutsoupias et al. [19]; they studied graphs with unit edge length. The result of Koutsoupias et al. is restricted to the *offline* case where the graph is completely known a priori. Only the goal remains hidden. The above concept goes beyond competitive analysis, although the definitions of the search ratio and the competitive factor are quite similar. In the competitive framework, we compare the online path from the start to the goal to the shortest s–to–t path for any possible goal. For an approximation of the optimal search ratio, we compare the online path to the best possible offline path for any goal, which – in turn – may already have a very bad competitive ratio.



Robotics, Fig. 2 In a fixed polygon, a search path π *sees* all points and attains a ratio for any single point. At p_π the target point p is detected for the first time and defines a ratio

The key idea for solving this problem also indicates the general dependency between *searching* and *exploration*. We make use of efficient (probably constant competitive) exploration strategies for the given environments. If they can be restricted to a bounded distance in a somewhat greater environment, we successively increase the exploration depth by a *doubling* factor.

Fleischer et al. [11] showed that it is possible to approximate an optimal search path for searching a point in a simple polygon by a factor of roughly 4 if the goal has to be visited directly. If a vision system is used, a factor of roughly 8 can be guaranteed. The result even holds when the environment is not given in advance. And the result also holds even though the optimal search path in a given simple polygon is not known.

Applications

In practice a robot can efficiently arrive at a target point (given by coordinates) in an unknown environment with obstacles by Lumelsky's BUG strategy [22]. Many such BUG-variants were developed and successfully applied, for example, in some of the mars rover expeditions. If Lumelsky's BUG algorithm is assumed to navigate between convex obstacles, in the worst case it moves at most once around every significant obstacle, which is optimal in this case. Additionally a robot with a compass can sometimes find the goal exponentially faster than a robot without a compass.

Theoretical paradigms have practical relevance in robotics; see Dudek und Jenkin [8]. The doubling heuristic is widely accepted as an approximation scheme in practice. The general concept for the approximation of the optimal search path has some influence. If somebody is searching for a target in an unknown

environment, it seems to be unavoidably that the environment has to be explored efficiently with increasing depth.

The concept also shows that there is sometimes no significant difference between a known environment with unknown targets and a fully unknown environment. Roughly speaking, if somebody is searching for a goal in unknown position, it is not important whether the corresponding environment is fully known in advance or not known at all.

Open Problems

For many settings, the precise competitive complexity of an online motion planning problems is not known. Tight lower bounds are much harder to achieve. Some examples are given below. The lower bound construction for the navigation among obstacles usually make use of arbitrary *thin* obstacles. Is it possible to get rid of such a restriction?

- Exploration of a simple room with visibility: Upper bound 26.5 vs. lower bound 1.28
- Exploration of graphs by k agents: Upper bound $O(k/\log k)$ vs. lower bound $\Omega(\log k/\log\log k)$
- Optimality of spiral search? Upper bounds given by spiral search.
  - Searching for a line in the plane: Upper bound 13.81113 ... vs. lower bound $\sqrt{2}\cdot 9$
  - Searching for a ray in the plane: Upper bound 22.513 ... vs. lower bound 17.289 ...
- Navigation among k obstacles: Does the lower bound of $\Omega(\sqrt{k})$ hold for a fixed aspect ratio of the obstacles?
- Optimal search path: How to compute the optimal search path for a given polygon or graph?

Cross-References

- ▸ Alternative Performance Measures in Online Algorithms
- ▸ Deterministic Searching on the Line
- ▸ Metrical Task Systems
- ▸ Mobile Agents and Exploration
- ▸ Randomized Searching on Rays or the Line

Recommended Reading

1. Albers S, Kursawe K, Schuierer S (2002) Exploring unknown environments with obstacles. Algorithmica 32:123–143
2. Baeza-Yates R, Culberson J, Rawlins G (1993) Searching in the plane. Inf Comput 106:234–252
3. Bar-Eli E, Berman P, Fiat A, Yan P (1994) Online navigation in a room. J Algorithms 17:319–341
4. Berman P (1998) On-line searching and navigation. In: Fiat A, Woeginger G (eds) Competitive analysis of algorithms. Springer, London UK
5. Berman P, Blum A, Fiat A, Karloff H, Rosen A, Saks M (1996) Randomized robot navigation algorithms. In: Proceedings of the 7th ACM-SIAM symposium on discrete algorithms, Atlanta, pp 75–84
6. Blum A, Raghavan P, Schieber B (1997) Navigating in unfamiliar geometric terrain. SIAM J Comput 26(1):110–137
7. Deng X, Kameda T, Papadimitriou C (1998) How to learn an unknown environment I: the rectilinear case. J ACM 45(2):215–245
8. Dudek G, Jenkin M (2010) Computational principles of mobile robotics, 2nd edn. Cambridge University Press, New York
9. Dynia M, Łopuszański J, Schindelhauer C (2007) Why robots need maps. In: SIROCCO '07: proceedings of the 14th colloquium on structural information an communication complexity, Şirince. Lecture notes in computer science. Springer, pp 37–46
10. Fiat A, Woeginger G (eds) (1998) On-line algorithms: the state of the art. Lecture notes in computer science, vol 1442. Springer, Berlin/New York
11. Fleischer R, Kamphans T, Klein R, Langetepe E, Trippen G (2008) Competitive online approximation of the optimal search ratio. SIAM J Comput 38(3):881–898
12. Fraigniaud P, Gasieniec L, Kowalski DR, Pelc A (2006) Collective tree exploration. Networks 43(3):166–177
13. Gal S (1980) Search games. Mathematics in science and engeneering, vol 149. Academic, New York
14. Gal S, (2011) Search games. In: Wiley encyclopedia of operations research and management science, John Wiley & Sons, Inc.
15. Hagius R, Icking C, Langetepe E (2004) Lower bounds for the polygon exploration problem. In: Abstracts 20th European workshop computational geometry, Universidad de Sevilla, Sevilla, pp 135–138
16. Hoffmann F, Icking C, Klein R, Kriegel K (2001) The polygon exploration problem. SIAM J Comput 31:577–600

17. Kao MY, Reif JH, Tate SR (1996) Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. Inf Comput 133(1):63–79
18. Karloff H, Rabani Y, Ravid Y (1994) Lower bounds for randomized k-server and motion-planning algorithms. SIAM J Comput 23:293–312
19. Koutsoupias E, Papadimitriou CH, Yannakakis M (1996) Searching a fixed graph. In: Proceedings of the 23th international colloquium on automata language, and programming, Paderborn. Lecture notes in computer science, vol 1099, Springer, pp 280–289
20. Langetepe E (2010) On the optimality of spiral search. In: SODA 2010: proceedings of the 21st annual ACM-SIAM symposium on discrete algorithms, Austin, pp 1–12
21. López-Ortiz A, Schuierer S (2002) Online parallel heuristics and robot searching under the competitive framework. In: Proceedings of the 8th Scandinavian workshop on algorithm theory, Turku. Lecture notes in computer science, vol 2368. Springer, pp 260–269
22. Lumelsky VJ, Stepanov AA (1987) Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica 2:403–430
23. Mitchell JSB (2000) Geometric shortest paths and network optimization. In: Sack JR, Urrutia J (eds) Handbook of computational geometry. Elsevier Science/North-Holland, Amsterdam, pp 633–701
24. Papadimitriou CH, Yannakakis M (1991) Shortest paths without a map. Theor Comput Sci 84(1):127–150
25. Rao NSV, Kareti S, Shi W, Iyengar SS (1993) Robot navigation in unknown terrains: introductory survey of non-heuristic algorithms. Technical report, ORNL/TM-12410, Oak Ridge National Laboratory

Robust Bin Packing

Kim-Manuel Klein
University Kiel, Kiel, Germany

Keywords

Approximation; Bin packing; Competitive ratio; Migration factor; Online; Robust

Years and Authors of Summarized Original Work

2009; Epstein, Levin
2013; Jansen, Klein

Problem Definition

Consider the classical online bin packing problem, where items of sizes in $(0, 1]$ arrive over time. At the arrival of each item, it has to be assigned to a bin of capacity 1 such that the total size of all items in the bin does not exceed its capacity. The objective is to minimize the number of used bins.

Online bin packing was introduced by Ullman [10] and has seen enormous research since then (see the survey of Seiden [9] for an overview). The quality of an online algorithm is typically measured by the asymptotic performance guarantee of the algorithm divided by the optimal offline solution and is called the (asymptotic) *competitive ratio*. In the case of online bin packing, the best known algorithm has an asymptotic competitive ratio of 1.58889 (see [9]). On the other hand, it was shown that no algorithm can achieve a ratio better than 1.54037 (see [1]).

To obtain algorithms with improved competitive ratio for online bin packing, one can allow to rearrange already packed items as soon as a new item arrives. The notion of robustness allows to repack a set of already packed items with limited total size whenever a new item arrives. On the one hand, we want to guarantee that we use as few bins as possible, and on the other hand, when a new item arrives, we want to minimize the total size of repacked items.

A modern way to measure the repacking costs is the notion of the *migration factor*, developed by Sanders, Sivadasan, and Skutella [8]. It is defined by the total size of all moved items divided by the size of the arriving item. Following the notation of Sanders et al., an online algorithm with (asymptotic) approximation ratio $1 + \epsilon$ is called *robust* if its migration factor is of the size $f(\frac{1}{\epsilon})$, where f is an arbitrary function that only depends on $\frac{1}{\epsilon}$.

Key Results

In the case of robust bin packing, Epstein and Levin [3] proved that the asymptotic competitive

ratio of the robust bin packing problem can be arbitrarily close to the optimum. They developed an asymptotic PTAS for the problem using a migration factor of $2^{\mathcal{O}(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})}$. They also proved that there is no online algorithm for this problem which has a constant migration factor and that maintains an optimal solution. The asymptotic PTAS by Epstein and Levin was later improved by Jansen and Klein [7], who developed an asymptotic FPTAS for the problem with a migration factor of $\mathcal{O}(\frac{1}{\epsilon^4})$.

Techniques

Most robust algorithms rely on a sensitivity result for integer linear programs (ILPs) by Cook et al. [2]. It was first used by Sanders et al. [8] to develop a robust PTAS for the scheduling problem on identical machines with the objective value of minimizing the makespan. The theorem of Cook et al. roughly states that for every optimal integral solution y' of the ILP $\min\{cy \mid Ax \leq b'\}$, there exists an optimal integral solution y'' of the ILP $\min\{cy \mid Ax \leq b''\}$ with changed right-hand side b'' such that the distance between y' and y'' can be bounded by $\|y'' - y'\|_\infty \leq n\Delta\,(\|b'' - b'\|_\infty + 2)$, where n is the number of variables and Δ is the absolute value of the largest subdeterminant of A.

The major contribution by Epstein and Levin for the robust bin packing problem was to develop a dynamic rounding technique. Based on a classical rounding by Fernandez de La Vega and Lueker [5], the dynamic rounding techniques present a way on how item sizes can be rounded in a setting where new items arrive over time. This allows to formulate an ILP of fixed dimension. As a new item arrives online, the formulated ILP changes accordingly. The changed ILP has additional columns and the right-hand side of the ILP is increased. Using the theorem of Cooks et al. [2] allows then to find a solution for the changed ILP that is close to the existing solution. This way a new packing is constructed for the bin packing instance containing the newly arrived item.

Since the number of variables n and the largest subdeterminant Δ in the ILP formulation can only be bounded by an exponential term in $\frac{1}{\epsilon}$, the use of Cooks et al. theorem leads to an exponential migration factor. Jansen and Klein developed new LP and ILP techniques which are based on approximate solutions of the corresponding LP. Their central idea is to show that for any approximate solution x' with objective value $\|x'\|_1 \leq (1+\delta)\text{LIN}$, there is an approximate solution x'' with improved objective value $\|x'\|_1 \leq (1+\delta)\text{LIN} - 1$ such that $\|y'' - y'\|_1 = \mathcal{O}(\frac{1}{\delta})$. Based on this observation, they can avoid the use of Cooks et al. theorem to obtain an asymptotic PTAS for the bin packing problem with polynomial migration.

Open Problems

- There is the obvious open question on how much the migration factor of $\mathcal{O}(\frac{1}{\epsilon^4})$ from [7] can be improved and whether there are lower bounds for the migration factor. The existence of a robust algorithm with constant or sublinear migration is still open.
- Is there a robust approximation scheme for the case when items not only arrive but also depart? In the literature this problem is called fully dynamic bin packing and was considered by Ivković and Lloyd [6]. They developed an algorithm which achieves an asymptotic competitive ratio of $\frac{5}{4}$ using amortized $\mathcal{O}(\log n)$ shifting moves, where n is the number of packed items. A shifting move repacks one large item or a bundle of small items of bounded size.
- Epstein and Levin developed a robust asymptotic PTAS for the generalized bin packing problem, where d-dimensional cubes have to be packed into unit-sized cubes [4]. It would be interesting to find other robust approximation schemes for other packing problems like online strip packing or online bin packing with bins of different capacities.

Cross-References

▸ Approximation Schemes for Bin Packing
▸ Robust Scheduling Algorithms

Recommended Reading

1. Balogh J, Békési J, Galambos G (2010) New lower bounds for certain classes of bin packing algorithms. In: Workshop on approximation and online algorithms (WAOA), Liverpool. LNCS, vol 6534, pp 25–36
2. Cook W, Gerards A, Schrijver A, Tardos E (1986) Sensitivity theorems in integer linear programming. Math Program 34(3):251–264
3. Epstein L, Levin A (2009) A robust APTAS for the classical bin packing problem. Math Program 119(1):33–49
4. Epstein L, Levin A (2013) Robust approximation schemes for cube packing. SIAM J Optim 23(2):1310–1343
5. Fernandez de la Vega W, Lueker G (1981) Bin packing can be solved within $1 + \epsilon$ in linear time. Combinatorica 1(4):349–355
6. Ivković Z, Lloyd E (1998) Fully dynamic algorithms for bin packing: being (mostly) myopic helps. SIAM J Comput 28(2):574–611
7. Jansen K, Klein K (2013) A robust AFPTAS for online bin packing with polynomial migration. In: International colloquium on automata, languages, and programming (ICALP), Riga, pp 589–600
8. Sanders P, Sivadasan N, Skutella M (2009) Online scheduling with bounded migration. Math Oper Res 34(2):481–498
9. Seiden S (2002) On the online bin packing problem. J ACM 49(5):640–671
10. Ullman J (1971) The performance of a memory allocation algorithm. Technical report, Princeton University

Robust Geometric Computation

Chee K. Yap and Vikram Sharma
Department of Computer Science, New York University, New York, NY, USA

Keywords

Exact geometric computation Floating-point filter; Dynamic and static filters; Topological consistency

Years and Authors of Summarized Original Work

2004; Li, Yap

Problem Definition

Algorithms in computational geometry are usually designed under the Real RAM model. In implementing these algorithms, however, fixed-precision arithmetic is used in place of exact arithmetic. This substitution introduces numerical errors in the computations that may lead to nonrobust behavior in the implementation, such as infinite loops or segmentation faults.

There are various approaches in the the literature addressing the problem of nonrobustness in geometric computations; see [9] for a survey. These approaches can be classified along two lines: the **arithmetic approach** and the **geometric approach**.

The arithmetic approach tries to address nonrobustness in geometric algorithms by handling the numerical errors arising because of fixed-precision arithmetic; this can be done, for instance, by using multi-precision arithmetic [6], or by using rational arithmetic whenever possible. In general, all the arithmetic operations, including exact comparison, can be performed on algebraic quantities. The drawback of such a general approach is its inefficiency.

The geometric approaches guarantee that certain geometric properties are maintained by the algorithm. For example, if the Voronoi diagram of a planar point set is being computed then it is desirable to ensure that the output is a planar graph as well. Other geometric approaches are finite resolution geometry [7], approximate predicates and fat geometry [8], consistency and topological approaches [4], and topology oriented approach [13]. The common drawback of these approaches is that they are problem or algorithm specific.

In the past decade, a general approach called the **Exact Geometric Computation** (EGC) [15] has become very successful in handling the issue of nonrobustness in geometric computations; strictly speaking, this approach is subsumed in the arithmetic approaches. To understand the EGC approach, it helps to understand the two parts common to all geometric computations: a *combinatorial structure* characterizing the discrete relations between geometric objects, e.g.,

whether a point is on a hyperplane or not; and a *numerical part* that consists of the numerical representation of the geometric objects, e.g., the coordinates of a point expressed as rational or floating-point numbers. Geometric algorithms characterize the combinatorial structure by numerically computing the discrete relations (that are embodied in geometric predicates) between geometric objects. Nonrobustness arises when numerical errors in the computations yield an incorrect characterization. The EGC approach ensures that all the geometric predicates are evaluated correctly thereby ensuring the correctness of the computed combinatorial structure and hence the robustness of the algorithm.

Notation

An **expression** E refers to a syntactic object constructed from a given set of operators over the reals $\mathbb{R}$. For example, the set of expressions on the set of operators $\{\mathbb{Z}, +, -, \times, \sqrt{}\}$ is the set of division-free radical expressions on the integers; more concretely, expressions can be viewed as directed acyclic graphs (DAG) where the internal nodes are operators with arity at least one, and the leaves are constants, i.e., operators with arity zero. The value of an expression is naturally defined using induction; note that the value may be undefined. Let E represent both the value of the expression and the expression itself.

Key Results

Following are the key results that have led to the feasibility and success of the EGC approach.

Constructive Zero Bounds

The possibility of EGC approach hinges on the computability of the sign of an expression. For determining the sign of algebraic expressions EGC libraries currently use a numerical approach based upon zero bounds. A **zero bound** $b > 0$ for an expression E is such that absolute value $|E|$ of E is greater than b if the value of E is valid and nonzero. To determine the sign of the expression E, compute an approximation $\tilde{E}$ to E such that $|\tilde{E} - E| < \frac{b}{2}$ if E is valid, otherwise $\tilde{E}$ is also invalid. Then sign of E is the same as the sign of $\tilde{E}$ if $|\tilde{E}| \geq \frac{b}{2}$, otherwise it is zero. A **constructive zero bound** is an effectively computable function B from the set of expressions to real numbers $\mathbb{R}$ such that $B(E)$ is a zero bound for any expression E. For examples of constructive zero bounds, see [2, 11].

Approximate Expression Evaluation

Another crucial feature in developing the EGC approach is developing algorithms for approximate expression evaluation, i.e., given an expression E and a relative or absolute precision p, compute an approximation to the value of the expression within precision p. The main computational paradigm for such algorithms is the **precision-driven approach** [15]. Intuitively, this is a downward-upward process on the input expression DAG; propagate precision values down to the leaves in the downward direction; at the leaves of the DAG, assume the ability to approximate the value associated with the leaf to any desired precision; finally, propagate the approximations in the upward direction towards the root. Ouchi [10] has given detailed algorithms for the propagation of "composite precision", a generalization of relative and absolute precision.

Numerical Filters

Implementing approximate expression evaluation requires multi-precision arithmetic. But efficiency can be gained by exploiting machine floating-point arithmetic, which is fast and optimized on current hardware. The basic idea is to to check the output of machine evaluation of predicates, and fallback on multi-precision methods if the check fails. These checks are called numerical filters; they certify certain properties of computed numerical values, such as their sign. There are two main classifications of numerical filters: *static filters* are those that can be mostly computed at compile time, but they yield overly pessimistic error bounds and thus are less effective; *dynamic filters* are implemented during run time and even though they have higher costs they are much more effective than static

filters, i.e., have better estimate on error bounds. See Fortune and van Wyk [5].

Applications

The EGC approach has led to the development of libraries, such as LEDA Real and CORE, that provide EGC number types, i.e., a class of expressions whose signs are guaranteed. CGAL, another major EGC Library that provides robust implementation of algorithms in computational geometry, offers various specialized EGC number types, but for general algebraic numbers it can also use LEDA Real or CORE.

Open Problems

1. An important challenge from the perspective of efficiency for EGC approach is high degree algebraic computation, such as those found in Computer Aided Design. These issues are beginning to be addressed, for instance [1].
2. The *fundamental problem of EGC* is the **zero problem**: given any set of real algebraic operators, decide whether any expression over this set is zero or not. The main focus here is on the decidability of the zero problem for non-algebraic expressions. The importance of this problem has been highlighted by Richardson [12]; recently some progress has been made for special non-algebraic problems [3].
3. When algorithms in EGC approach are embedded in larger application systems (such as mesh generation systems), the output of one algorithm needs to be cascaded as input to another; the output of such algorithms may be in high precision, so it is desirable to reduce the precision in the cascade. The geometric version of this problem is called the **geometric rounding problem**: given a consistent geometric object in high precision, "round" it to a consistent geometric object at a lower precision.
4. Recently a computational model for the EGC approach has been proposed [14]. The corresponding complexity model needs to be developed. Standard complexity analysis based on input size is inadequate for evaluating the complexity of real computation; the complexity should be expressed in terms of the output precision.

URL to Code

1 `Core Library:` http://www.cs.nyu.edu/exact
2 `LEDA:` http://www.mpi-sb.mpg.de/LEDA
3 `CGAL:` http://www.cgal.org

Recommended Reading

1. Berberich E, Eigenwillig A, Hemmer M, Hert S, Schmer KM, Schmer E (2002) A computational basis for conic arcs and boolean operations on conic polygons. In: 10th European symposium on algorithms (ESA'02). Lecture notes in computer science, vol 2461, pp 174–186
2. Burnikel C, Funke S, Mehlhorn K, Schirra S, Schmitt S (2001) A separation bound for real algebraic expressions. In: Lecture notes in computer science, vol 2161. Springer, pp 254–265
3. Chang EC, Choi SW, Kwon D, Park H, Yap C (2006) Shortest paths for disc obstacles is computable. Int J Comput Geom Appl 16(5–6):567–590. In: Gao XS, Michelucci D (eds) Special issue on geometric constraints. Also appeared in Proceedings of the 21st ACM symposium on computational geometry, pp 116–125 (2005)
4. Fortune SJ (1989) Stable maintenance of point-set triangulations in two dimensions. IEEE Found Comput Sci 30:494–499
5. Fortune SJ, van Wyk CJ (1993) Efficient exact arithmetic for computational geometry. In: Proceedings of the 9th ACM symposium on computational geometry, pp 163–172
6. Gowland P, Lester D (2000) A survey of exact arithmetic implementations. In: Blank J, Brattka V, Hertling P (eds) Computability and complexity in analysis. 4th International workshop, CCA 2000, selected papers. Lecture notes in computer science, No. 2064. Springer, Swansea, 17–19 Sept 2000, pp 30–47
7. Greene DH, Yao FF (1986) Finite-resolution computational geometry. IEEE Found Comput Sci 27: 143–152
8. Guibas L, Salesin D, Stolfi J (1989) Epsilon geometry: building robust algorithms from imprecise computations. ACM Symp Comput Geom 5:208–217
9. Li C, Pion S, Yap CK (2004) Recent progress in exact geometric computation. J Log Algebra Program 64(1):85–111

10. Ouchi K (1997) Real/expr: implementation of an exact computation package. Master's thesis, Department of Computer Science, Courant Institute, New York University. Jan 1997. http://cs.nyu.edu/exact/doc/
11. Pion S, Yap C (2002) Constructive root bound method for k-ary rational input numbers, Sept 2002. Extended abstract. Submitted (2003) ACM Symposium on Computational Geometry
12. Richardson D (1997) How to recognize zero. J Symb Comput 24:627–645
13. Sugihara K, Iri M, Inagaki H, Imai T (2000) Topology-oriented implementation – an approach to robust geometric algorithms. Algorithmica 27:5–20
14. Yap CK (2006) Theory of real computation according to EGC. To appear in LNCS Volume based on talks at a Dagstuhl Seminar "Reliable implementation of real number algorithms: theory and practice", Jan 8–13 2006
15. Yap CK, Dubé T (1995) The exact computation paradigm. In: Du DZ, Hwang FK (eds) Computing in euclidean geometry, 2nd edn. World Scientific Press, Singapore, pp 452–492

Robust Scheduling Algorithms

José Verschae
Departamento de Matemáticas and
Departamento de Ingeniería Industrial y de Sistemas, Pontificia Universidad Católica de Chile, Santiago, Chile

Keywords

Load balancing; Makespan minimization; Online scheduling; Recourse; Robustness

Years and Authors of Summarized Original Work

2004, 2009; Sanders, Sivadasan, Skutella

Problem Definition

In the classic online scheduling model, jobs arrive one after another. At the arrival of a new job, the scheduler must immediately and irrevocably assign it to a machine. In the *parallel machine* case, we have m identical machines to process the jobs. Each job j has a processing time p_j that is revealed at the moment of its appearance. The load of a machine is the sum of processing times of jobs assigned to it. The objective is to minimize the makespan, that is, the maximum machine load.

The fact that decisions are irrevocable imposes a hard constraint on the scheduler. However, many applications allow some amount of flexibility. *Robust scheduling* algorithms take this flexibility into account: whenever a job arrives, some reassignment of jobs can be performed. More precisely, given a parameter $\beta > 0$, the arrival of job j allows to migrate a set of jobs with a total processing time of at most $\beta \cdot p_j$. The factor β is called the *migration factor* of the algorithm and it is a measure of its robustness. In this context, the quality of solutions is assessed by competitive analysis: an algorithm is α-competitive if for any sequence of job arrivals the makespan of the algorithm is at most α times the (offline) optimum cost for the set of available jobs. An important goal in this area is to understand the trade-off between the migration and competitive factors.

Key Results

Greedy Approaches

In a setting where no migration is allowed, i.e., if $\beta = 0$, a competitive ratio of $2 - \frac{1}{m}$ is achievable by a greedy *list-scheduling* algorithm [7]. Although more sophisticated algorithms have smaller competitive ratios (see, e.g., [6]), no algorithm can achieve a performance guarantee smaller than $e/(e-1) \approx 1.58$ [1, 11], even if randomization is allowed.

Sanders et al. [10] give algorithms with improved competitive ratios for small values of β. A simplified version of their most basic algorithm is as follows. Let j be an arriving job and denote by OPT the minimum makespan of the instance including j. The algorithm works as follows.

1. If $p_j \leq \mathrm{OPT}/2$, assign job j to the machine with the smallest load.
2. Otherwise, consider a machine i in which all jobs are of size at most $\mathrm{OPT}/2$. Greedily remove jobs from i until their total processing

time is at least p_j. Add job j to i and greedily reassign each removed job to the least loaded machine.

The existence of the claimed machine in Step (2) follows since there can be at most m jobs of size larger than OPT/2 in the instance. The proof that the algorithm is 3/2-competitive is a simple exercise that follows since each greedily assigned job has size at most OPT/2. By construction the algorithm has migration factor 2. The fact that the algorithm needs the value OPT as input can be avoided by trying out a handful of different solutions.

With this simple approach the competitive guarantee is already below the lower bound of 1.58 for $\beta = 0$. Sanders et al. [10] shows that a refinement of this algorithm gives the same competitive guarantee and reduces β to 4/3. They also provide more sophisticated algorithms with smaller competitive factors, for example, a 4/3-competitive algorithm with migration factor 5/2.

Robust Approximation Schemes

The algorithms above show that already small migration factors can help to significantly improve the quality of solutions. However, they tell little about the trade-off between the competitive and migration ratios. Sanders et al. [10] study this trade-off by giving a *robust polynomial time approximation scheme* (robust PTAS), that is, a family of algorithms $\{A_\varepsilon\}_{\varepsilon>0}$ such that for any constant $\varepsilon > 0$ the algorithm A_ε is $(1+\varepsilon)$-competitive and uses a migration factor of $\beta(\varepsilon)$. We remark that β is a constant that depends only on ε and not on the specific input data.

The robust PTAS borrows ideas from the known PTAS for the offline problem [8]. At the arrival of a job j, the algorithm takes the given $(1+\varepsilon)$-approximate solution and updates it to a schedule with the same approximation guarantee. The algorithm behaves differently depending on the size of j. If p_j is in $O(\varepsilon\text{OPT})$, where OPT denotes the optimal makespan for the current instance including j, then we can safely assign this job to the least loaded machine and maintain the approximation guarantee. Otherwise, the processing times are rounded to simplify the instance and add symmetry to the solution. The corresponding offline minimum makespan problem for this instance can be posed as an integer program (IP) of the form $\min\{c^t x : A \cdot x = b, x \in \mathbb{N}^d\}$. A component x_C of x corresponds to the number of machines with a given *configuration* C, where each configuration is a compact description of a one machine schedule. Crucially, the number of different configurations, that is, the number of variables d in the IP, is a constant $2^{\text{poly}(1/\varepsilon)}$. Moreover, the complete instance is encoded in the right-hand side b. After a new job arrives, the corresponding IP can be updated by increasing one coordinate of b by one, obtaining a new vector b'. A sensitivity analysis result by Cook et al. [2] implies that for any optimal solution x of the IP, there exists an optimal solution x' with the right-hand side changed to b' such that $||x - x'||_1 \leq d^2 \cdot \Delta(A) \cdot (||b - b'||_\infty + 2)$. Here $\Delta(A)$ is the maximum $|\det(B)|$ over all square submatrices B of A, which in this case can be bounded by $2^{\text{poly}(1/\varepsilon)}$. Therefore, the number of machines that need to be modified in order to go from schedule x to x' is $||x - x'||_1 \leq 2^{\text{poly}(1/\varepsilon)}$. Since we are assuming that the new job has processing time in $\Omega(\varepsilon\text{OPT})$, and each machine has a load of $O(\text{OPT})$, we obtain an algorithm with migration factor $2^{\text{poly}(1/\varepsilon)}$.

Theorem 1 (Sanders et al. [10]) *The problem of minimum makespan on identical machines admits a robust PTAS with migration factor* $\beta = 2^{poly(1/\varepsilon)}$.

Applications

The basic technique for constructing a robust PTAS has been adapted to different related problem. Most results are based on the sensitivity analysis result mentioned above, but differ in other parts of the algorithm and analysis. In particular robust PTASs have been developed for bin packing [3] and cube packing [4]. Other objective functions for identical machine scheduling have also been considered, for example, minimizing the ℓ_p-norm of the vector of loads or maximizing

the minimum machine load. These problems do not admit robust PTASs; however, it is possible to design such algorithms for an amortized analogue of the migration factor [12]. Epstein and Levin [5] consider preemptive scheduling problems on parallel machines, obtaining a 1-competitive algorithm with migration factor $1 - 1/m$ for the minimum makespan and minimum ℓ_p-norm objectives. As opposed to the previous results, this algorithm does not rely on sensitivity analysis results for IPs.

The bin packing problem was considered by Jansen and Klein [9]. They improve the result in [3] to a robust PTAS with $\beta = \text{poly}(1/\varepsilon)$. To obtain this migration factor, they develop new sensitivity analysis results aimed specifically at approximation algorithms.

Open Problems

An interesting question is to determine the precise trade-off between the competitive and migration factors for the minimum makespan problem on identical machines. In particular, determine if the migration can be made to depend polynomially on $1/\varepsilon$ for a $(1 + \varepsilon)$-competitive algorithm. Another natural question is to extend these results to *related* machines. In this setting, each machine i runs at a speed s_i, and thus the time it takes to process job j on machine i is p_j/s_i.

Other natural objective functions on parallel machine scheduling are not fully understood. The machine covering version, where we seek to maximize the minimum machine load, does not admit a robust PTAS. More specifically, a competitive ratio smaller than $20/19$ is not possible with constant migration factor [12]. It is open if this competitive ratio is indeed achievable or if the lower bound can be improved. A similar situation holds for minimizing the ℓ_p-norm for any $p > 1$ [4].

Cross-References

- ▸ Approximation Schemes for Bin Packing
- ▸ Bin Packing
- ▸ List Scheduling
- ▸ Online Load Balancing of Temporary Tasks

Recommended Reading

1. Chen B, van Vliet A, Woeginger GJ (1994) A lower bound for randomized on-line scheduling algorithms. Inf Process Lett 51:219–222
2. Cook W, Gerards AMH, Schrijver A, Tardos É (1986) Sensitivity theorems in integer linear programming. Math Program 34:251–264
3. Epstein L, Levin A (2009) A robust APTAS for the classical bin packing problem. Math Program 119:33–49
4. Epstein L, Levin A (2013) Robust approximation schemes for cube packing. SIAM J Optim 23:1310–1343
5. Epstein L, Levin A (2014) Robust algorithms for preemptive scheduling. Algorithmica 69:26–57
6. Fleischer R, Wahl M (2000) On-line scheduling revisited. J Sched 3:343–353
7. Graham RL (1966) Bounds for certain multiprocessing anomalies. Bell Syst Tech J 45:1563–1581
8. Hochbaum DS, Shmoys DB (1987) Using dual approximation algorithms for scheduling problems: theoretical and practical results. J ACM 34:144–162
9. Jansen K, Klein KM (2013) A robust AFPTAS for online bin packing with polynomial migration. In: Fomin FV, Freivalds R, Kwiatkowska M, Peleg D (eds) Automata, languages, and programming (ICALP 2013). Lecture notes in computer science, vol 7965. Springer, Berlin/Heidelberg, Riga, Latvia pp 589–600
10. Sanders P, Sivadasan N, Skutella M (2009) Online scheduling with bounded migration. Math Oper Res 34:481–498
11. Sgall J (1997) A lower bound for randomized on-line multiprocessor scheduling. Inf Process Lett 63:51–55
12. Skutella M, Verschae J (2010) A robust PTAS for machine covering and packing. In: Berg M, Meyer U (eds) Algorithms – ESA 2010. Lecture notes in computer science, vol 6346. Springer, Berlin/Heidelberg, Liverpool, UK pp 36–47

Robustness in Self-Assembly

Ho-Lin Chen
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan

Keywords

Error correction; Proofreading

Years and Authors of Summarized Original Work

2004; Winfree, Bekbolatov
2004; Chen, Goel

Robustness in Self-Assembly

The abstract tile assembly model (aTAM), originally proposed by Winfree [1], provides a useful framework to study algorithmic tile self-assembly. As described in other sections, many theoretical studies have shown the efficiency and computational power of aTAM.

The aTAM, although widely accepted and experimentally verified, is an overly simplified combinatorial model in describing the self-assembly of DNA tiles. In reality, several effects are observed which lead to a loss of robustness compared to the aTAM. The assembly tends to be reversible, i.e., tiles can fall off from an existing assembly, even when the total binding strength exceeds the temperature threshold τ. Also, tiles sometimes attach with a weak strength but then quickly get incorporated and locked into a growing assembly, much like defects in a crystal. However, for sophisticated combinatorial assemblies like counters, which form the basis for controlling the size of a structure, a single error can lead to assemblies drastically larger or smaller (or different in other ways) than the intended structure. An error rate of 0.5–10 % is observed in previous experimental studies.

KTAM

A more sophisticated and accurate stochastic model called the kinetic tile assembly model (kTAM) was introduced by Winfree [1]. The kTAM calculates rates for various types of attachments and removals based on thermodynamic constants. It has the following assumptions:

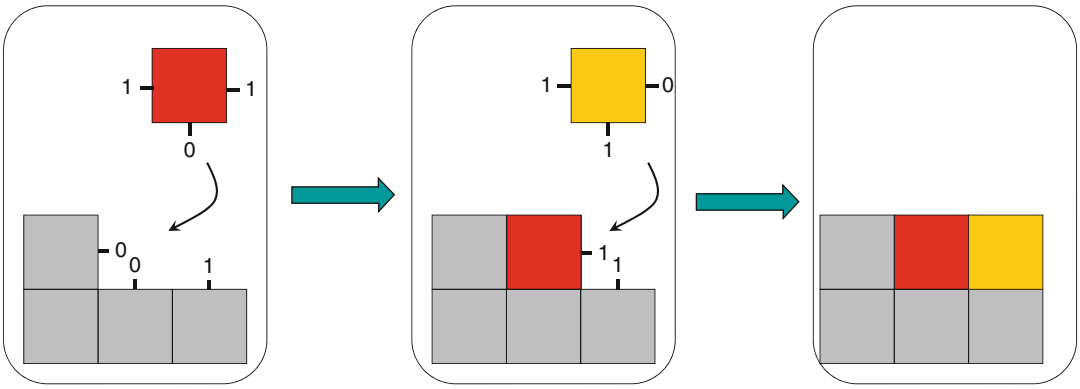
1. Tile concentrations are held constant throughout the self-assembly process.
2. Supertiles do not interact with each other. The only two reactions allowed are addition of a tile to a supertile and the dissociation of a tile from a supertile.
3. The forward rate constants for all tiles only depend on concentrations.
4. The reverse rate depends exponentially on the number of base-pair bonds which must be broken, and the mismatched sticky ends make no base-pair bonds.

There are two free parameters in this model, both of which are dimensionless free energies: $G_{mc} > 0$ measures the entropic cost of putting a tile at a binding site and depends on the tile concentration, and $G_{se} > 0$ measures the free energy cost of breaking a single strength-1 bond. Under this model, we can approximate the forward and reverse rates for each of the tile-supertile reactions in the process of self-assembly of DNA tiles as follows:

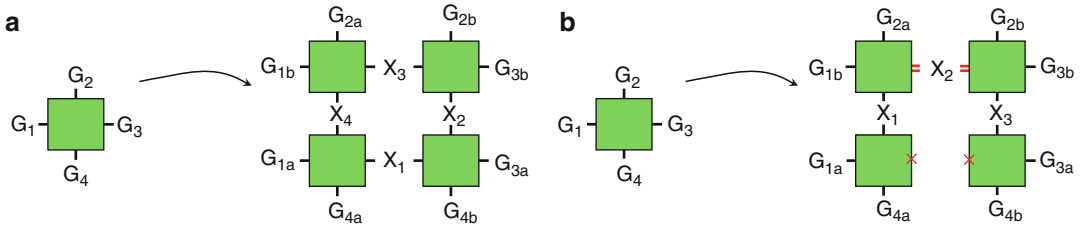
The rate of addition of a tile to a supertile, f, is $pe^{-G_{mc}}$. The rate of dissociation of a tile from a supertile, r_b, is $pe^{-bG_{se}}$, where b is the strength with which the tile is attached to the supertiles. The parameter p simply gives us the time scale for the self-assembly. Winfree showed that by setting appropriate tile concentrations and binding strengths such that $G_{mc} = 2G_{sc} - \epsilon$, the behavior predicted by kTAM approaches the behavior described by aTAM as $\epsilon \to 0$. However, the growth speed also goes to 0 (attachment and dissociation form an unbiased random walk) as $\epsilon \to 0$.

Problem Definition

Self-assembly processes in nature are often equipped with explicit mechanisms for both error prevention and error correction. For artificial self-assembly, these problems are even more important since we are interested in assembling large systems with great precision. Previously, a phenomenon called *insufficient attachments* has been identified to be a main source of error [3]. An insufficient attachment is the process in which a tile t first attach with total strength less than the temperature. However, before t falls off, adjacent tiles attach and secure t in place. An insufficient



Robustness in Self-Assembly, Fig. 1 An example of growth error caused by an insufficient attachment. The *red tile* first attaches with a weaker strength. Then the *yellow tile* attaches and secures the *red tile* in place



Robustness in Self-Assembly, Fig. 2 (**a**) An example of a 2×2 proofreading block. (**b**) An example of a 2×2 snaked proofreading block

attachment happening at a location at which another correct tile can attach via cooperative bindings is called a *growth error*. One example of growth error caused by insufficient attachment is illustrated in Fig. 1. An insufficient attachment happening at a location at which no tiles can attach according to aTAM is called a *facet error*. The rate at which any specific insufficient attachment happens is $e^{-2G_{mc}}/(e^{-G_{mc}} + e^{-G_{se}})$, which is roughly $e^{-G_{se}}$ times the rate of tile attachments when $G_{mc} \approx 2G_{sc}$. The main goal of this section is to introduce error-correction systems that deal with insufficient attachments.

Key Results

Proofreading Tilesets

The first error-correction scheme for the tile assembly model was the proofreading scheme proposed by Winfree and Bekbolatov [2]. The proofreading scheme turns any tile system with unidirectional growth into a new system that produces the same pattern (with scaling). The scheme replaces each tile type t by k^2 distinct tile types. These k^2 tile types are designed to form a $k \times k$ block. All internal glues have strength 1 and are unique to this block. The glues on the boundary of the block are duplicates of glues on the original tile t. A 2×2 block is illustrated in Fig. 2a. This construction enforces that whenever a growth error happens, at least k growth errors must happen locally in order for the assembly process to proceed. Thus when growth error happens, the erroneous tiles are more likely to detach than to stay and wait for another $k - 1$ growth errors to happen. Since the probability that a growth error happens at any given location is roughly $e^{-G_{se}}$, the probability that a growth error happens at any proofreading block and proceeds to produce the final incorrect assembly is $O(e^{-kG_{se}})$.

Snaked Proofreading Tilesets

The abovementioned proofreading system only handles growth errors but not facet errors, which

are a major source of error both in theoretical analysis and computer simulations [3]. Chen and Goel [3] proposed the snaked proofreading scheme which handles all errors caused by insufficient attachments. Similar to the proofreading scheme, the snaked proofreading scheme replaces each tile type by a block of $2k \times 2k$ tile types. The only difference is that some internal glues have strength 0 or 2 instead of 1. A 2×2 block is illustrated in Fig. 2b. Under correct growth, the snaked proofreading block checks its two input sides alternatively. Therefore, k insufficient attachments must happen before an erroneous block "thinks" it attaches correctly and propagate toward any single direction. As a result, the snaked proofreading scheme with block size $2k \times 2k$ ensures that without k insufficient attachments happening locally, all erroneous structures have $O(k^2)$ tiles and are expected to fall off in time polynomial in k. Assuming that the thermodynamic parameters G_{mc} and G_{se} can be set arbitrarily, the following theorem characterizes the performance of the snaked proofreading system.

Theorem 1 *With a $2k \times 2k$ snaked tile system, $k = O(\log n)$, an $n \times n$ square of blocks can be assembled in expected time $\tilde{O}(n)$ and with high probability, while remaining stable for $\tilde{\Omega}(n)$ time after being assembled.*

One main drawback for the proofreading and the snaked proofreading scheme is the resolution loss. Since each tile in the original system is replaced by a $k \times k$ block, the size of the original pattern is increased by a factor of k. Chen, Goel, and Luhrs [4] showed that if the third dimension can be used, two-dimensional tile systems can be proofread with no resolution loss. Their system replaces each tile by a column in the third dimension and thus maintains the original scale on the two-dimensional plane.

Recommended Reading

1. Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology
2. Winfree E, Bekbolatov R (2004) Proofreading tile sets: error correction for algorithmic self-assembly. In: DNA computers 9. LNCS, vol 2943. Springer, Berlin/Heidelberg, pp 126–144
3. Chen HL, Goel A (2004) Error free self-assembly using error prone tiles. In: Tenth international meeting on DNA computing, Milano
4. Chen HL, Goel A, Luhrs C (2008) Dimension augmentation and combinatorial criteria for efficient error-resistant DNA self-assembly. In: ACM-SIAM symposium on discrete algorithms, San Francisco

Routing

József Békési and Gábor Galambos
Department of Computer Science, Juhász Gyula Teachers Training College, Szeged, Hungary

Keywords

Network flows; Oblivious routing; Routing algorithms

Years and Authors of Summarized Original Work

2003; Azar, Cohen, Fiat, Kaplan, Räcke

Problem Definition

One of the most often used techniques in modern computer networks is routing. *Routing* means selecting paths in a network along which to send data. Demands usually randomly appear on the nodes of a network, and routing algorithms should be able to send data to their destination. The transfer is done through intermediate nodes, using the connecting links, based on the topology of the network. The user waits for the network to guarantee that it has the required capacity during data transfer, meaning that the network behaves like its nodes would be connected directly by a physical line. Such service is usually called the *permanent virtual circuit (PVC)* service. To model real life situations, assume that demands arrive *on line*, given by source and destination points, and capacity (bandwidth) requirements.

Similar routing problems may occur in other environments, for example in parallel computation. In this case there are several processors connected together by wires. During an operation some data appear at given processors which should be sent to specific destinations. Thus, this also defines a routing problem. However, this paper mainly considers the network model, not the parallel computer one.

For any given situation there are several routing possibilities. A natural question is to ask which is the best possible algorithm. To find the best algorithm one must define an objective function, which expresses the effectiveness of the algorithm. For example, the aim may be to reduce the load of the network. Load can be measured in different ways, but to measure the utilization percent of the nodes or the links of the network is the most natural. In the online setting, it is interesting to compare the behavior of a routing algorithm designed for a specific instance to the best possible routing.

There are two fundamental approaches towards routing algorithms. The first approach is to route *adaptively*, i.e., depending on the actual loads of the nodes or the links. The second approach is to route *obviously*, without using any information about the current state of the network. Here the authors survey only results on oblivious routing algorithms.

Notations and Definitions

A mathematical model of the network routing problem is now presented.

Let $G(V, E, c)$ be a capacitated network, where V is the set of nodes and E is the set of edges with a capacity function $c : E \to R^+$. Let $|V| = n, |E| = m$. It can be assumed that G is directed, because if G is undirected then for each undirected edge $e = (u, v)$ two new nodes x, y and four new directed edges $e_1 = (u, x), e_2 = (v, x), e_3 = (y, u), e_4 = (y, u)$ with infinite capacity may be added to the graph. If e is considered as an undirected edge with the same capacity then a directed network equivalent to the original one is received.

Definition 1 A set of functions $f := \{f_{ij} | i, j \in V, f_{ij} : E(G) \to R^+\}$ is called a **multi–commodity flow** if

$$\sum_{e \in E_k^+} f_{ij}(e) = \sum_{e \in E_k^-} f_{ij}(e)$$

holds for all $k \neq i, k \neq j$, where $k \in V$ and E_k^+, E_k^- are the set of edges coming out from k and coming into k resp. Each function f_{ij} defines a **single–commodity flow** from i to j.

Definition 2 The **value** of a multi–commodity *flow* is an $n \times n$ matrix $T_f = (t_{ij}^f)$, where

$$t_{ij}^f = \sum_{e \in E_i^+} f_{ij}(e) - \sum_{e \in E_i^-} f_{ij}(e) \,,$$

$$\text{if } i \neq j \text{ and } v_{ii}^f = 0 \,, \text{ forall } i, j \in V \,.$$

Definition 3 Let D be a nonnegative $n \times n$ matrix where the diagonal entries are 0. D is called as **demand matrix**. The **flow** on an edge $e \in E$ routing the demand matrix D by routing r is defined by

$$\text{flow}(e, r, D) = \sum_{i,j \in V} d_{ij} r_{ij}(e) \,,$$

while the **edge congestion** is

$$\text{con}(e, r, D) = \frac{\text{flow}(e, r, D)}{c(e)} \,.$$

The **congestion** of demand D using routing r is

$$\text{con}(r, D) = \max_{e \in E} \text{con}(e, r, D) \,.$$

Definition 4 A multi–commodity flow r is called **routing** if $t_{ij}^r = 1$, and if $i \neq j$ for all $i, j \in V$.

Routing represents a way of sending information over a network. The real load of the edges can be represented by scaling the edge congestions with the demands.

Definition 5 The **oblivious performance ratio** P_r of routing r is

$$P_r = \sup_D \left\{ \frac{\text{con}(r, D)}{\text{opt}(D)} \right\}$$

where *opt*(*D*) is the optimal congestion which can be achieved on *D*. The **optimal oblivious routing ratio** for a network *G* is denoted by *opt*(*G*), where

$$\text{opt}(G) = \min_r P_r$$

Problem
Input: A capacitated network $G(V, E, c)$.
Output: An oblivious routing *r*, where P_r is minimal.

Key Results

Theorem 1 *There is a polynomial time algorithm that for any input network G (directed or undirected) finds the optimal oblivious routing ratio and the corresponding routing r.*

Theorem 2 *There is a directed graph G of n vertices such that opt(G) is at least* $\Omega(\sqrt{n})$.

Applications

Most importantly, with these results one can efficiently calculate the best routing strategy for a network topology with capacity constraints. This is a good tool for network planning. The effectiveness of a given topology can be tested without any knowledge of the the network traffic using this analysis.

Many researchers have investigated the variants of routing problems. For surveys on the most important models and results, see [10] and [11]. Oblivious routing algorithms were first analyzed by Valiant and Brebner [15]. Here, they considered the parallel computer model and investigated specific architectures, like hypercube, square grids, etc. Borodin and Hopcroft investigated general networks [6]. They showed that such simple deterministic strategies like oblivious routing can not be very efficient for online routing and proved a lower bound on the competitive ration of oblivious algorithms. This lower bound was later improved by Kaklamanis et al. [9], and they also gave an optimal oblivious deterministic algorithm for the hypercube.

In 2002, Räcke constructed a polylog competitive randomized algorithm for general undirected networks. More precisely, he proved that for any demand there is a routing such that the maximum edge congestion is at most polylog(n) times the optimal congestion for this demand [12]. The work of Azar et al. extends this result by giving a polynomial method for calculating the optimal oblivious routing for a network. They also prove that for directed networks no logarithmic oblivious performance ratio exists. Recently, Hajiaghayi et al. present an oblivious routing algorithm which is $O\left(\log^2 n\right)$-competitive with high probability in directed networks [8].

A special online model has been investigated in [5], where the authors define the so called "repeated game" setting, where the algorithm is allowed to chose a new routing in each day. This means that it is oblivious to the demands, that will occur the next day. They present an $1 + \varepsilon$-competitive algorithm for this model.

There are better algorithms for the adaptive case, for example in [2]. For the offline case Raghavan and Thomson gave an efficient algorithm in [13].

Open Problems

The authors investigated edge congestion in this paper, but in practice, node congestion may be interesting as well. Node congestion means the ratio of the total traffic traversing a node to its capacity. Some results can be found for this problem in [7] and in [3]. It is an open problem whether this method used for edge congestion analysis can be applied for such a model. Another interesting open question may be whether there is a more efficient algorithm to compute the optimal oblivious performance ratio of a network [1, 14].

Experimental Results

The authors applied their method on ISP network topologies and found that the calculated optimal oblivious ratios are surprisingly low, between 1.4 and 2. Other research dealing with this question found similar results [1, 14].

Cross-References

- Direct Routing Algorithms
- Mobile Agents and Exploration
- Oblivious Routing
- Online Load Balancing of Temporary Tasks
- Probabilistic Data Forwarding in Wireless Sensor Networks

Recommended Reading

1. Applegate D, Cohen E (2006) Making routing robust to changing traffic demands: algorithms and evaluation. IEEE/ACM Trans Networking 14(6):1193–1206. doi:10.1109/TNET.2006.886296
2. Aspnes J, Azar Y, Fiat A, Plotkin S, Waarts O (1997) On-line routing of virtual circuits with applications to load balancing and machine scheduling. J ACM 44(3):486–504
3. Azar Y, Chaiutin Y (2006) Optimal node routing. In: Proceedings of the 23rd international symposium on theoretical aspects of computer science, pp 596–607
4. Azar Y, Cohen E, Fiat A, Kaplan H, Räcke H (2003) Optimal oblivious routing in polynomial time. In: Proceedings of the thirty-fifth annual ACM symposium on theory of computing, pp 383–388
5. Bansal N, Blum A, Chawla S, Meyerson A (2003) Online oblivious routing. In: Proceedings of the 15th annual ACM symposium on parallel algorithms, pp 44–49
6. Borodin A, Hopcroft JE (1985) Routing, merging and sorting on parallel models of computation. J Comput Syst Sci 30(1):130–145
7. Hajiaghayi MT, Kleinberg RD, Leighton T, Räcke H (2005) Oblivious routing on node-capacitated and directed graphs. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms, pp 782–790
8. Hajiaghayi MT, Kim JH, Leighton T, Räcke H (2005) Oblivious routing in directed graphs with random demands. In: Proceedings of the 37th annual ACM symposium on theory of computing, pp 193–201
9. Kaklamanis C, Krizanc D, Tsantilas A (1990) Tight bounds for oblivious routing in the hypercube. In: Proceedings of the 2nd annual ACM symposium on parallel algorithms and architectures, pp 31–36
10. Leighton FT (1992) Introduction to parallel algorithms and architectures arrays, trees, hypercubes. Morgan Kaufmann Publishers, San Fransisco
11. Leonardi S (1998) On-line network routing, Chapter 11. In: Fiat A, Woeginger G (eds) Online algorithms – the state of the art. Springer, Heidelberg, pp 242–267
12. Räcke H (2002) Minimizing congestions in general networks. In: Proceedings of the 43rd symposium on foundations of computer science, pp 43–52
13. Raghavan P, Thompson CD (1987) Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica 7:365–374
14. Spring N, Mahajan R, Wetherall D (2002) Measuring ISP topologies with Rocket fuel. In: Proceedings of the ACMSIGCOMM'02 conference. ACM, New York
15. Valiant LG, Brebner G (1981) Universal schemes for parallel communication. In: Proceedings of the 13th ACM symposium on theory of computing, pp 263–277

Routing in Geometric Networks

Stephane Durocher[1], Leszek Gąsieniec[2], and Prudence W.H. Wong[2]
[1]University of Manitoba, Winnipeg, MB, Canada
[2]University of Liverpool, Liverpool, UK

Synonyms

Geographic routing; Location-based routing

Keywords

Face routing; Geometric routing; Unit disk graph; Wireless communication

Years and Authors of Summarized Original Work

1999; Kranakis, Singh, Urrutia
1999; Bose, Morin, Stojmenovic, Urrutia
2003; Kuhn, Wattenhofer, Zhang, Zollinger

Problem Definition

Wireless networks are often modelled using geometric graphs. Using only local geometric information to compute a sequence of distributed forwarding decisions that send a message to its destination, routing algorithms can succeed on several common classes of geometric graphs. These graphs' geometric properties provide navigational cues that allow routing to succeed using only limited local information at each node.

Network Model

A common geometric graph model for wireless networks is to represent each node by a point in the Euclidean plane, $\mathcal{R}^2$, and to add an edge (u, v) for each pair of nodes that can communicate by direct wireless transmission. The absence of the edge (u, v) signifies that u cannot transmit directly to v, requiring a multi-hop transmission via a sequence of intermediate nodes that forms a route from u to v. The cost $c(e)$ of sending a message over an edge $e = (u, v)$ has been modeled in different ways; the most common measures include the *hop (link) metric* ($c(e) = 1$), the *Euclidean metric* ($c(e) = |e|$, where $|e| = \text{dist}(u, v)$ is the Euclidean length of the edge e), and the *energy metric* ($c(e) = |e|^\alpha$ for $\alpha \geq 2$).

In some models, transmission is assumed to be uniform in all directions and of equal range, say r, for all nodes. Under this assumption, the undirected edge (u, v) exists if and only if $\text{dist}(u, v) \leq r$. Thus, for each node v there is an edge from v to every node u that lies within a disk of radius r centered at v. This is the *unit disk graph* model for wireless networks. Common classes of geometric graphs that are used to model wireless networks include:

Unit Disk Graph. Vertices are points in $\mathcal{R}^2$ and each edge (u, v) exists if and only if $\text{dist}(u, v) \leq r$, for a given fixed $r > 0$.

Plane Graph. Vertices are points in $\mathcal{R}^2$ and no two edges cross.

Triangulation. Vertices are points in $\mathcal{R}^2$ and every interior face is a triangle.

Quasi-unit Disk Graph. Vertices are points in $\mathcal{R}^2$ and each edge (u, v) exists if $\text{dist}(u, v) \leq r_1$, may exist if $r_1 < \text{dist}(u, v) \leq r_2$, but does not exist if $\text{dist}(u, v) > r_2$, for given fixed $r_2 > r_1 > 0$.

Unit Ball Graph. Vertices are points in $\mathcal{R}^3$ and each edge (u, v) exists if and only if $\text{dist}(u, v) \leq r$, for a given fixed $r > 0$.

Gabriel Graph. Vertices are points in $\mathcal{R}^2$ and each edge (u, v) exists if and only if the disk with diameter (u, v) does not contain any other vertices.

Other classes of geometric graphs used to model wireless networks include relative neighborhood graphs, Delaunay triangulations, Yao graphs, convex subdivisions, monotone subdivisions, edge-augmented plane graphs, and physically based models such as SINR.

A geometric graph G is *civilized* with λ-*precision* if for every pair of nodes u and v in G, $\text{dist}(u, v) \geq \lambda$ for a given fixed $\lambda > 0$, where λ is independent of n, the number of nodes in G.

Communication Protocol

In several wireless network protocols, e.g., ad hoc or wireless sensor networks, there is no fixed infrastructure for routing nor any central servers. All nodes act as hosts as well as routers. Apart from a node's immediate neighborhood, the topology of the network is unknown, i.e., each node is aware of its own location (its (x, y) coordinates) as well as the coordinates of its neighbors. Nodes must discover and maintain routes in a distributed manner without knowledge of precomputed routing tables, any particular vertex labeling (other than spatial coordinates), nor the support of a central server. Additionally, some models incorporate constraints for limited memory and power. Depending on the particular model, a limited amount of information can be stored in message headers to assist with routing. When a node receives a message, it reads the header (possibly modifying the header information) before selecting one of its neighbors to which to forward the message. A *stateless* algorithm does not modify the header. Network nodes have no memory themselves; any dynamic state information is stored in the message header.

Furthermore, no precomputed information about the network is known to the nodes.

Geometric Routing

Given the coordinates of a target node t in a (wireless) geometric network G, a source node s in G is tasked with sending a message via a multi-hop route through G from s to t. Routing proceeds by computing a sequence of distributed forwarding decisions, where each node along the route selects one of its neighbors to which to forward the message. Geometric routing is uniform in that all nodes execute the same protocol. Each node makes a forwarding decision as a function of its coordinates, the coordinates of its neighbors, the coordinates of t, and any available state bits stored in the message header. The number of state bits available is critical to guaranteeing delivery in some classes of geometric graphs by enabling the route to avoid looping and reach t. A node may modify the state bits before forwarding the message. In some models, this state information corresponds to storing data about $O(1)$ nodes, e.g., storing the coordinates of $O(1)$ nodes.

The primary objective is to guarantee message delivery to the target node t. Secondary objectives include minimizing the total cost of communication (the sum of $c(e)$ for all edges e on the route) and minimizing the worst-case or average *dilation* (the ratio of the cost of the route followed relative to that of the route of lowest cost). These secondary objectives are motivated by the need for nodes to conserve power in many wireless networking settings.

Key Results

Local geometric routing assumes only limited control information stored in message headers and local information available at each node along the route. This locality provides network independence that results in natural scalability to larger networks and continued functionality after arbitrary changes to the network. A routing algorithm is said to *succeed* on a particular class of geometric graphs if it guarantees delivery from any source node s to any target node t on any graph in the class; otherwise, the algorithm *fails* on that class of graphs.

Below we summarize key local geometric routing algorithms and their properties.

Greedy Routing. Upon receiving a message, a node forwards it to its neighbor closest to the target node t. Greedy routing is stateless. This strategy succeeds on Delaunay triangulations, but fails on more general classes of geometric graphs such as non-Delaunay triangulations, convex subdivisions, plane graphs, and unit disk graphs.

Compass Routing [7]. Upon receiving a message, a node u forwards it to its neighbor v that minimizes the angle $\angle vut$ with the target node t. Compass routing is stateless. This strategy succeeds on regular triangulations but fails on more general classes of geometric graphs such as non-regular triangulations, convex subdivisions, plane graphs, and unit disk graphs.

Greedy-Compass Routing [2]. Upon receiving a message, a node u considers its two neighbors on either side of the line segment $\overline{ut}$ (node u's *compass neighbors*) and forwards the message to the one closest to t. Greedy-compass routing is stateless. This strategy succeeds on all triangulations but fails on more general classes of geometric graphs such as convex subdivisions, plane graphs, and unit disk graphs.

Bose et al. [2] show that no stateless algorithm can succeed on convex subdivisions (including plane graphs and unit disk graphs). Therefore, to succeed on classes of geometric graphs beyond triangulations, local routing algorithms require storing one or more state bits in the message header or predecessor information, i.e., the coordinates of the node that last forwarded the message.

One State Bit [4]. Upon receiving a message, a node u chooses between forwarding the message to its clockwise or counter-clockwise compass neighbor, depending on the value of a state bit. If the compass neighbor lies opposite the vertical line through t, the state bit is flipped. This

algorithm uses a single state bit. This strategy succeeds on all triangulations and convex subdivisions, but fails on more general classes of geometric graphs such as plane graphs and unit disk graphs.

Predecessor Awareness and Monotonicity [4]. Each node locally identifies its topmost left neighbor as its parent and its right neighbors as its children. With knowledge of the predecessor, the node forwards the message to its $(i+1)$st child after receiving it from its ith child and eventually back to its parent after receiving it from its last child. The resulting route contains a depth-first traversal of a spanning tree of the network. This algorithm is stateless, but each node requires knowledge of its predecessor, i.e., the coordinates of the node that last forwarded the message. This strategy succeeds on triangulations, convex subdivisions, monotone subdivisions, and edge-augmented graphs from these classes but fails on more general classes of geometric graphs such as non-monotone plane graphs and unit disk graphs.

Face Routing [1, 7]. The message is forwarded along the perimeters of faces in the sequence of faces that intersect the line segment from the source node s to the target node t. This strategy applies the *right-hand principle*, in which each face in the sequence is traversed in a counterclockwise direction, as if one were walking while sliding the right hand along the wall. To avoid cycling indefinitely, the algorithm must store the coordinates of $O(1)$ nodes that act as progress markers. Furthermore, each node requires knowledge of its predecessor. This strategy succeeds on plane graphs, including triangulations, convex subdivisions, and Gabriel graphs. The intersection of a unit disk graph with the Gabriel graph of a set of points is planar and remains connected if the original unit disk graph is connected. Furthermore, this subgraph can be computed locally; this property allows face routing to succeed on unit disk graphs [1], as well as quasi-unit disk graphs with bounded ratio $r_2/r_1 < \sqrt{2}$ and unit ball graphs contained within slabs of thickness less than $1/\sqrt{2}$ [6]. Although unit disk graphs are nonplanar in general, the nonplanarity is localized; face routing fails on more general classes of nonplanar geometric graphs such as quasi-unit disk graphs and unit ball graphs [6] and edge-augmented plane graphs. Face routing can have dilation $\Theta(n)$, where n is the number of network nodes.

Adaptive Face Routing (AFR) [8]. Adaptive face routing is a variant of face routing that achieves optimality on civilized unit disk graphs and civilized planar graphs with the Gabriel property. Like face routing, $O(1)$ state data are stored in the message header and each node requires knowledge of its predecessor. The algorithm attempts to estimate the length c of the shortest path from s and t by $\hat{c}$ (starting with $\hat{c} = 2|\overline{st}|$ and doubling it in every consecutive round). In each round, the face traversal is restricted to the region formed by the ellipse with the major axis $\hat{c}$ centered on $\overline{st}$. Each edge is traversed at most four times, and the dilation achieved is $\Theta(c)$.

Geometric Ad-hoc Routing (GOAFR$^+$) [9]. Combining methods from greedy routing, face routing, and adaptive face routing allows this hybrid algorithm to meet the bounds of adaptive routing on any unit disk graphs and planar graphs with the Gabriel property (not necessarily civilized). The algorithm first applies greedy routing and switches to face routing when the routed message enters a local minimum (a dead end), before again resuming greedy routing as early as possible by applying an *early fallback* technique.

General (Non-geometric) Networks

Is geometry necessary for local routing to succeed? Even with knowledge of the predecessor, stateless routing algorithms require knowledge of the induced subgraph of nodes up to distance $n/3$ away in the worst case [3]. That is, stateless routing using only *local* information is impossible. With $\Theta(\log n)$ state bits, local routing on arbitrary (not necessarily geometric) graphs is possible by deterministically recomputing a polynomial-length universal traversal sequence at each node along the route, where $\Theta(\log n)$ bits store an index into the sequence [5].

Open Problems

If a node's coordinates can be stored using $O(\log n)$ bits (e.g., if network nodes are positioned on a $n^c \times n^c$ grid), then face routing can be applied using $O(\log n)$ state bits. It remains open whether any local geometric routing algorithm can succeed on plane graphs using $o(\log n)$ state bits. Similarly, it would be interesting to characterize broad classes of geometric graphs on which local geometric routing is possible using $O(1)$ state bits. In addition to guaranteeing delivery, bounding dilation is of interest. For example, can $O(1)$ dilation be guaranteed on convex subdivisions using $O(1)$ state bits? Finally, the problem of traversing a graph (visiting all nodes) by a sequence of local forwarding decisions is interesting. Stateless algorithms are impossible for any non-Hamiltonian network. How many state bits are necessary for a local algorithm to traverse a triangulation?

Cross-References

- ▸ Communication in Ad Hoc Mobile Networks Using Random Walks
- ▸ Geographic Routing
- ▸ Local Computation in Unstructured Radio Networks
- ▸ Minimum k-Connected Geometric Networks

Recommended Reading

1. Bose P, Morin P, Stojmenovic I, Urrutia J (1999) Routing with guaranteed delivery in ad hoc wireless networks. In: Proceedings of the third international workshop on discrete algorithm and methods for mobility, Seattle, Aug 1999, pp 48–55
2. Bose P, Brodnik A, Carlsson S, Demaine ED, Fleischer R, López-Ortiz A, Morin P, Munro I (2002) Online routing in convex subdivisions. Int J Comput Geom Appl 12(4):283–295
3. Bose P, Carmi P, Durocher S (2013) Bounding the locality of distributed routing algorithms. Distrib Comput 26(1):39–58
4. Bose P, Durocher S, Mondal D, Peabody M, Skala M, Wahid MA (2015) Local routing in convex subdivisions. In: Proceedings of the forty-first international conference on current trends in theory and practice of computer science, Pec pod Sněžkou, Jan 2015, vol 8939, pp 140–151
5. Braverman M (2008) On ad hoc routing with guaranteed delivery. In: Proceedings of the twenty-seventh ACM symposium on principles of distributed computing, Toronto, vol 27, p 418
6. Durocher S, Kirkpatrick DG, Narayanan L (2010) On routing with guaranteed delivery in three-dimensional ad hoc wireless networks. Wirel Netw 16(1):227–235
7. Kranakis E, Singh H, Urrutia J (1999) Compass routing on geometric networks. In: Proceedings of the eleventh Canadian conference on computational geometry, Vancouver, Aug 1999, pp 51–54
8. Kuhn F, Wattenhofer R, Zollinger A (2002) Asymptotically optimal geometric mobile ad-hoc routing. In: Proceedings of the sixth international workshop on discrete algorithm and methods for mobility, Atlanta, Sept 2002, pp 24–33
9. Kuhn F, Wattenhofer R, Zhang Y, Zollinger A (2003) Geometric ad-hoc routing: of theory and practice. In: Proceedings of the twenty-second ACM symposium on the principles of distributed computing, Boston, July 2003, pp 63–72

Routing in Road Networks with Transit Nodes

Dominik Schultes
Institute for Computer Science, University of Karlsruhe, Karlsruhe, Germany

Keywords

Shortest paths

Years and Authors of Summarized Original Work

2007; Bast, Funke, Sanders, Schultes

Problem Definition

For a given directed graph $G = (V, E)$ with non-negative edge weights, the problem is to compute a shortest path in G from a source node s to a target node t for given s and t. Under the assumption that G does not change and that a lot

of source-target queries have to be answered, it pays to invest some time for a preprocessing step that allows for very fast queries. As output, either a full description of the shortest path or only its length $d(s, t)$ is expected – depending on the application.

Dijkstra's classical algorithm for this problem [4] iteratively visits all nodes in the order of their distance from the source until the target is reached. When dealing with very large graphs, this general algorithm gets too slow for many applications so that more specific techniques are needed that exploit special properties of the particular graph. One practically very relevant case is routing in road networks where junctions are represented by nodes and road segments by edges whose weight is determined by some weighting of, for example, expected travel time, distance, and fuel consumption. Road networks are typically sparse (i.e., $|E| = O(|V|)$), almost planar (i.e., there are only a few overpasses), and hierarchical (i.e., more or less 'important' roads can be distinguished). An overview on various speedup techniques for this specific problem is given in [7].

Key Results

Transit-node routing [2, 3] is based on a simple observation intuitively used by humans: When you start from a source node s and drive to somewhere 'far away', you will leave your current location via one of only a few 'important' traffic junctions, called (forward) *access nodes* $\overrightarrow{A}(s)$. An analogous argument applies to the target t, i.e., the target is reached from one of only a few backward access nodes $\overleftarrow{A}(t)$. Moreover, the union of all forward and backward access nodes of all nodes, called *transit-node set* $\mathcal{T}$, is rather small. The two observations imply that for each node the distances to/from its forward/backward access nodes and for each transit-node pair (u, v), the distance between u and v can be stored. For given source and target nodes s and t, the length of the shortest path that passes at least one transit node is given by

$$d_{\mathcal{T}}(s,t) = \min\{d(s,u) + d(u,v) + d(v,t) \mid u \in \overrightarrow{A}(s), v \in \overleftarrow{A}(t)\}.$$

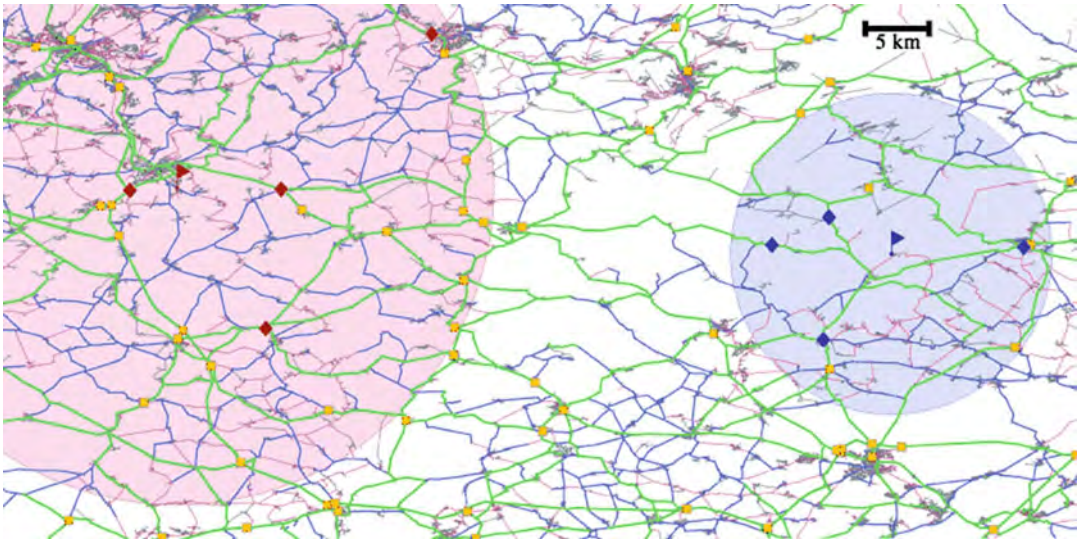
Note that all involved distances $d(s, u)$, $d(u, v)$, and $d(v, t)$ can be directly looked up in the precomputed data structures. As a final ingredient, a *locality filter* $L{:}V \times V \to \{\mathsf{true}, \mathsf{false}\}$ is needed that decides whether given nodes s and t are too close to travel via a transit node. L has to fulfill the property that $\neg L(s,t)$ implies that $d(s,t) = d_{\mathcal{T}}(s,t)$. Note that in general the converse need not hold since this might hinder an efficient realization of the locality filter. Thus, *false positives*, i.e., "$L(s,t) \wedge d(s,t) = d_{\mathcal{T}}(s,t)$", may occur.

The following algorithm can be used to compute $d(s, t)$:

> If $\neg L(s,t)$, then compute and return $d_{\mathcal{T}}(s,t)$; else, use any other routing algorithm.

Figure 1 gives an example. Knowing the length of the shortest path, a complete description of it can be efficiently derived using iterative table lookups and precomputed representations of paths between transit nodes. Provided that the above observations hold and that the percentage of false positives is low, the above algorithm is very efficient since a large fraction of all queries can be handled in line 1, $d_{\mathcal{T}}(s,t)$ can be computed using only a few table lookups, and source and target of the remaining queries in line 2 are quite close. Indeed, the remaining queries can be further accelerated by introducing a secondary layer of transit-node routing, based on a set of *secondary transit nodes* $\mathcal{T}_2 \supset \mathcal{T}$. Here, it is not necessary to compute and store a complete $\mathcal{T}_2 \times \mathcal{T}_2$ distance table, but it is sufficient to store only distances $\{d(u,v) \mid u, v \in \mathcal{T}_2 \wedge d(u,v) \neq d_{\mathcal{T}}(s,t)\}$, i.e., distances that cannot be obtained using the primary layer. Analogously, further layers can be added.

There are two different implementations: one is based on a simple geometric grid and one on *highway hierarchies*, the fastest previous approach [5, 6]. A highway hierarchy consists of a sequence of levels (Fig. 1), where level $i + 1$ is



Routing in Road Networks with Transit Nodes, Fig. 1 Finding the optimal travel time between two points (*flags*) somewhere between Saarbrücken and Karlsruhe amounts to retrieving the 2 × 4 *access nodes* (*diamonds*), performing 16 table lookups between all pairs of access nodes, and checking that the two disks defining the *locality filter* do not overlap. *Transit nodes* that do not belong to the access node sets of the selected source and target nodes are drawn as small squares. The figure draws the levels of the highway hierarchy using colors *gray*, *red*, *blue*, and *green* for levels 0–1, 2, 3, and 4, respectively

constructed from level i by bypassing low-degree nodes and removing edges that never appear far away from the source or target of a shortest path. Interestingly, these levels are geometrically decreasing in size and otherwise similar to each other. The highest level contains the most 'important' nodes and becomes the primary transit-node set. The nodes of lower levels are used to form the transit-node sets of subordinated layers.

Applications

Apart from the most obvious applications in car navigation systems and server-based route planning systems, transit-node routing can be applied to several other fields, for instance to massive traffic simulations and to various optimization problems in logistics.

Open Problems

It is an open question whether one can find better transit-node sets or a better locality filter so that the performance can be further improved. It is also not clear if transit-node routing can be successfully applied to other graph types than road networks. In this context, it would be desirable to derive some theoretical guarantees that apply to any graph that fulfills certain properties. For some practical applications, a *dynamic* version of transit-node routing would be required in order to deal with time-dependent networks or unexpected edge weight changes caused, for example, by traffic jams. The latter scenario can be handled by a related approach [8], which is, however, considerably slower than transit-node routing.

Experimental Results

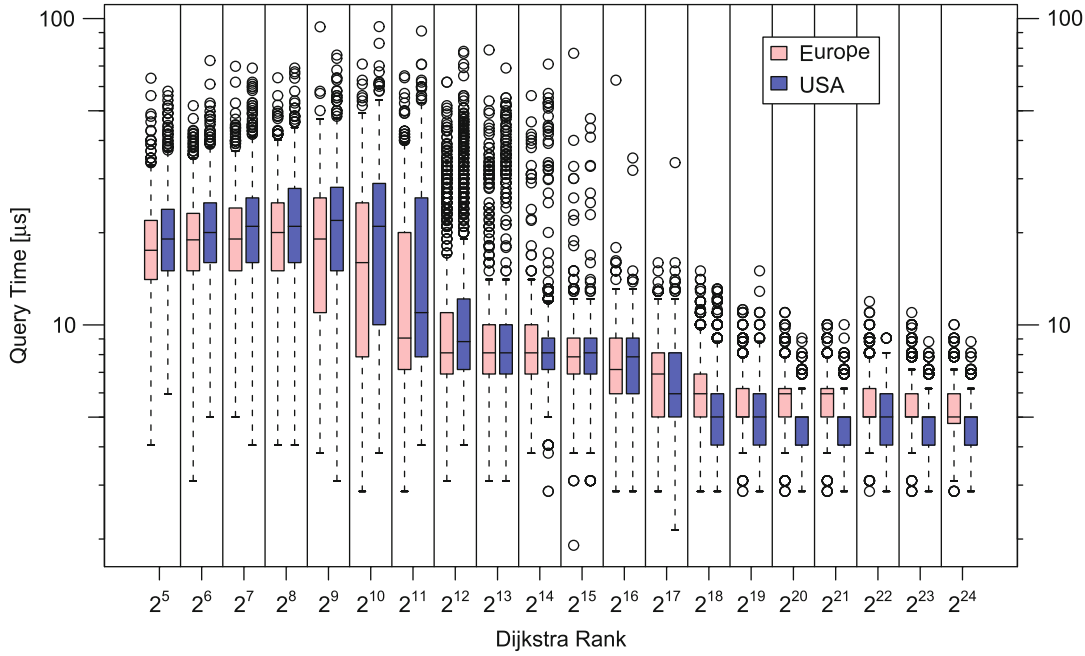
Experiments were performed on road networks of Western Europe and the USA using a cost function that solely takes expected travel time into account. The results exhibit various trade-offs between average query time (5–63 μs for the USA), preprocessing time (59 min to 1,200 min), and storage overhead (21 bytes/node to 244 bytes/node). For the variant that uses three

Routing in Road Networks with Transit Nodes, Table 1 Statistics on preprocessing. The size of transit-node sets, the number of entries in distance tables, and the average number of access nodes to the respective layer are given; furthermore, the space overhead and the preprocessing time

	Layer 1		Layer 2			Layer 3			
	$\lvert\mathcal{T}\rvert$	$\lvert A\rvert$ Avg.	$\lvert\mathcal{T}_2\rvert$	$\lvert\text{Table}_2\rvert$ [$\times 10^6$]	$\lvert A_2\rvert$ Avg.	$\lvert\mathcal{T}_3\rvert$	$\lvert\text{Table}_3\rvert$ [$\times 10^6$]	Space [B/node]	Time [h]
Europe	11,293	9.9	323,356	130	4.1	2,954,721	119	251	2:44
USA	10,674	5.7	485,410	204	4.2	3,855,407	173	244	3:25

Routing in Road Networks with Transit Nodes, Table 2 Performance of transit-node routing with respect to 10,000,000 random queries. The column for layer i specifies which fraction of the queries is correctly answered using only information available at layers $\leq i$. Each box spreads from the lower to the upper quartile and contains the median, the whiskers extend to the minimum and maximum value omitting outliers, which are plotted individually

	#nodes	#edges	Layer 1	Layer 2	Layer 3	Query
Europe	18 029 721	42 199 587	99.74 %	99.9984 %	99.99981 %	5.6 µs
USA	24 278 285	58 213 192	99.89 %	99.9986 %	99.99986 %	4.9 µs



Routing in Road Networks with Transit Nodes, Fig. 2 Query time distribution as a function of Dijkstra rank–the number of iterations Dijkstra's algorithm would need to solve this instance. The distributions are represented as box-and-whisker plots: each box spreads from the lower to the upper quartile and contains the median, the whiskers extended to the minimum and maximum value omitting, which are plotted individually

layers and is tuned for best query times, Tables 1 and 2 show statistics on the preprocessing and the query performance, respectively. The average query times of about 5 µs are six orders of magnitude faster than Dijkstra's algorithm. In addition, Fig. 2 gives for each rank r on the x-axis a distribution for 1,000 queries with random starting point s and the target node t for which Dijkstra's algorithm would need r iterations to find it. The three layers of transit-node routing with small transition zones in between can be recognized: for large ranks, it is sufficient to access only the

primary layer yielding query times of about 5 µs, for smaller ranks, additional layers have to be accessed resulting in median query times of up to 20 µs.

Data Sets

The European road network has been provided by the company PTV AG, the US network has been obtained from the TIGER/Line Files [9]. Both graphs have also been used in the 9th DIMACS Implementation Challenge on Shortest Paths [1].

URL to Code

The source code might be published at some point in the future at http://algo2.iti.uka.de/schultes/hwy/.

Cross-References

- All Pairs Shortest Paths in Sparse Graphs
- All Pairs Shortest Paths via Matrix Multiplication
- Decremental All-Pairs Shortest Paths
- Engineering Algorithms for Large Network Applications
- Fully Dynamic All Pairs Shortest Paths
- Geographic Routing
- Implementation Challenge for Shortest Paths
- Shortest Paths Approaches for Timetable Information
- Shortest Paths in Planar Graphs with Negative Weight Edges
- Single-Source Shortest Paths

Recommended Reading

1. 9th DIMACS implementation challenge: shortest paths (2006). http://www.dis.uniroma1.it/~challenge9/
2. Bast H, Funke S, Matijevic D, Sanders P, Schultes D (2007) In transit to constant time shortest-path queries in road networks. In: Workshop on algorithm engineering and experiments, pp 46–59
3. Bast H, Funke S, Sanders P, Schultes D (2007) Fast routing in road networks with transit nodes. Science 316(5824):566
4. Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1:269–271
5. Sanders P, Schultes D (2005) Highway hierarchies hasten exact shortest path queries. In: 13th European symposium on algorithms. LNCS, vol 3669. Springer, Berlin, pp 568–579
6. Sanders P, Schultes D (2006) Engineering highway hierarchies. In: 14th European symposium on algorithms. LNCS, vol 4168. Springer, Berlin, pp 804–816
7. Sanders P, Schultes D (2007) Engineering fast route planning algorithms. In: 6th workshop on experimental algorithms. LNCS, vol 4525. Springer, Berlin, pp 23–36
8. Schultes D, Sanders P (2007) Dynamic highway-node routing. In: 6th workshop on experimental algorithms. LNCS, vol 4525. Springer, Berlin, pp 66–79
9. U.S. Census Bureau, Washington, DC (2002) UA census 2000 TIGER/line files. http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html

Routing-Cost Constrained Connected Dominating Set

Hongwei Du and Haiming Luo
Department of Computer Science and Technology, Shenzhen Graduate School, Harbin Institute of Technology, Shenzhen, China

Keywords

Connected dominating set; Routing constraint; Wireless multihop networks; Wireless sensor networks

Years and Authors of Summarized Original Work

2011; Du, Ye, Wu, Lee, Li, Du, Howard
2012; Du, Ye, Zhong, Wang, Lee, Park
2013; Du, Wu, Ye, Li, Lee, Xu

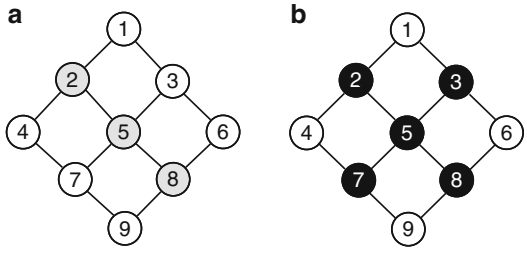
Problem Definition

Connected dominating set *CDS* is typically adapted in wireless multihop networks such as

wireless sensor, ad hoc networks. In order to achieve routing efficiency, a virtual backbone which is inspired by the backbone in wired networks is often used to improve routing because it can reduce the path search space and the routing table size [3]. According to [8, 10], there are many methods to construct a virtual backbone, and the competitive approach is connected dominating set(*CDS*). The detailed definition of *CDS* is as follows.

Given a connected graph $G(V, E)$ represents as a wireless sensor network, where V is the set of sensor nodes and E is the set of edges connecting sensor nodes in V. If there is a subset D($D \subseteq V$), each sensor node in V either belonging to D or adjacent to a sensor node in D, then we call D is a dominating set (*DS*). If the subgraph induced by *DS* is a connected graph, then we call *DS* is a connected dominating set(*CDS*).

Intuitively, if the size of *CDS* is smaller, the virtual backbone can play a greater role in routing. Many studies such as [1, 13, 14] and [9, 12] also aimed to construct a virtual backbone based on a *CDS* with minimum size which is called minimum connected dominating set (*MCDS*). A minimum CDS (MCDS) is a CDS that has the minimum number of nodes. For example, the gray nodes in Fig. 1a form an MCDS of the sample graph, while the black nodes in Fig. 1b make a CDS. However, these studies didn't ignore that if the size of the *CDS* is too small, some sensor nodes couldn't find a shortest routing path to their destination. Du et al. [10] analyzes how a virtual backbone based on *MCDS* makes some routing paths much longer than the shortest paths. Thus, there exists a disadvantage in *MCDS* which is the unavailability of shortest routing paths. If we only aim to construct a virtual backbone based on *MCDS*, we couldn't achieve a guaranteed routing constraint in data delivery. Routing constraint in *CDS* becomes much more important when constructing a virtual backbone.



Routing-Cost Constrained Connected Dominating Set, Fig. 1 (**a**) An example MCDS. (**b**) An example CDS

In this section, the problem of the routing constraint in connected dominating set (R-CDS) is given in formal by considering the wireless multihop network environment. The problem of R-CDS is defined as follows:

Given a graph $G = (V, E)$ where V represents a node set and E denotes an edge set, we would like to find a CDS D in polynomial time so that, for every pair of nodes u and v, there exists a path between u and v with intermediate nodes in D and path length at most $\alpha \cdot d(u, v)$, where α is a constant and $d(u, v)$ is the length of the shortest path between u and v. In addition, the size of the resulting CDS $|D|$ is bounded by $\beta \cdot opt_{MCDS}$, where β is a constant and opt_{MCDS} is the size of the MCDS.

The problem specified in wireless multihop networks has been considered under both general graph and unit disk graph(UDG) model and will be further discussed in the following section. With the UDG model, all nodes in the network have the same transmission range, and there does not exist any obstacle. As a result, as long as a receiving node is within the transmission range of a sending node, the receiving node will be able to receive the data successfully. With general graph model, the nodes in the network could have different transmission ranges, and obstacles might interfere with normal data communication. As a result, being in the transmission range of a sending node does not guarantee successful transmission.

Key Results

Many literatures also focus on the study of routing constraint in *CDS* which can be classified into two categories: the general graph and UDG.

In the general graph category, Ding et al. [2] studied a special connected dominating set (*CDS*) problem named minimum routing cost CDS

(*MOC-CDS*). They proved that constructing a minimum *MOC-CDS* in general graph is *NP*-hard and proposed a distributed heuristic algorithm (called as FlagContest) for constructing *MOC-CDS* with performance ratio $(1 - ln2) + 2ln\delta$. Du et al. [8] presented a constant-approximation scheme which produces a connected dominating set D, whose size |D| is within a factor α from that of the minimum connected dominating set, and each node pair exists in a routing path with all intermediate nodes in D and with length at most $5d(u, v)$, where $d(u, v)$ is the length of shortest path of this node pair. Ding et al. [3] developed an exact algorithm for minimum *CDS* with shortest path constraint called *SPCDS* and proved that finding such a minimum *SPCDS* can be achieved in polynomial time. Ding et al. [4] showed that under general graph model, α-*MOC-CDS* is NP-hard for any $\alpha \geq 1$. Ding et al. [5, 6] studied virtual backbone with guaranteed routing costs, named α minimum routing cost directional virtual backbone(α-*MOC-DVB*). They proved that the construction of a minimum α-*MOC-DVB* is an NP-hard problem in a general directed graph. Du et al. [10] proved that there is no polynomial-time constant approximation for α-*MOC-CDS* unless $P = NP$ when $\alpha \geq 2$.

In the UDG category, Wu et al. [15] studied the relationship between minimum connected dominating sets and maximal independent sets in unit disk graphs. Kim et al. [11] proposed a distributed algorithm under UDG model, *CDS-BD-D*, which constructs a *CDS* whose size and maximum path length are bounded. Du et al. [8] proposed two algorithms which are centralized algorithm and distributed algorithm to achieve constant-approximation performance ratio on MCDS and routing cost. Du et al. [7] studied a problem of minimizing the size of connected dominating set $|\ D\ |$ under constraint that for any two nodes u and v, $m_D(u, v) \leq \alpha m(u,\ v)$ where α is a constant, $m_D(u,\ v)$ is the number of intermediate nodes on a shortest path connecting u and v through D, and $m(u,\ v)$ is the number of intermediate nodes in a shortest path between u and v in a given unit disk graph.

In this chapter, we introduce that, under general graph model, there is no polynomial-time constant-approximation solution in terms of CDS size unless $P = NP$ shown in [10]. Under UDG model, we will present a polynomial-time constant-approximation algorithm GOC-MCDS-C which produces a CDS D with size $|D| \leq 176 \cdot opt_{MCDS} + 64$ and with a property that for any pair of nodes u and v, $d_D(u, v) \leq 7 \cdot d(u, v)$ in [10]. The distributed version of the algorithm, GOC-MCDS-D, is thoroughly analyzed.

Algorithm 1: Centralized algorithm GOC-MCDS-C

1: **Initially** Set $D \leftarrow \emptyset$.
2: **Step 1.** Construct a maximal independent set I.
3: **Step 2.** For every pair of nodes u, v in I with $d(u, v) \leq 3$, compute a shortest path $p(u, v)$ and put all intermediate nodes of $p(u, v)$ into C.
4: **Output** $D = C \cup I$.

Algorithm 2: Construct an MIS I (Stage 1)

1: **Initially** Every node is colored in white and is assigned with a positive integer ID; different nodes have different IDs.
2: **Step 1** Every white node sends its ID to its neighbors and then compares its ID with received IDs from neighbors. If its ID is smaller than every received ID from neighbors, then it turns the color from white to black.
3: **Step 2** Every black node sends message "black" to its neighbors. If a white node receives a message "black," then it turns its color from white to gray.
4: **Step 3** Go back to Step 1 until no white node exists.
5: **Output** All black nodes form a maximal independent set I

GOC-MCDS-C: The Centralized Algorithm

Under general graph model, the existing proof is that there is no polynomial-time constant approximation for the problem under investigation unless NP=P shown in [10]. However, under UDG model, polynomial-time constant-approximation algorithms do exist. Kim et al. [11] proposed a distributed algorithm, CDS-BD-D, that constructs a CDS whose size and maximum path length are bounded. In this section, we advance Kim et al.'s results by presenting the details of an innovative polynomial-time constant-approximation algorithm, GOC-MCDS-C. The proposed algorithm produces a

Algorithm 3: Connect the MIS I (Stage 2)

1: **Step 1** Every black node sends its ID to its neigh -bors.
2: **Step 2** Every node adds its own ID id_2 to each received ID id_1 and then sends those pairs of IDs (id_1, id_2) to all its neighbors.
3: **Step 3** Each node does the following: Suppose its ID is id^*:

1. For each pair of IDs id_1 and id_{1*} received in Step 1, if $id_1 < id_{1*}$, then send a message (id_{1*}, id^*, id_1) to the neighbor with ID id_1.
2. For each pair of messages (id_1, id_2) and (id_{1*}, id_{2*}) received at Step 2, if $id_1 < id_{1*}$, then send a message $(id_{1*}, id_{2*}, id^*, id_2, id_1)$ to the neighbor with ID id_2.
3. For each message (id_1, id_2) received at Step 2 and ID id_{1*} received at Step 1, if $id_1 < id_{1*}$, then send a message $(id_{1*}, id^*, id_2, id_1)$ to the neighbor with ID id_2; otherwise, send a message $(id_1, id_2, id^*, id_{1*})$ to the neighbor with ID id_{1*}.

4: **Step 4** When a node with ID id_2 received a message $(id_{1*}, id_{2*}, id^*, id_2, id_1)$ or $(id_{1*}, id^*, id_2, id_1)$, it sends this message to its neighbor with ID id_1.
5: **Step 5** Each black node with ID id_1 collects all messages in form (id_3, id_2, id_1) or (id_4, id_3, id_2, id_1) or $(id_5, id_4, id_3, id_2, id_1)$ received in Step 3 and Step 4. Suppose those messages form a set M. Then it performs the following computation:
while $M \neq \emptyset$ **do begin**
choose $(id_h, \ldots, id_2, id_1) \in M$;
send message $(id_h, \ldots, id_2, id_1)$ to node with ID id_2;
delete all messages starting with id_h from M;
end-while
6: **Step 6** When a node with ID id_i received a message $(\ldots, id_{i-1}, id_i, \ldots)$, it turns black. In addition, if id_i is not the leftmost id in the message, then it passes this message to node with ID id_{i-1}; if id_i is the leftmost id in the message, do nothing.
7: **Step 7** If no message is passed in Step 6, then stop. Otherwise, go back to Step 6.

CDS D with size $|D| \leq 176 \cdot opt_{MCDS} + 64$ and with a property that for any pair of nodes u and v, $d_D(u, v) \leq 7d(u, v)$[10]. Note that GOC-MCDS-C is a centralized algorithm. GOC-MCDS-C follows the steps of regular MCDS construction algorithms. Namely, there are two steps in total. During the first step, an MIS is constructed. In the second step, the nodes in the MIS are connected in order to form a CDS.

GOC-MCDS-D: The Distributed Algorithm

In this section, the distributed algorithm GOC-MCDS-D is described in details. The performance of GOC-MCDS-D is the same as that of GOC-MCDS-C shown in [10]. Similar to the centralized algorithm GOC-MCDS-C, GOC-MCDS-D consists of two stages. In the first stage, an MIS is constructed using Algorithm 2. In the second stage, the MIS is connected using Algorithm 3.

Open Problems

The coverage problems in wireless sensor networks which related to the routing-cost constrained in CDS are still an open problem.

Cross-References

▸ Strongly Connected Dominating Set

Recommended Reading

1. Cheng X, Ding M, Du DH, Jia X (2006) Virtual backbone construction in multihop ad hoc wireless networks. Wirel Commun Mobile Comput 6:183–190
2. Ding L, Gao X, Wu W, Lee W, Zhu X, Du D-Z (2010) Distributed construction of connected dominating sets with minimum routing cost in wireless network. In: Proceedings of the 30th international conference on distributed computing systems (ICDCS), Genova, pp 448–457
3. Ding L, Gao X, Wu W, Lee W, Zhu X, Du D-Z (2011) An exact algorithm for minimum CDS with shortest path constraint in wireless networks. J Optim Lett 5(2):297–306
4. Ding L, Wu W, Willson J, Du H, Lee W, Du D-Z (2011) Efficient algorithms for topology control problem with routing cost constraints in wireless networks. IEEE Trans Parallel Distrib Syst 22(10):1601–1609
5. Ding L, Wu W, Willson JK, Du H (2011) Construction of directional virtual backbones with minimum routing cost in wireless networks. In: Proceedings of 30th IEEE international conference on computer communications (INFOCOM), Shanghai

6. Ding L, Wu W, Willson J, Du H, Lee W (2012) Efficient virtual backbone construction with routing cost constraint in wireless networks using directional antennas. IEEE Trans Mobile Comput 11(7): 1102–1112
7. Du H, Ye Q, Zhong J, Wang Y, Lee W, Park H (2010) PTAS for minimum connected dominating set with routing cost constraint in wireless sensor networks. J Comb Optim Appl 6508:252–259
8. Du H, Ye Q, Wu W, Lee W, Li D, Du D, Howard S (2011) Constant approximation for virtual backbone construction with guaranteed routing cost in wireless sensor networks. In: Proceedings of the 30th IEEE international conference on computer communications (INFOCOM), Shanghai
9. Funke S, Kesselman A, Meyer U (2006) A simple improved distributed algorithm for minimum CDS in unit disk graphs. ACM Trans Sens Netw 2(3):444–453
10. Hongwei Du, Weili Wu, Qiang Ye, Deying Li, Wonjun Lee, Xuepeng Xu(2013) CDS-based virtual backbone construction with guaranteed routing cost in wireless sensor networks. IEEE Trans Parallel Distrib Syst 24(4):652–661
11. Kim D, Wu Y, Li Y, Zou F, Du D-Z (2009) Constructing minimum connected dominating sets with bounded diameters in wireless networks. IEEE Trans Parallel Distrib Syste 20(2):147–157
12. Li D, Du H, Wan P-J, Gao X, Zhang Z, Wu W (2009) Construction of strongly connected dominating sets in asymmetric multihop wireless networks. Theor Comput Sci 410(8–10): 661–669
13. Min M, Du H, Jia X, Huang CX, Huang SC-H, Wu W (2006) Improving construction for connected dominating set with Steiner tree in wireless sensor networks. J Glob Optim 35(1):111–119
14. Ruan L, Du H, Jia X, Wu W, Li Y, Ko K-I (2004) A greedy approximation for minimum connected dominating sets. Theor Comput Sci 329(1–3):325–330
15. Wu W, Du H, Jia X, Li Y, Huang SC-H (2006) Minimum connected dominating sets and maximal independent sets in unit disk graphs. Theor Comput Sci 352(1–3):1–7

R-Trees

Ke Yi
Hong Kong University of Science and Technology, Hong Kong, China

Keywords

External memory data structures; R-trees; Spatial databases

Years and Authors of Summarized Original Work

2004; Arge, de Berg, Haverkort, Yi

Problem Definition

Problem statement and the I/O model. Let S be a set of N axis-parallel hypercubes in $\mathbb{R}^d$. A very basic operation in a spatial database is to answer *window queries* on the set S. A window query Q is also an axis-parallel hypercube in $\mathbb{R}^d$ that asks us to return all hypercubes in S that intersect Q. Since the set S is typically huge in a large spatial database, the goal is to design a *disk-based* or *external memory* data structure (often called an *index* in the database literature) such that these window queries can be answered efficiently. In addition, given S, the data structure should be constructed efficiently and should be able to support insertions and deletions of objects.

When external memory data structures are concerned, the standard *external memory model* [2], a.k.a. the *I/O model*, is often used as the model of computation. In this model, the machine consists of an infinite-size external memory (disk) and a main memory of size M. A block of B consecutive elements can be transferred between main memory and disk in one *I/O operation* (or simply *I/O*). An external memory data structure is a structure that is stored on disk in blocks, but computation can only occur on elements in main memory, so any operation (e.g., query, update, and construction) on the data structure must be performed using a number I/Os, which is the measure for the complexity of the operation.

R-trees. The *R-tree*, first proposed by Guttman [9], is a multi-way tree $\mathcal{T}$, very similar to a *B-tree*, that is used to store the set S such that a window query can be answered efficiently. Each node of $\mathcal{T}$ fits in one disk block. The hypercubes of S are stored only in the leaves of $\mathcal{T}$. All leaves of $\mathcal{T}$ are on the same level, and each stores $\Theta(B)$ hypercubes from S; while

each internal node, except the root, has a fan-out of $\Theta(B)$. The root of $\mathcal{T}$ may have a fan-out as small as 2. For any node $u \in \mathcal{T}$, let $R(u)$ be the smallest axis-parallel hypercube, called the *minimal bounding box*, that encloses all the hypercubes stored below u. At each internal node $v \in \mathcal{T}$, whose children are denoted $v_1, \ldots, v_k$, the bounding box $R(v_i)$ is stored along with the pointer to v_i for $i = 1, \ldots, k$. Note that these bounding boxes may overlap. Please see Fig. 1 for an example of an R-tree in two dimensions.

For a window query Q, the query answering process starts from the root of $\mathcal{T}$ and visits all nodes u for which $R(u)$ intersects Q. When reaching a leaf v, it checks each hypercube stored at v to decide if it should be reported. The correctness of the algorithm is obvious, and the efficiency (the number of I/Os) is determined by the number of nodes visited.

Any R-tree occupies a linear number $O(N/B)$ disk blocks, but different R-trees might have different query, update, and construction costs. When analyzing the query complexity of window queries, the output size T is also used, in addition to N, M, and B.

Key Results

Although the structure of an R-tree is restricted, there is much freedom in grouping the hypercubes into leaves and grouping subtrees into bigger subtrees. Different grouping strategies result in different variants of R-trees. Most of the existing R-trees use various heuristics to group together hypercubes that are "close" spatially, so that a window query will not visit too many unnecessary nodes. Generally speaking, there are two ways to build an R-tree: repeated insertion and bulk loading. The former type of algorithms include the original R-tree [9], the R^+-tree [15], the R*-tree [6], etc. These algorithms use $O(\log_B N)$ I/Os to insert an object and hence $O(N \log_B N)$ I/Os to build the R-tree on S, which is not scalable for large N. When the set S is known in advance, it is much more efficient to bulk load the entire R-tree at once. Many bulk-loading algorithms have been proposed, e.g., [7, 8, 11, 13]. Most of these algorithms build the R-tree with $O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$ I/Os (the number of I/Os needed to sort N elements), and they typically result in better R-trees than those obtained by repeated insertion. During the past decades, there have been a large number of works devoted to R-trees from the database community, and the list here is by no means complete. The reader is referred to the book by Manolopoulos et al. [14] for an excellent survey on this subject in the database literature. However, no R-tree variant mentioned above has a guarantee on the query complexity; in fact, Arge et al. [3] constructed an example showing that some of the most popular R-trees may have to visit all the nodes without reporting a single result.

From the theoretical perspective, the following are the two main results concerning the worst-case query complexity of R-trees.

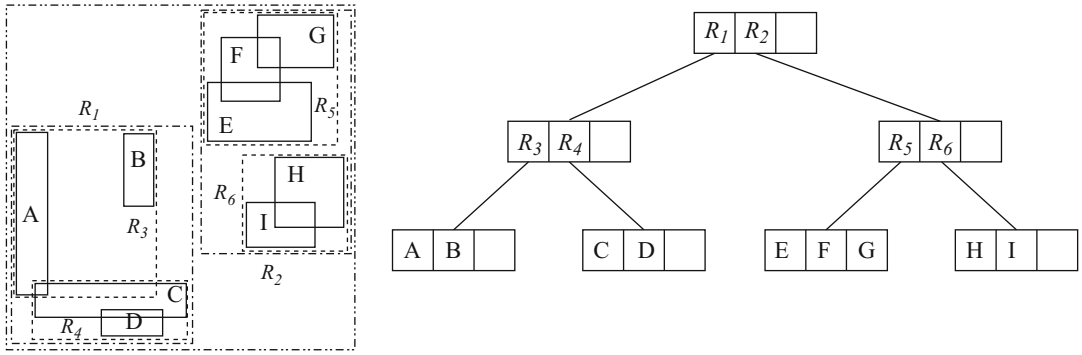
Theorem 1 ([1, 12]) *There is a set of N points in $\mathbb{R}^d$, such that for any R-tree $\mathcal{T}$ built on these points, there exists an empty window query for which the query algorithm has to visit $\Omega((N/B)^{1-1/d})$ nodes of $\mathcal{T}$.*

The *priority R-tree*, proposed by Arge et al. [3], matches the above lower bound.

Theorem 2 ([3]) *For any set S of N axis-parallel hypercubes in $\mathbb{R}^d$, the priority R-tree answers a window query with $O((N/B)^{1-1/d} + T/B)$ I/Os. It can be constructed with $O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$ I/Os.*

It is also reported that the priority R-tree performs well in practice, too [3]. However, it is not known how to update it efficiently while preserving the worst-case bound. The logarithmic method was used to support insertions and deletions [3], but the resulted structure is no longer an R-tree.

Note that the lower bound in Theorem 1 only holds for R-trees. If the data structure is not restricted to R-trees, better query bounds can be obtained for the window-query problem; see e.g., [4].



R-Trees, Fig. 1 An R-tree example in two dimensions

Applications

R-trees have been used widely in practice due to its simplicity and ability to store spatial objects of various shapes and to answer various queries. The areas of applications span from geographical information systems (GIS), computer-aided design, computer vision, and robotics. When the objects are not axis-parallel hypercubes, they are often approximated by their minimal bounding boxes, and the R-tree is then built on these bounding boxes. To answer a window query, first the R-tree is used to locate all the intersecting bounding boxes, followed by a filtering step that checks the objects exactly. The R-tree can also be used to support other kinds of queries, for example, aggregation queries, nearest neighbors, etc. In aggregation queries, each object o in S is associated with a weight $w(o) \in \mathbb{R}$, and the goal is to compute $\sum w(o)$ where the sum is taken over all objects that intersect the query range Q. The query algorithm is same as before, except that in addition it keeps running sum while traversing the R-tree and may skip an entire subtree rooted at some u if $R(u)$ is completely contained in Q. To find the nearest neighbor of a query point q, a priority queue is maintained, which stores all the nodes u that might contain an object that is closer to the current nearest neighbor found so far. The priority of u in the queue is the distance between q and $R(u)$. The search terminates when the current nearest neighbor is closer than the top element in the priority queue. However, no worst-case guarantees are known for R-trees answering these other types of queries, although they tend to perform well in practice.

Open Problems

Several interesting problems remain open with respect to R-trees. Some of them are listed here:

- Is it possible to design an R-tree with the optimal query bound $O((N/B)^{1-1/d} + T/B)$ that can also be efficiently updated? Or prove a lower bound on the update cost for such an R-tree.
- Is there an R-tree that supports aggregation queries for axis-parallel hypercubes in $O((N/B)^{1-1/d})$ I/Os? This would be optimal because the lower bound of Theorem 1 also holds for aggregation queries on R-trees. Note that, however, no sublinear worst-case bound exists for nearest-neighbor queries, since it is not difficult to design a worst-case example for which the distance between the query point q and any bounding box is smaller than the distance between q and its true nearest neighbor.
- When the window query Q shrinks to a point, that is, the query asks for all hypercubes in

S that contain the query point, the problem is often referred to as *stabbing queries* or *point enclosure queries*. The lower bound of Theorem 1 does not hold for this special case, while a lower bound of $\Omega(\log_2 N + T/B)$ was proven in [5], which holds in the strong *indexability* model. It is intriguing to find out the true complexity for stabbing queries using R-trees, which is between $\Omega(\log_2 N + T/B)$ and $O((N/B)^{1-1/d} + T/B)$.

Experimental Results

Nearly all studies on R-trees include experimental evaluations, mostly in two dimensions. Reportedly the Hilbert R-tree [10, 11] has been shown to have good query performance while being easy to construct. The R*-tree's insertion algorithm [6] has often been used for updating the R-tree. Please refer to the book by Manolopoulos et al. [14] for more discussions on the practical performance of R-trees.

Data Sets

Besides some synthetic data sets, the TIGER/Line data (http://www.census.gov/geo/www/tiger/) from the US Census Bureau has been frequently used as real-world data to test R-trees. The R-tree portal (http://www.rtreeportal.org/) also contains many interesting data sets.

URL to Code

Code for many R-tree variants is available at the R-tree portal (http://www.rtreeportal.org/). The code for the priority R-tree is available at (http://www.cs.duke.edu/~yike/prtree/).

Cross-References

- B-Trees
- External Sorting and Permuting
- I/O-Model

Recommended Reading

1. Agarwal PK, de Berg M, Gudmundsson J, Hammar M, Haverkort HJ (2002) Box-trees and R-trees with near-optimal query time. Discret Comput Geom 28:291–312
2. Aggarwal A, Vitter JS (1988) The Input/Output complexity of sorting and related problems. Commun ACM 31:1116–1127
3. Arge L, de Berg M, Haverkort HJ, Yi K (2004) The priority R-tree: a practically efficient and worst-case optimal R-tree. In: Proceedings of the SIGMOD international conference on management of data, pp 347–358
4. Arge L, Samoladas V, Vitter JS (1999) On two-dimensional indexability and optimal range search indexing. In: Proceedings of the ACM symposium on principles of database systems, pp 346–357
5. Arge L, Samoladas V, Yi K (2004) Optimal external memory planar point enclosure. In: Proceedings of the European symposium on algorithms
6. Beckmann N, Seeger B (2009) A revised R*-tree in comparison with related index structures. In: Proceedings of the SIGMOD international conference on management of data, pp 799–812
7. DeWitt DJ, Kabra N, Luo J, Patel JM, Yu J-B (1994) Client-server paradise. In: Proceedings of the international conference on very large databases, pp 558–569
8. García YJ, López MA, Leutenegger ST (1998) A greedy algorithm for bulk loading R-trees. In: Proceedings of the 6th ACM symposium on advances in GIS, pp 163–164
9. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proceedings of the SIGMOD international conference on management of data, pp 47–57
10. Haverkort H, van Walderveen F (2011) Four-dimensional Hilbert curves for R-trees. J Exp Algorithmics vol 16:Article no 3.4
11. Kamel I, Faloutsos C (1994) Hilbert R-tree: an improved R-tree using fractals. In: Proceedings of the international conference on very large databases, pp 500–509
12. Kanth KVR, Singh AK (1999) Optimal dynamic range searching in non-replicating index structures. In: Proceedings of the international conference on database theory. LNCS, vol 1540, pp 257–276
13. Leutenegger ST, Lopez MA, Edington J (1997) STR: a simple and efficient algorithm for R-tree packing. In: Proceedings of the 13th IEEE international conference on data engineering, pp 497–506
14. Manolopoulos Y, Nanopoulos A, Papadopoulos AN, Theodoridis Y (2005) R-trees: theory and applications. Springer, London
15. Sellis T, Roussopoulos N, Faloutsos C (1987) The R^+-tree: a dynamic index for multi-dimensional objects. In: Proceedings of the international conference on very large databases, pp 507–518

Rumor Blocking

Lidan Fan[1] and Weili Wu[2,3,4]
[1]Department of Computer Science, The University of Texas, Tyler, TX, USA
[2]College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi Province, China
[3]Department of Computer Science, California State University, Los Angeles, CA, USA
[4]Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Keywords

Influence diffusion models; Information cascades; Link analysis; Protector; Rumor; Submodular function; Social networks

Years and Authors of Summarized Original Work

2008; Kimura, Saito, Motoda
2011; Budak, Agrawal, Abbadi
2013; Fan, Lu, Wu, Thuraisingham, Ma, Bi

Problem Definition

The aim is to design effective algorithms for controlling rumor propagation in social networks. Here, a rumor is viewed as an undesirable thing. Social networks are represented by undirected or directed graphs, depending on different contexts. In these graphs, nodes denote individuals and edges denote the influence between individuals.

A list of strategies has been proposed to limit the spread of a rumor in a network. We group some of the existing research works into two categories. The first one includes the works that launch the opposite cascade, protector, to spread in a network, such that the number of nodes that adopt the rumor at the end of both cascades diffusion is limited. The second contains the works that are concerned with the network structure, that is, they control the rumor propagation by blocking some edges or nodes or both of them together in a network.

In this work, we mainly introduce two specific works belonging to the two different categories. For each of them, we briefly introduce some related works.

Problem 1 [2]
Two cascades, rumor (bad campaign) and protector (limiting campaign), diffuse simultaneously in a network. Influence diffusion models are used to capture their propagation processes. The objective is to limit the rumor propagation through protector diffusion.

Given a directed graph $G = (V, E)$, original rumor sources $R \subseteq V$, an integer $k > 0$, and the time delay d (a nonnegative integer) for detecting rumor sources, the objective is to identify k nodes as initial protectors, such that the expected number of nodes adopting the rumor at the end of both rumor and protector propagation processes is minimized, or equivalently, the reduction in the expected number of nodes adopting the rumor is maximized.

Two Influence Diffusion Models

Two influence diffusion models are adopted in [2].

Multi-campaign Independent Cascade Model (MCICM) In this model, a network is viewed as a directed graph $G = (V, E)$. The initial set of rumor sources is denoted by R, and the initial set of protectors is denoted by P. Each node must be in one of the three statuses: infected (by the rumor), protected (by the protector), and inactive (neither infected nor protected). Each edge e_{uv} is associated with two values $0 \leq p_r(u, v) \leq 1$ and $0 \leq p_p(u, v) \leq 1$. Once a node becomes infected or protected, it remains so forever.

The diffusion process unfolds in discrete time steps. In any step $t \geq 1$, when a node u first becomes infected (protected), it has a single chance to activate each currently inactive neighbor v, and it succeeds with probability $p_r(u, v)$ $(p_p(u, v))$ provided no neighbor of v tries activating v at the

same step. In other words, at step $t+1$, node v will become infected (protected) with probability $p_r(u,v)$ ($p_p(u,v)$) provided no neighbor of v tries activating v at the same step. If there are two or more nodes trying to activate v at the same step, at most one of them can succeed. If infected node(s) and protected node(s) try to activate a node at the same step, protected nodes have priority over infected nodes. The process continues until no newly infected or protected node appears.

Campaign-Oblivious Independent Cascade (COICM) This model is similar to the MCICM model; the only difference is that instead of two probabilities are associated with each edge, only one probability $0 \leq p(u,v) \leq 1$ is associated with each edge e_{uv}. That is, each node has the same probability to forward the two kinds of information, indicating both rumor and protector cascades pass through the same edge with the same probability.

Problem 2 [8]

A single cascade, rumor, diffuses through networks. Influence diffusion models are used to capture rumor propagation process. The objective is to limit the propagation of rumors through blocking links in networks. The aim of [8] is to minimize contamination degree by appropriately removing a fixed number of links. Here, the contamination degree of a network is used to measure how badly the rumor will contaminate the network; see its definition later.

Given a directed graph $G=(V,E)$, a positive integer k where $k<|E|$, find a subset $B^* \subset E$ with $|B^*|=k$ such that $c(G(B^*)) \leq c(G(B))$ for any $B \subset E$ with $|B|=k$. Here $c(G)$ denotes the contamination degree. For any link $e \in E$, let $G(e)$ denote the graph $G(V, E \setminus e)$. And $G(e)$ is used as the graph constructed by blocking e in G. Similarly, for any $B \subset E$, let $G(B)$ denote graph $G(V, E \setminus B)$. Then $G(B)$ represents the graph constructed by blocking B in G.

Independent Cascade Model

In this model, a network is considered as a directed graph $G=(V,E)$. Each edge $e_{uv} \in E$ is assigned an influence probability $p(u,v)$, representing the possibility that node u influences node v successfully. For $e_{uv} \notin E$, let $p(u,v)=0$. Each node can only be in one of the following two statuses: inactive or infected. Once a node becomes infected, it stays infected forever.

The diffusion process unfolds in discrete time steps. Starting with an initial set of infected nodes A_0, at any step $t \geq 1$, when node u first becomes infected in step t, it has a single chance to activate any of its currently inactive neighbors. For neighbor node v, it succeeds with probability $p(u,v)$. If u succeeds in activating v, then v will become infected in step $t+1$, and if u fails in activating v, then v will stay inactive. If node u does not succeed in activating v, it will not have a second chance to do in all subsequent steps. The process continues until no more activations are possible. If multiple newly activated nodes are in-neighbors of the same inactive node, then their activation attempts are sequenced in an arbitrary order.

Key Results

For Problem 1

Given the set of rumor sources R, a set of initial protectors P, and rumor detection delay d, a set function $f_{Rd}(P)$ represents the number of nodes that are prevented by P with diffusion delay d from adopting R. In other words, function $f_{Rd}(P)$ denotes the nodes that will be infected by R if, instead of P, the empty set is selected as the set of protectors. Therefore, the problem is to select P such that the expectation of $f_{Rd}(P)$ is maximized.

The NP-hardness of this problem is proved. Then for the MCICM model, the high-effectiveness property where $p_p(u,v)=1$ for edge $e_{uv} \in E$ is adopted. Then the objective functions are proved to be submodular and monotone under both the MCICM model with the high-effectiveness property and the COICM model. Therefore, Algorithm 1 is applied to provide $(1-1/e)$-approximation solutions for the problem. However, the objective function is not

Algorithm 1: The greedy algorithm

Input: Graph $G = (V, E)$, the set of initial rumor sources R, rumor detection delay d, a positive integer k, a positive number n, representing the simulation times
Output: Set P
$P = \emptyset$
for *i = 1 to k* **do**
  for *each* $u \in V \setminus (R \cup P)$ **do**
    $N_u = 0$
    for *j = 1 to n* **do**
      $N_u += f_{Rd}(P \cup \{u\}) - f_{Rd}(P)$
    end
    $N_u = N_u / n$
  end
  $Loc = \arg\max_{u \in (V \setminus (R \cup P))} \{N_u\}$
  $P = P \cup Loc$
end
Output P

submodular under the MCICM model without the high-effectiveness property.

Variants of Problem 1

Several works in different contexts also consider using the diffusion of protectors to contain the spread of rumors. In comparison to Problem 1, they have adopted different influence diffusion models, as well as formulated different optimization problems.

Selection of Fixed Number of Protectors

The work of He et al. [5] studies rumors blocking maximization under an extension of the classical Linear Threshold (LT) model [6], in which they incorporate two cascades, rumor and protector. Each node in this model can be in one of the three states: infected, protected, and inactive. For each node, its currently infected neighbors and protected neighbors determine whether it will become infected, protected, or stay inactive, respectively. When a node is activated by its infected neighbors and protected neighbors at the same time, then infected neighbors have priority over protected neighbors. Each edge $e = (u, v)$ has two weights, w^r_{uv} (rumor propagation) and w^p_{uv} (protector propagation). Each node u picks two independent thresholds from $[0, 1]$; one is for rumor diffusion and the other is for protector diffusion.

Then they develop the objective function $S_R(X)$ for this problem, which represents the expected number of nodes that is saved (from being infected by rumors R) by X. This problem is shown to be NP-hard and the objective function is proved to be submodular and monotone, then the greedy algorithm with performance ratio $1 - \frac{1}{e}$ is applied. To efficiently compute the values of $S_R(X)$, the authors propose the CLDAG algorithm.

Instead of choosing initial protectors from nodes not in rumor sources, the authors in [10] select a fixed number of nodes from initial infected nodes (rumor sources) and the rest of the nodes in a network as initial protectors, such that the number of nodes protected during T time steps is maximized. They study this problem under the LT model and the IC model. Two approximation algorithms are proposed.

Protection of a Subset of Nodes

Instead of limiting rumor diffusion through launching a fixed number of protectors, the work of [12] exploits the problem which aims to select the smallest set of influential people as protectors, such that the diffusion process starting from these protectors limits the propagation of rumors R in a fraction $0 \leq \alpha \leq 1$ of the whole network in T time steps. They study four variants of this problem, which are the combinations of the two parameters: R (can be unknown or known) and T (can be constrained or unconstrained). These problems are studied under the extensions of the IC model and the LT model, in which two cascades, rumors and protectors, are considered. For each edge, both of them have the same influence probability (IC) or influence weight (LT). For each node, they have the same threshold (LT). The key point is that when the two cascades try to activate a node at the same time, protectors have priority over rumors.

The authors prove the NP-hardness of the four problems under the proposed models. For the variant that R is unknown and T is unconstrained, the Greedy Viral Stopper (GVS) algorithm is adopted to select the protectors, and the solution obtained is within a constant factor (in terms of the number of nodes in the network) ex-

tra from the optimal solution. The GVS algorithm can be used in the variants that R is known and T is either constrained or unconstrained. These variants are shown to be hard to approximate to a logarithmic factor in terms of the number of nodes in the network. To get a good solution within short time, the Community-Based Heuristic algorithm is proposed.

Noticing the community structure of social networks, the work of Fan et al. [3] contains rumor propagation by selecting a minimal set of initial protectors to protect a special kind of vertex set, which play the role as the "gates" of rumors' neighborhood communities. Two variants of the problem are studied under two different models, both variants are shown to be NP-hard, and approximation algorithms are developed to obtain good solutions.

Game Theory Aspect

The rumor blocking is also studied from the game theory aspect [13], where it uses graphs with nodes representing the tribal leaders and edges representing possible transmission of influence. Under this context, rumor blocking is viewed as a two-player game, in which one player, the rumor, will attempt to maximize the number of nodes accepting it while the second player, the protector, will attempt to minimize the rumor's influence. Both the rumor and the protector will choose their action sources (initial rumor sources and initial protector sources). In the zero-sum game context, the rumor's payoff is equal to the expected number of nodes infected, and the protector's payoff is the opposite of the rumor's payoff. The authors propose a double oracle algorithm for this game.

For Problem 2

Under the IC model, given an initial active set X, define the number of active nodes at the end of the influence diffusion process on G as $f(X;G)$. Let $\sigma(X;G)$ denote the expected value of $f(X;G)$. $\sigma(X;G)$ is called as the influence degree of node set X on graph G. Two notions of containment degrees are defined. One is called *Average Contamination Degree*, representing the average of influence degree of all the nodes in G, denoted as $c_0(G)$. Its definition is $c_0(G) = \frac{1}{|V|}\sum_{v\in V}\sigma(v;G)$. The other is called *Worst Contamination Degree*, representing the maximum of influence degree of all the nodes in G, denoted as $c_+(G)$. Its definition is: $c_+(G) = \max_{v\in V}\sigma(v;G)$. Approximation algorithms are proposed to find good solution for the problem.

Algorithm 2: The greedy algorithm - IC

Input: Graph $G_0 = (V_0, E_0)$, a positive integer $k < |E_0|$
Output: The set of links blocked
Initialize a subset $L \subseteq E_0$ as $L =$ Initialize a graph $G = (V, E)$ as $V = V_0$, $E = E_0$
while $|L| < k$ **do**
  select a link e^*, such that
  $e^* = \arg\min_{e\in E} c(G(e))$
  $L = L \cup \{e^*\}$
  $E = E \setminus \{e^*\}$
end
Output L

For a given graph $G = (V, E)$, exactly computing influence degree $c(G(e)); e \in E$ in Algorithm 2 is an open problem. Therefore, heuristic strategies are proposed to estimate $c(G(e)); e \in E$. These estimations are based on the *Bond Percolation Method* proposed by Kimura in [7], which we describe below.

Bond Percolation Method [7]

Assume there are propagation probabilities $\{p_e; e \in E\}$ on a graph $G = (V, E)$. In terms of information diffusion on a network, the *occupied* links represent the links that the information propagates, and the *unoccupied* links represent the links that the information does not propagates. The bond percolation process with occupation probabilities $\{p_e; e \in E\}$ on a graph $G = (V, E)$ is a stochastic process in which the probability of each link $e \in E$ becomes occupied is p_e.

Construct N graphs through the bond percolation process, that is, $\{G_n = (V, E_n); n = 1, \ldots, N\}$. For any $u \in V'$ on graph $G' = (V', E')$, let $F(u; G')$ represent the set of all the nodes that are *reachable* from u on G'. A node v is said to be reachable from u if there is a path from u to v through the links on G'. Define function $g(u; G, N) =$

$\frac{1}{N}\sum_{n=1}^{N}|F(u;G^n)|$. Then, $g(u;G,N)$ can be used to estimate $\sigma(u;G)$, where $u \in V$ and if N is sufficiently large. Decompose each G_n into the *strongly connected components (SCC)* as $V = \bigcup_{i=1}^{I_n} \mathrm{SCC}(u_i^n;G_n)$, where $u_i^n \in V$ and $\mathrm{SCC}(u_i^n;G_n)$ denotes the SCC of graph G_n that contains u_i^n. I_n is the number of the SCC of graph G_n, using the fact that $|F(u;G^n)| = |F(u_i^n;G_n)|$ for all $u \in V$ to calculate $\{|F(u;G^n)|;u \in V,n = 1,\ldots,N\}$, then compute $g(u;G,N)$, and finally $\sigma(u;G)$ can be calculated, where $u \in V$.

Estimation Method

We now describe how to estimate $c(G(e));e \in E$ in [8]. For a graph $G = (V,E)$, first, construct N sample graphs through the bond percolation process as $\{G_n = (V,E_n);n = 1,\ldots,N\}$. Next, for each $e \in E$, identify the subset of N, which is denoted as $S_N(e)$ and satisfies $S_N(e) = \{n \in \{1,\ldots,N\};e \notin E_n\}$. Now apply the bond percolation process on the graph $G(e) = (V,E \setminus \{e\})$ for $|S_N(e)|$ times, then $|S_N(e)|$ graphs are obtained by the occupied links, denote them as $\{G(e)^n;n = 1,\ldots,|S_N(e)|\}$. Given that N is large enough to ensure $|S_N(e)|$ sufficiently large, then the function $g(u;G(e),|S_N(e)|)$ equals to $\frac{1}{|S_N(e)|}\sum_{n=1}^{|S_N(e)|}|F(u;G(e)^n)|$, where $u \in V$, can be used to estimate $\sigma(u;G(e))$. Since each link of the graph G is independently declared occupied in the bond percolation process, then an alternative, $g'(u,e) = \frac{1}{|S_N(e)|}\sum_{n\in S_N(e)}|F(u;G^n)|$, is used to estimate $\sigma(u;G(e))$.

Variants of Problem 2

The authors in [9] adapt the method they used for the IC model to study Problem 2 under the LT model [6]. In [1], the authors incorporate the trust among users in the information propagation process, and they propose a measure to compute trust between a pair of users. Then a Weighted Trust Network (WTN) is built, and the objective of the problem is to find the Maximum Spanning Tree (MST) in the WTN and, finally, immunize all the edges in the MST of the WTN. Another method that controls rumor spread through blocking nodes and links simultaneously can be found in [4].

Nguyen et al. [11] study the rumor blocking problem under a dynamic social network structure. They propose to distribute patches to the most influential nodes in the social network, such that the number of nodes influenced by rumors is limited. In their work, they first take into account the network community structure and adaptively keeps it updated as the social network evolves, and then select most influential individuals from each communities to be patched.

The work of Zhu [14] focuses on the rumor blocking problem in cellular networks. First, a social relationship graph between mobile phones is obtained based on network traffic; the authors develop two graph-partitioning algorithms to partition the graph into many separate parts as possible and contain the rumor diffusion within each part. Then a minimum set of *key nodes*, which separate these different parts, is selected to be patched. The intuition is that the infected nodes in a part need to go through some of these key nodes to influence nodes in another part. Once these nodes are patched, it is impossible for the influence propagates among different parts.

Applications

Practical applications can be seen in controlling: propagation of computer viruses and worms propagates over computer networks, spread of malicious rumors through social networks, diffusion of infections or epidemics (such as swine flu) among groups of people, propagation of mobile worm in cellular networks, and so on.

Open Problems

There are many interesting directions that deserve further explorations. One direction is to improve existing influence diffusion models by considering continuous time influence diffusion, users' preferences to different kinds of information,

factors influencing users' threshold in adopting a kind of information, etc. Another direction is to design efficient strategies to control the spread of rumors when only partial of a network structure is observable. Another research issue is incorporating the detection of rumor sources into rumor blocking and continuous time delay of protectors.

Cross-References

- ▸ Influence and Profit
- ▸ Influence Maximization

Recommended Reading

1. Bao Y, Niu Y, Yi C, Xue Y (2014) Effective immunization strategy for rumor propagation based on maximum spanning tree. In: Computing, Networking and Communications (ICNC), 2014 International Conference on, pp 11–15, DOI:10.1109/ICCNC.2014.6785296
2. Budak C, Agrawal D, El Abbadi A (2011) Limiting the spread of misinformation in social networks. In: Srinivasan S, Ramamritham K, Kumar A, Ravindra MP, Bertino E, Kumar R (eds) WWW, ACM, pp 665–674
3. Fan L, Lu Z, Wu W, Thuraisingham BM, Ma H, Bi Y (2013) Least cost rumor blocking in social networks. In: ICDCS, IEEE, pp 540–549
4. He J, Liang H, Yuan H (2011) Controlling infection by blocking nodes and links simultaneously. In: Chen N, Elkind E, Koutsoupias E (eds) WINE, Springer, Lecture Notes in Computer Science, vol 7090, pp 206–217
5. He X, Song G, Chen W, Jiang Q (2012) Inuence blocking maximization in social networks under the competitive linear threshold model. In: SDM, SIAM / Omnipress, pp 463–474
6. Kempe D, Kleinberg JM, Tardos E (2003) Maximizing the spread of inuence through a social network. In: Getoor L, Senator TE, Domingos P, Faloutsos C (eds) KDD, ACM, pp 137–146
7. Kimura M, Saito K, Nakano R (2007) Extracting inuential nodes for information diusion on a social network. In: AAAI, AAAI Press, pp 1371–1376
8. Kimura M, Saito K, Motoda H (2008) Minimizing the spread of contamination by blocking links in a network. In: Fox D, Gomes CP (eds) AAAI, AAAI Press, pp 1175–1180
9. Kimura M, Saito K, Motoda H (2008) Solving the contamination minimization problem on networks for the linear threshold model. In: Ho TB, Zhou ZH (eds) PRICAI, Springer, Lecture Notes in Computer Science, vol 5351, pp 977–984
10. Li S, Zhu Y, Li D, Kim D, Huang H (2013) Rumor restriction in online social networks. In: IPCCC, IEEE, pp 1–10
11. Nguyen N, Xuan Y, Thai M (2010) A novel method for worm containment on dynamic social networks. In: MILITARY COMMUNICATIONS CONFERENCE, 2010 - MILCOM 2010, pp 2180–2185, DOI:10.1109/MILCOM.2010.5680488
12. Nguyen NP, Yan G, Thai MT, Eidenbenz S (2012) Containment of misinformation spread in online social networks. In: Contractor NS, Uzzi B, Macy MW, Nejdl W (eds) WebSci, ACM, pp 213–222
13. Tsai J, Nguyen TH, Tambe M (2012) Security games for controlling contagion. In: Homann J, Selman B (eds) AAAI, AAAI Press
14. Zhu Z, Cao G, Zhu S, Ranjan S, Nucci A (2009) A social network based patching scheme for worm containment in cellular networks. In: INFOCOM, IEEE, pp 1476-1484

S

Schedulers for Optimistic Rate Based Flow Control

Panagiota Fatourou
Department of Computer Science, University of Ioannina, Ioannina, Greece

Keywords

Bandwidth allocation; Rate adjustment; Rate allocation

Years and Authors of Summarized Original Work

2005; Fatourou, Mavronicolas, Spirakis

Problem Definition

The problem concerns the design of efficient rate-based flow control algorithms for virtual-circuit communication networks where a connection is established by allocating a fixed path, called *session*, between the source and the destination. Rate-based flow-control algorithms repeatedly adjust the transmission rates of different sessions in an end-to-end manner with primary objectives to optimize the network utilization and achieve some kind of fairness in sharing bandwidth between different sessions.

A widely-accepted fairness criterion for flow-control is *max-min fairness* which requires that the rate of a session can be increased only if this increase does not cause a decrease to any other session with smaller or equal rate. Once max-min fairness has been achieved, no session rate can be increased any further without violating the above condition or exceeding the bandwidth *capacity* of some link. Call *max-min* rates the session rates when max-min fairness has been reached.

Rate-based flow control algorithms perform rate adjustments through a sequence of *operations* in a way that the capacities of network links are never exceeded. Some of these algorithms, called *conservative* [3, 6, 10, 11, 12], employ operations that gradually increase session rates until they converge to the max-min rates without ever performing any rate decreases. On the other hand, *optimistic* algorithms, introduced more recently by Afek, Mansour, and Ostfeld [1], allow for decreases, so that a session's rate may be intermediately be larger than its final max-min rate.

Optimistic algorithms [1, 7] employ a specific rate adjustment operation, called *update operation* (introduced in [1]). The goal of an `update` operation is to achieve fairness among a set of neighboring sessions and optimize the network utilization in a local basis. More specifically, an `update` operation calculates an increase for the rate of a particular session (the *updated* session) for each link the session traverses. The calculated increase on a particular link is the maximum possible that respects the max-min fairness condition between the sessions traversing the link; that

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

is, this increase should not cause a decrease to the rate of any other session traversing the link with smaller rate than the rate of the updated session after the increase. Once the maximum increase on each link has been calculated the minimum among them is applied to the session's rate (let e be the link for which the minimum increase has been calculated). This causes the decrease of the rates of those sessions traversing e which had larger rates than the increased rate of the updated session to the new rate. Moreover, the `update` operation guarantees that all the capacity of link e is allocated to the sessions traversing it (so the bandwidth of this link is fully utilized).

One important performance parameter of a rate-based flow control algorithm is its *locality* which is characterized by the amount of knowledge the algorithm requires to decide which session's rate to update next. *Oblivious* algorithms do not assume any knowledge of the network topology or the current session rates. *Partially oblivious* algorithms have access to session rates but they are unaware of the network topology, while *non-oblivious* algorithms require full knowledge of both the network topology and the session rates. Another crucial performance parameter of rate-based flow control algorithms is the *convergence complexity* measured as the maximum number of rate-adjustment operations performed in any execution until max-min fairness is achieved.

Key Results

Fatourou, Mavronicolas and Spirakis [7] have studied the convergence complexity of optimistic rate-based flow control algorithms under varying degrees of locality. More specifically, they have proved lower and upper bounds on the convergence complexity of oblivious, partially-oblivious and non-oblivious algorithms. These bounds are expressed in terms of n the number of sessions laid out on the network.

Theorem 1 (Lower Bound for Oblivious Algorithms, Fatourou, Mavronicolas and Spirakis [7]) *Any optimistic, oblivious, rate-based flow control algorithm requires $\Omega(n^2)$ update operations to compute the max-min rates.*

Fatourou, Mavronicolas and Spirakis [7] have presented algorithm `RoundRobin`, which applies `update` operations to sessions in a round robin order. Obviously, `RoundRobin` is oblivious. It has been proved [7] that the convergence complexity of `RoundRobin` is $O(n^2)$. This shows that the lower bound for oblivious algorithms is tight.

Theorem 2 (Upper Bound for Oblivious Algorithms, Fatourou, Mavronicolas and Spirakis [7]) `RoundRobin` *computes the max-min rates after performing $O(n^2)$ update operations.*

`RoundRobin` belongs to a class of oblivious algorithms, called `Epoch` [7]. Each algorithm of this class repeatedly chooses some permutation of all session indices and applies `update` operations on the sessions in the order determined by this permutation. This is performed n times. Clearly, `Epoch` is a class of oblivious algorithms. It has been proved [7] that each of the algorithms in this class has convergence complexity $O(n^2)$.

Another oblivious algorithm, called `Arbitrary`, has been presented in [1]. The algorithm works in a very simple way by choosing the next session to be updated in an arbitrary way, but it requires an exponential number of `update` operations to compute the max-min rates.

Fatourou, Mavronicolas and Spirakis [7] have proved that partially-oblivious algorithms do not achieve better convergence complexity than oblivious algorithms despite the knowledge they employ.

Theorem 3 (Lower Bound for Partially Oblivious Algorithms, Fatourou, Mavronicolas and Spirakis [7]) *Any optimistic, partially oblivious, rate-based flow control algorithm*

requires $\Omega(n^2)$ update operations to compute the max-min rates.

Afek, Mansour and Ostfeld [1] have presented a partially oblivious algorithm, called `GlobalMin`. The algorithm chooses as the session to update next the one with the minimum rate among all sessions. The convergence complexity of `GlobalMin` is $O(n^2)$ [1]. This shows that the lower bound for partially-oblivious algorithms is tight.

Theorem 4 (Upper Bound for Partially Oblivious algorithms, Afek, Mansour and Ostfeld [1]) `GlobalMin` *computes the max-min rates after performing $O(n^2)$ update operations.*

Another partially-oblivious algorithm, called `LocalMin`, is also presented in [1]. The algorithm chooses to schedule next a session which has a minimum rate among all the sessions that share a link with it. `LocalMin` has time complexity $O(n^2)$.

Fatourou, Mavronicolas and Spirakis [7] have presented a non-oblivious algorithm, called `Linear`, that exhibits linear convergence complexity. `Linear` follows the classical idea [3, 12] of selecting as the next updated session one of the sessions that traverse the most congested link in the network. To discover such a session, `Linear` requires knowledge of the network topology and the session rates.

Theorem 5 (Upper Bound for Non-Oblivious Algorithms, Fatourou, Mavronicolas and Spirakis [7]) *Linear computes the max-min rates after performing O(n) update operations.*

The convergence complexity of `Linear` is optimal, since n rate adjustments must be performed in any execution of an optimistic rate-based flow control algorithm (assuming that the initial session rates are zero). However, this comes at a remarkable cost in locality which makes `Linear` impractical.

Applications

Flow control is the dominant technique used in most communication networks for preventing data traffic congestion when the externally injected transmission load is larger than what can be handled even with optimal routing. Flow control is also used to ensure high network utilization and fairness among the different connections. Examples of networking technologies where flow control techniques have been extensively employed to achieve these goals are TCP streams [5] and ATM networks [4]. An overview of flow control in practice is provided in [3].

The idea of controlling the rate of a traffic source originates back to the data networking protocols of the ANSI Frame Relay Standard. Rate-based flow control is considered attractive due to its simplicity and its low hardware requirements. It has been chosen by the ATM Forum on Traffic Management as the best suited technique for the goals of ABR service [4].

A substantial amount of research work has been devoted in past to conservative flow control algorithms [3, 6, 10, 11, 12]. The optimistic framework has been introduced much later by Afek et al. [1] as a more suitable approach for real dynamic networks where decreases of session rates may be necessary (e.g., for accommodating the arrival of new sessions). The algorithms presented in [7] improve upon the original algorithms proposed in [1] in terms of either convergence complexity, or locality, or both. Moreover, they identify that certain classical scheduling techniques, such as round-robin [11], or adjusting the rates of sessions traversing one of the most congested links [3, 12] can be efficient under the optimistic framework. The first general lower bounds on the convergence complexity of rate-based flow control algorithms are also presented in [7].

The performance of optimistic algorithms has been theoretically analyzed in terms of an abstraction, namely the `update` operation, which has been designed to address most of the intricacies encountered by rate-based flow control algorithms. However, the `update` operation

masks low-level implementation details, while it may incur non-trivial, local computations on the switches of the network. Fatourou, Mavronicolas and Spirakis [9] have studied the impact on the efficiency of optimistic algorithms of local computations required at network switches in order to implement the `update` operation, and proposed a distributed scheme that implements a broad class of such algorithms. On a different avenue, Afek, Mansour and Ostfeld [2] have proposed a simple flow control scheme, called `Phantom`, which employs the idea of considering an imaginary session on each link [10, 12], and they have discussed how `Phantom` can be applied to ATM networks and networks of TCP routers.

A broad class of modern distributed applications (e.g., remote video, multimedia conferencing, data visualization, virtual reality, etc.) exhibit highly differing bandwidth requirements and need some kind of quality of service guarantees. To efficiently support a wide diversity of applications sharing available bandwidth, a lot of research work has been devoted on incorporating priority schemes on current networking technologies. Priorities offer a basis for modeling the diverse resource requirements of modern distributed applications, and they have been used to accommodate the needs of network management policies, traffic levels, or pricing. The first efforts for embedding priority issues into max-min fair, rate-based flow control were performed in [10, 12]. An extension of the classical theory of max-min fair, rate-based flow control to accommodate priorities of different sessions has been presented in [8]. (A number of other papers addressing similar generalizations of max-min fairness to account for priorities and utility have been presented after the original publication of [8].)

Many modern applications are not based solely on point-to-point communication but they rather require multipoint-to-multipoint transmissions. A max-min fair rate-based flow control algorithm for multicast networks is presented in [14]. Max-min fair allocation of bandwidth in wireless adhoc networks is studied in [15].

Open Problems

The research work on optimistic, rate-based flow control algorithms leaves open several interesting questions. The convergence complexity of the proposed optimistic algorithms has been analyzed only for a static set of sessions laid out on the network. It would be interesting to evaluate these algorithms under a dynamic network setting, and possibly extend the techniques they employ to efficiently accommodate arriving and departing sessions.

Although max-min fairness has emerged as the most frequently praised fairness criterion for flow control algorithms, achieving it might be expensive in highly dynamic situations. Afek et al. [1] have proposed a modified version of the `update` operation, called *approximate* `update`, which applies an increase to some session only if it is larger than some quantity $\delta > 0$. An *approximate* optimistic algorithm uses the approximate `update` operation and terminates if no session rate can be increased by more than δ. Obviously such an algorithm does not necessarily reach max-min fairness. It has been proved [1] that for some network topologies every approximate optimistic algorithm may converge to session rates that are away from their max-min counterparts by an exponential factor. The consideration of other versions of `update` operation or different termination conditions might lead to better max-min fairness approximations and deserves more study; different choices may also significantly impact the convergence complexity of approximate optimistic algorithms. It would be also interesting to derive trade-off results between the convergence complexity of such algorithms and the distance of the terminating rates they achieve to the max-min rates.

Fairness formulations that naturally approximate the max-min condition have been proposed by Kleinberg et al. [13] as suitable fairness criteria for certain routing and load balancing applications. Studying these formulations under the rate-based flow control setting is an interesting open problem.

Cross-References

▸ Multicommodity Flow, Well-Linked Terminals and Routing Problems

Recommended Reading

1. Afek Y, Mansour Y, Ostfeld Z (1999) Convergence complexity of optimistic rate based flow control algorithms. J Algorithms 30(1):106–143
2. Afek Y, Mansour Y, Ostfeld Z (2000) Phantom: a simple and effective flow control scheme. Comput Netw 32(3):277–305
3. Bertsekas DP, Gallager RG (1992) Data networks, 2nd edn. Prentice Hall, Englewood Cliffs
4. Bonomi F, Fendick K (1995) The rate-based flow control for available bit rate ATM service. IEEE/ACM Trans Netw 9(2):25–39
5. Brakmo LS, Peterson L (1995) TCP vegas: end-to-end congestion avoidance on a global internet. IEEE J Sel Areas Commun 13(8):1465–1480
6. Charny A (1994) An algorithm for rate-allocation in a packet switching network with feedback. Technical report MIT/LCS/TR-601, Massachusetts Institute of Technology, Apr 1994
7. Fatourou P, Mavronicolas M, Spirakis P (2005) Efficiency of oblivious versus non-oblivious schedulers for optimistic, rate-based flow control. SIAM J Comput 34(5):1216–1252
8. Fatourou P, Mavronicolas M, Spirakis P (2005) Max-min fair flow control sensitive to priorities. J Interconnect Netw 6(2):85–114, Also in Proceedings of the 2nd international conference on principles of distributed computing, pp 45–59 (1998)
9. Fatourou P, Mavronicolas M, Spirakis P (1998) The global efficiency of distributed, rate-based flow control algorithms. In: Proceedings of the 5th colloquium on structural information and communication complexity, June 1998, pp 244–258
10. Gafni E, Bertsekas D (1984) Dynamic control of session input rates in communication networks. IEEE Trans Autom Control 29(11):1009–1016
11. Hahne E (1991) Round Robin scheduling for max-min fairness in data networks. IEEE J Sel Areas Commun 9(7):1024–1039
12. Jaffe J (1981) Bottleneck flow control. IEEE Trans Commun 29(7):954–962
13. Kleinberg J, Rabani Y, Tardos É (1999) Fairness in routing and load balancing. In: Proceedings of the 40th annual IEEE symposium on foundations of computer science, Oct 1999, pp 568–578
14. Sarkar S, Tassiulas L (2005) Fair distributed congestion control in multirate multicast networks. IEEE/ACM Trans Netw 13(1):121–133
15. Tassiulas L, Sarkar S (2005) Maxmin fair scheduling in wireless adhoc networks. IEEE J Sel Areas Commun 23(1):163–173

Scheduling in Data Broadcasting

Xiaofeng Gao
Department of Computer Science, Shanghai Jiao Tong University, Shanghai, China

Keywords

Algebraic algorithm; Data retrieval; Scheduling; Wireless data broadcast

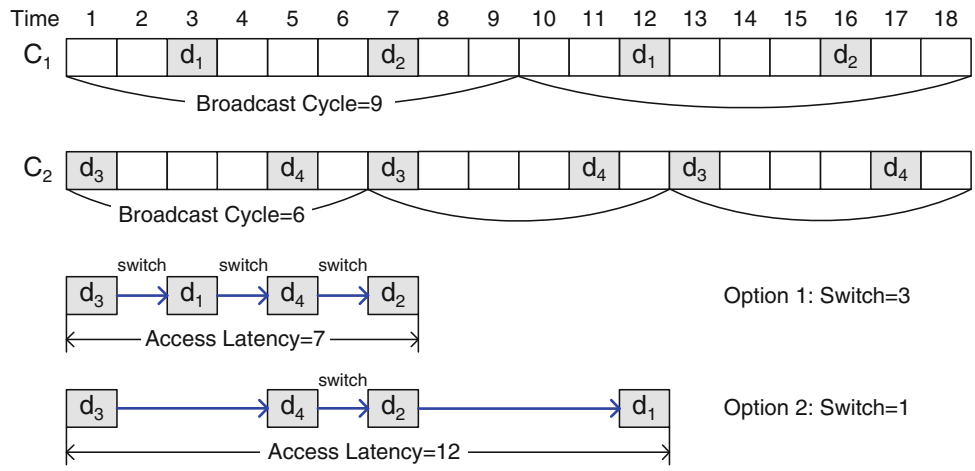
Years and Authors of Summarized Original Work

2011; Gao, Lu, Wu, Fu
2013; Gao, Lu, Wu, Fu

Problem Definition

Wireless data broadcasting means a set of data are repeatedly broadcast from a base station to a mass number of wireless and mobile clients. If a client wants a specific datum, it will access onto the broadcasting channel, get the location (appearance time) of the datum with the help indices, and wait until the datum has been broadcast. The scheduling problem in data broadcasting deals with the design of an efficient permutation strategy for a client to download a required subset of data from an multichannel broadcasting system, with both time and energy constraints. Here time constraint means the client wants the minimum downloading time from when it starts the query until the moment it has successfully download each piece of datum, while the energy constraint means the client wants the minimum switching numbers among channels to reduce extra battery consumption. Correspondingly, we can define the scheduling problem formally as follows:

A client wants to download a group of k data items $D = \{d_1, d_2, \ldots, d_k\}$, each with different sizes. Those data items are broadcasted on n different channels $C = \{c_1, c_2, \ldots, c_n\}$ repeatedly together with many other data items. Each channel may have different bandwidth and



Scheduling in Data Broadcasting, Fig. 1 Example of possible objective contradiction

broadcast cycle length. Let the time to download the smallest transmission packet be a unit time, and the length of d_i can be represented as l_i (also referring as downloading time).

Assume the client knows the locations (channel id and time offsets) of the required data set beforehand at the starting time $t = 0$ (with the help of indices, which is beyond the scope of our problem), and the target is how to download k known data from n channels efficiently with minimum downloading time (we also refer it as *access latency*) and minimum switching numbers.

Unfortunately, the two objectives in this problem are conflicting to each other. Figure 1 is an example to illustrate this phenomenon. In Fig. 1, there are two channels broadcasting 15 data items repeatedly. Suppose the gray data items {1,2,3,4} are of the request. The starting point of the client retrieving process is at $t = 0$. If we want to minimize the access latency, the request should be retrieved in the order of "$3 \rightarrow 1 \rightarrow 4 \rightarrow 2$" which takes only 7 time units but needs 3 switches (shown as Option 1 in Fig. 1). However, if we want to minimize the switches, the best retrieving order should be "$3 \rightarrow 4 \rightarrow 2 \rightarrow 1$" which needs only 1 switch but takes 12 time units (shown as Option 2 in Fig. 1). This example exhibits that access latency and number of switches cannot be minimized at the same time. They are contradictory factors.

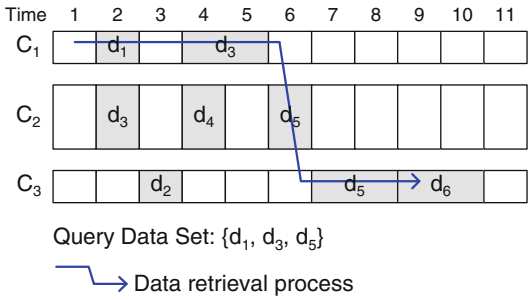
As a consequence, we want to fix one factor and minimize another objective, and thus have the following objective:

Objective

We hope to design a data downloading order for a client to download k data items from n broadcasting channels, such that the access latency t is minimized if we restrict the switch number among channels (denoted as h); otherwise, we will minimize the number of switches h once the access latency t is bounded.

Constraints

1. **Switch Constraint**: Note that if a client is downloading a data from channel c_i at time t_0, then it cannot switch to channel c_j, where $j \neq i$, to download another data at time $t_0 + 1$ due to connection protocols. Thus, we assume if a client wants to download data from another channel, it needs at least one time unit for channel switching. Figure 2 gives a typical process of data retrieval in multichannel broadcast environments. The query data set is $\{d_1, d_3, d_5\}$, and a user can download data object d_1 and d_3 from channel c_1 and then switch to channel c_3 at time $t = 6$ to download data object d_5 at time $t = 7$. However, after time $t = 5$, the user cannot switch from channel c_1 to c_2 to download data



Scheduling in Data Broadcasting, Fig. 2 Switch constraint

d_5 at time $t = 6$. From Fig. 2, we also get that the bandwidths of different channels are not necessarily the same. Actually, the bandwidth of channel c_2 is twice as that of c_1 or c_3, thus d_3 or d_5, which take two time slots on c_1 or c_3, can be broadcasted in one time unit by c_2.

2. **Objective Constraint**: We have to setup a reasonable threshold for latency constraint t and switch constraint h, such that we would achieve a feasible solution for the corresponding minimized switches and shortest access latency.

Problem 1 (Scheduling in Data Broadcasting)

INPUT: *The required data subset* $D = \{d_1, d_2, \ldots, d_k\}$ *broadcast on* n *different channels with their locations and downloading time* l_i, *a switch constraint* h *or latency constraint* t.

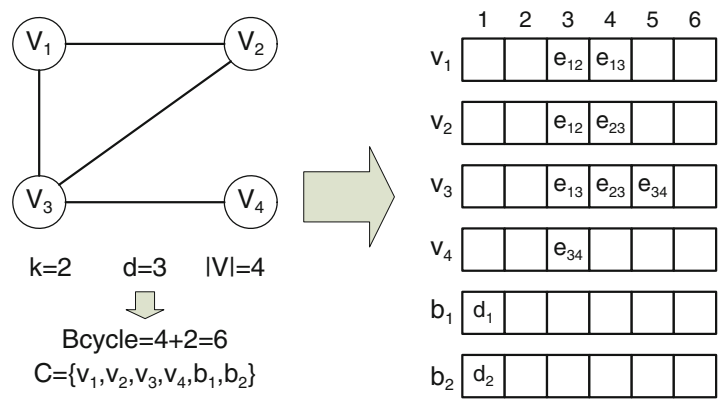
OUTPUT: *A permutation of* D *such that if starting from time slot zero, a client would achieve the shortest access latency (with switch threshold* h*) or the minimum switch numbers (with latency threshold* t*) if it follows this permutation to download each data item sequentially with switch constraint.*

Key Results

Scheduling is an important part in the wireless data broadcast system. Researchers tend to divide the scheduling problems into two subproblems. The first one is the data allocation problem in the server side, while the other one is the data retrieval problem in the client size.

With respect to server side scheduling, several works have been proposed to improve the system performance [1–5]. Acharya et al. [1] first dealt with the data allocation problem for single-channel environment. He proposed a scheduling algorithm considering data access frequencies and allowed frequent accessed data to be broadcasted more often. Most works concerned multi-channel environment. For data set with uniform length, Yee et al. [2] proposed an $O(t^2m)$ time-complexity dynamic programming algorithm to find the optimal schedule and also a near optimal greedy algorithm to reduce the time complexity. For nonuniform lengths case, Ardizzoni et al. [3] proved that this problem is strong NP-hard. Ardizzoni et al. [3], Anticaglia et al. [4], and Kenyon et al. [5] designed algorithms based on greedy and heuristic strategy.

Also most of the literature discussed the data allocation problem from server's point of view; several works [6–10] considered the data retrieval scheduling problem from the client's point of view. Shi et al. [6] defined the data retrieval problem in MIMO environment as parallel data retrieval scheduling with MIMO Antennae (PADRS-MIMO) and proposed two greedy heuristics to guarantee minimum switchings among channels or reduce the downloading time when the number of antennae in the mobile devices are limited. Lu et al. [7, 8] defined the largest number data retrieval (LNDR) and maximum cost data retrieval (MCDR) problems and considered the hopping cost. He also proved that when the hopping cost cannot be ignored, LNDR is NP-hard and designed a 1/2-approximation algorithm. Gao et al. [9, 10] designed a randomized algebraic algorithm that takes both energy cost and access time into consideration to schedule the data retrieval process in multichannel environments. The algorithm proposed can detect whether a given data retrieval problem has a solution with access time t and number of switchings h in $O\left(2^k(hnt)^{O(1)}\right)$ time, where n is the number of channels and k is the number of requested data items.



Scheduling in Data Broadcasting, Fig. 3 Example of VC reduction

Hardness Analysis

Define a tuple $s = \{i_s, j_s, t_s, t'_s\}$ to denote the datum d_{i_s}, which can be downloaded from channel c_{j_s} during the time span $[t_s, t'_s]$; then it is clear that a valid data retrieval schedule is a sequence of k intervals $s_1, s_2, \ldots, s_k$, each tuple corresponds to a distinct data item in D, and there are no conflicts between any two of the k tuples. To analyze the NP-hardness, we then define the decision problem of MCDR.

Definition 1 (Decision MCDR) Given a data set D, a channel set C, a time threshold t, and a switching threshold h, find a valid data retrieval schedule to download all the data in D from C before time t with at most h switchings.

Theorem 1 *MCDR problem is NP-hard.*

Proof We use $VC \leq_p MCDR$ to prove this theorem. Here VC is the decision problem of vertex cover, say, given a graph $G = (V, E)$, we want to find a minimum size vertex subset $VC \subseteq V$ such that for any edge $(v_i, v_j) \in E$, either $v_i \in VC$ or $v_j \in VC$. An instance of vector cover is: given a graph $G = (V, E)$ and integer k, does it have a vertex cover VC with size k? Then we construct an instance of MCDR from G and k as follows:

1. For each vertex $v_i \in V$, define a channel v_i. Define another k channels $b_1, \ldots, b_k$. Then the channel set is $C = \{v_1, \ldots, v_{|V|}, b_1, \ldots, b_k\}$. Totally $|V| + k$ channels. Let δ be the maximum vertex degree in G, and then each channel has broadcast cycle length $\delta + 3$.
2. For each edge $(v_i, v_j) \in E$, define a unit length data item e_{ij} in data set D_e and append it on channel c_i and c_j (the order can be arbitrary and starting from the third time unit).
3. For each channel b_i, define a unit length data item d_i in data set D_d and allocate it on the first time unit of channel b_i.
4. The data set $D = D_e \cup D_b$.

Figure 3 is an example to show how to construct the broadcast system. In this figure, $\delta = 3$, $k = 2$, $|V| = 4$; thus, the channel set should be $\{v_1, v_2, v_3, v_4, b_1, b_2\}$, each having broadcast length $\delta + 3 = 6$. Each e_{ij} represents an edge (v_i, v_j), and it is clear that if we download all data items from channel v_i, then it means we cover the edges connecting node v_i.

Next, we prove that G has a vertex cover with size k if and only if there is a valid data retrieval schedule S such that $t = k(\delta+3)$ and $h = 2k-1$.

$\Longrightarrow$: If G has a vertex cover VC with size k, then we can select the corresponding k channels in $\{v_i | v_i \in VC\}$ to receive all the data in k cycles. At the beginning of ith cycle (iteration), the client will visit b_i at $t = 1$, and hop to some $v_i \in VC$ channel, stay on this channel till the last time unit of the broadcast cycle, and then hop to b_{i+1}. There are k b_is, so each iteration client will download one of them. VC is a vertex cover, so following all

$v_i \in VC$ we must download every e_{ij}. The length of each broadcast cycle is $\delta + 3$, so the total access latency is $k(\delta + 3)$. In each broadcast cycle, the client will switch twice (except the last cycle), so $h = 2k - 1$.

$\Longleftarrow$: Assume MCDR has a valid schedule S with $t = k(\delta + 3)$ and $h = 2k - 1$. Let us consider D_b first. There are k b_i's located at the first time unit on k different channels. It means we have to switch at least $k - 1$ hops to download D_b, and then we only have another k hops for D_e, which means we can visit at most k channels in $\{v_i\}$. At the beginning of each broadcast cycle, we always stay at some channel b_i to download d_i, and then we switch to some v_i, and at the end of this cycle, we have to switch to channel b_{i+1} for d_{i+1}. This means we cannot switch to two vertex channels within one broadcast cycle, otherwise we cannot download $D = D_e \cup D_b$ in k iterations. Since S is valid, we visit k vertex channels and download all D_e data items, it means these k vertices form a vertex cover with size k.

This reduction can be done in polynomial time, and we can conclude that MCDR is NP-hard.

Randomized Algebraic Algorithm

To solve the above decision problem, we developed a randomized algebraic algorithm. It can detect if a given problem has a schedule to download all the requested data before time t and with at most h channel switchings in $O\left(2^k(nht)^{O(1)}\right)$ time, where n is the number of channels and k is the number of required data items. We also provide a fixed parameter tractable (FPT) algorithm with computational time $O\left(2^l(nht)^{O(1)}\right)$. It can determine whether there is a scheduling to download l data items from D in at most n time slots and at most h channel switches. Service provider can adjust n and h freely to fit their own requirement. We firstly give some preliminaries and then present our algorithms in detail.

Preliminaries

Here we introduce some notions about group algebra which are not often used in algorithm design.

Definition 2 Assume that $x_1, \ldots, x_k$ are variables in group algebra. Then,

1. A *monomial* has format $x_1^{a_1} x_2^{a_2} \ldots x_k^{a_k}$.
2. A *multilinear monomial* is a monomial such that each variable has degree exactly one. For example, $x_3x_5x_6$ is a multilinear monomial, but $x_3x_5^2x_6^3$ is not.
3. For a *polynomial* $p(x_1, \ldots, x_k)$, its *sum of product expansion* is $\sum_j p_j(x_1, \ldots, x_k)$, where each p_j is a monomial, which has a format $c_j x_1^{a_{j_1}} \ldots x_k^{a_{j_k}}$ with c_j respect to its coefficient.
4. $G_2 = (\{0, 1\}, +, \cdot)$ is a field with two elements $\{0, 1\}$ and two operations $+$ and $\cdot$. The addition operation is under the modular of 2 (mod 2).
5. Z_2^k is the group of binary k-vectors. Let w_0 denote the all-zero vector, which is the identity of Z_2^k, and then for every $v \in Z_2^k$, $v^2 = w_0$, $v \cdot w_0 = v$.

The operations between elements in the group algebra are standard.

Algorithm Description

The basic idea of our algebraic algorithm is that for each item $d_i \in D$, where D is the query data set, we create a variable x_i to represent it. Therefore, given $D = \{d_1, d_2, \ldots, d_k\}$, we construct a variable set $X = \{x_1, x_2, \ldots, x_k\}$. We then design a circuit $H_{t,h,n}$ such that a schedule without conflict will be generated by a multilinear monomial in the sum of product expansion of the circuit. The existence of schedules to download all the data items in D from the multiple channel set C is converted into the existence of multilinear monomials of $H_{t,h,n}$. Replace each variable by a specified binary vector which can remove all of the non-multilinear monomials by converting them to zero. Thus, the data retrieval problem is transformed into testing if a multivariate polynomial is zero. It is well known that randomized

algorithms can be used to check if a circuit is identical to zero in polynomial time. Thus, we have the following statements.

Lemma 1 *There is a polynomial time algorithm such that given a channel c_i, a time interval $[t_1, t_2]$, and an integer m, it constructs a circuit of polynomial $P_{i,t_1,t_2,m}$ such that for any subset $D' = \{d_{i_1}, \ldots, d_{i_m}\} \subseteq D$ which has a size of m and is downloadable in the time interval $[t_1, t_2]$ from channel c_i, the product expansion of $P_{i,t_1,t_2,m}$ contains a multilinear monomial $x_{i_1} x_{i_2} \ldots x_{i_m}$.*

Proof We can use a recursive way to compute the circuit $P_{i,t_1,t_2,m}$ in polynomial time.

1. $P_{i,t_1,t_2,0} = 0$.
2. $P_{i,t_1,t_2,1} = \sum_j x_j$, $x_j \subseteq X$, and the corresponding datad_j is entirely in the time interval $[t_1, t_2]$ of channel c_i.
3. $P_{i,t_1,t_2,l+1} = \sum_j x_j \cdot P_{i,t_1,t'_2,l} + P_{i,t_1,t'_2,l+1}$, d_j starts at time $t'_2 + 1$ and ends before time t_2 on channel c_i.

When computing $P_{i,t_1,t_2,l+1}$, x_j multiplies $P_{i,t_1,t'_2,l}$ is based on the case that d_j is downloadable from time $t'_2 + 1$ to t_2 in the final phase, and the other l data items are downloadable before time t'_2. The term $P_{i,t_1,t'_2,l+1}$ is the case that $l + 1$ items are downloaded before time t'_2. Note that the parameter m in $P_{i,t1,t2,m}$ controls the total number of data to be downloaded.

Definition 3 A subset data items $D' = \{d_{i_1}, \ldots, d_{i_m}\} \subseteq D$ is (i, t, h)-downloadable if we can download all data items in D' before time t, the total number of channel switches is at most h, and the last downloaded item is from channel c_i.

Lemma 2 *Given two integers t and h, there is a polynomial time algorithm to construct a circuit of polynomial $F_{i,t,h,m}$ such that for any (i, t, h)-downloadable subset $D' = \{d_{i_1}, \ldots, d_{i_m}\} \subseteq D$, the product expansion of $F_{i,t,h,m}$ contains a multilinear monomial $\left(x_{i_1}, \ldots, x_{i_m}\right) Y$, where Y is a multilinear monomial which does not include any variable in X.*

Proof We still use a recursive way to construct the circuit. Some additional variables are used as needed. Without loss of generality, we assume the data retrieval process start at time 0.

1. $F_{i,t,0,0} = 0$.
2. $F_{i,t,0,1} = P_{i,1,t,1} \cdot y_{i,t,0,1}$.
3. $F_{i,t,h'+1,m'+1} = y_{i,t,h'+1,m'+1,0} \left(\sum_{t'<t} F_{i,t',h'+1,m'} \cdot P_{i,t'+1,t,1}\right) + y_{i,t,h'+1,m'+1,1} \left(\sum_{j \neq i} \sum_{t'<t} F_{i,t'-1,h',m'} \cdot P_{i,t'+1,t,1}\right)$.

Then we can get Lemma 2 immediately.

The computation of $F_{i,t,h'+1,m'+1}$ is based on two cases, and we use two variables, $y_{i,t,h'+1,m'+1,0}$ and $y_{i,t,h'+1,m'+1,1}$, to mark them respectively. We now present an algorithm that involves one layer randomization to determine if there is a schedule to download all the data items in D before time t and with at most h channel switchings.

Theorem 2 *There is an $O\left(2^k (hnt)^{O(1)}\right)$ time randomized algorithm to determine if there is a scheduling to download $|D| = k$ data items before time t and the number of channel switches is at most h, where n is the total number of channels.*

Proof By Lemma 2, we can construct a circuit $H_{t,h,n} = \sum_{i=1}^{n} F_{i,t,h,k}$ in polynomial time. It is easy to see there is a scheduling for downloading the k data items before time t and with h channel switches, if and only if the sum product expansion of $H_{t,h,n}$ has a multilinear monomial $(x_1, \ldots, x_k)Y$.

We can replace each s_i by a vector $w_i = w_0^T + v_i^T$, where w_0 is the all-zero vector of dimension k and v_i is a binary vector of dimension k with its ith element being 1 and all other elements being 0. Assume $k = 3$, we define the following operations:

$$v_a \cdot v_b = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} (a_1 + b_1)(\text{mod}2) \\ (a_1 + b_2)(\text{mod}2) \\ (a_1 + b_3)(\text{mod}2) \end{pmatrix} \quad (1)$$

$$(v_a + v_b) \cdot v_c = v_a \cdot v_c + v_b \cdot v_c \quad (2)$$

By Eqs. 1 and 2, for any k-dimensional binary vector $w' = w_0 + v$, we have $w'^2 = w_0^2 + 2w_0 \cdot v + v^2 = w_0 + 2(w_0 \cdot v) + w_0 = 2(w_0 \cdot v) + 2w_0 = 0$, because of the coefficients are in the field of G_2. The replacement $x_i = w_i (i = 1, \ldots, m)$ makes all the non-multilinear monomials become zero. Meanwhile, all the multilinear monomials remain nonzero. Hence, it is clear that there is a scheduling to download all the data items in D before time t and with at most h channel switchings if and only if $H_{t,h,n|x_i=w_i(i=1,\ldots,k)}$ is a nonzero polynomial. The variables in Y makes it impossible to have cancelation when adding two identical multilinear monomials, which can be generated from different paths with variables in $\{x_1, \ldots, x_k\}$. It is well known that randomized algorithms can be used to check if a circuit is identical to zero in polynomial time [11, 12].

The algorithm generates less than 2^k terms during the computing process since there are at most 2^k distinct binary vectors. Therefore, the computational time is $O\left(2^k (nht)^{O(1)}\right)$.

Example Let $H_1 = x_1 x_2 y_1 + x_2^2 y_2$ and $H_2 = x_1^2 y_1 + x_2^2 y_2$. Consider the replacement $x_1 = w_1 = \binom{0}{0} + \binom{1}{0}$ and $x_2 = w_2 = \binom{0}{0} + \binom{0}{1}$. We have the following steps of operations.

$$\begin{aligned} &H_1|x_1 = w_1, x_2 = w_2 \\ &= \left(\binom{0}{0} + \binom{1}{0}\right)\left(\binom{0}{0} + \binom{0}{1}\right) y_1 \\ &\quad + \left(\binom{0}{0} + \binom{0}{1}\right)^2 y_2 \\ &= \left(\binom{0}{0} + \binom{0}{1} + \binom{1}{0} + \binom{1}{1}\right) y_1 \\ &\quad + \left(\binom{0}{0} + \binom{0}{1}\right) y_2 \\ &= \left(\binom{0}{0} + \binom{0}{1} + \binom{1}{0} + \binom{1}{1}\right) y_1 \\ &\quad + \left(2\binom{0}{0} + 2\binom{0}{1}\right) y_2 \\ &= \left(\binom{0}{0} + \binom{0}{1} + \binom{1}{0} + \binom{1}{1}\right) y_1 + (0+0) y_2 \\ &= \left(\binom{0}{0} + \binom{0}{1} + \binom{1}{0} + \binom{1}{1}\right) y_1 + 0 \\ &= \left(\binom{0}{0} + \binom{0}{1} + \binom{1}{0} + \binom{1}{1}\right) y_1 \\ &\neq 0 \end{aligned}$$

$$\begin{aligned} &H_2|x_1 = w_1, x_2 = w_2 \\ &= \left(\binom{0}{0} + \binom{1}{0}\right)^2 y_1 + \left(\binom{0}{0} + \binom{0}{1}\right)^2 y_2 \\ &= \left(\binom{0}{0} + \binom{1}{0} + \binom{1}{0} + \binom{0}{0}\right) y_1 \\ &\quad + \left(\binom{0}{0} + \binom{0}{1} + \binom{0}{1} + \binom{0}{0}\right) y_2 \\ &= \left(2\binom{0}{0} + 2\binom{1}{0}\right) y_1 + \left(2\binom{0}{0} + 2\binom{0}{1}\right) y_2 \\ &= (0+0) y_1 + (0+0) y_2 \\ &= 0 \end{aligned}$$

H_1 is a polynomial that contains a multilinear monomial. It becomes nonzero after replacement. H_2 is a polynomial that is without multilinear monomials. It becomes zero after the replacement. If we just down a subset of l data items in set D, we have the following theorem that involves two layers of randomization.

Theorem 3 *There is an $O\left(2^l (hnt)^{O(1)}\right)$ time randomized algorithm to determine if there is a scheduling to download l data items from D in at most t time units and at most h channel switches.*

Proof By Lemma 2, we can construct a polynomial $H_{t,h,l} = \sum_{i=1}^{l} F_{i,t,h,l}$ in polynomial time. Replace each x_i with a vector $w_i = w_0^T + v_i^T$, where w_0 is the all-zero vector of dimension l and v_i is a random distinct vector of dimension l. The replacement $x_i = w_i$ $(i = 1, \ldots, k)$ makes all monomials which has non-multilinear monomial at x part become zero.

Therefore, there is a scheduling to download l data items of D before time t and with the number of switches no more than h if and only if $H_{t,h,k|x_i=w_i(i=1,\ldots,k)}$ is not a zero polynomial in the field of G_2. Assume that the product expansion of $H_{t.h,l}$ has a multilinear monomial $(x_{i_1}, \ldots, x_{i_l})Y$, where Y is a multilinear monomial with variables not in $x_1, \ldots, x_k$. For a series of randomly assigned vectors with dimension l: $v_{j_1}, \ldots, v_{j_l}$, the probability that v_{j_i} is a linear combination of $v_{j_1}, \ldots, v_{j_{i-1}}$ is at most $\frac{2^{i-1}}{2^l} = \frac{1}{2^{l-i+1}}$. Therefore, with probability at most $\sum_{i=1}^{l} \frac{1}{2^{l-i+1}}$, v_{j_i} is a linear combination of $v_{j_1}, \ldots, v_{j_{i-1}}$ for some $i \leq l$. When $v_{j_i}, \ldots, v_{j_l}$ are linearly independent, the product of $v_{j_1}, \ldots, v_{j_l}$ is nonzero. Every multilinear monomial in the product expansion has different variables to form Y since it is determined by a unique path to generate the polynomial. Therefore, for those random vectors v_i, every multilinear monomial has a chance at least $1 - \frac{3}{4} = \frac{1}{4}$ to be nonzero. Therefore, if there is a solution, $H_{t,h,k|x_i=w_i(i,1,\ldots,l)}$ with random assignment Y is not zero in the field of G_2 with probability at least $\frac{1}{4}$.

After the replacements, it generates less than $2l$ terms since there are at most $2k$ different vectors for a group of Z_2^l. The coefficient of each vector is kept as a polynomial size circuit. Therefore, the computational time of our algorithm is $O\left(2^l (hnt)^{O(1)}\right)$, and if we run it 30 times, the error rate is $\left(\frac{3}{4}\right)^{30} < 0.0002$.

Applications

Scheduling problem is one of the most fundamental problems in combinatorial optimization, which could model various real-world practical applications. Especially, scheduling problem at client hand side would be very useful for data retrieval problem in wireless data broadcasting or data streaming environment to reduce energy consumption and improve query efficiency. Such problem would also be helpful for parallel query applications in distributed storage systems.

Open Problems

How to download data items efficiently in wireless data broadcast environment can usually be formulized as NP-hard problems with different constraints, and can be categorized into two kinds: single channel process and multiple channel process. The best known result for the former problem is constant-factor approximations, while currently there is no polynomial time approximation scheme (PTAS) for both of them. The results for this problem is also helpful for parallel data retrieval problem in distributed data storage system and cloud system.

Experimental Results

Many literature proposed experimental results for scheduling problem in data broadcasting. Shi et al. [6] simulated a base station with n broadcast channels and 10,000 items, each of size 1KB, and multiple clients with various requests of data. The access probability of the database follows Zipf distribution, n varies from 5 to 30, the number of antennae varies from 1 to 10, and the size of a request varies from 10 to 1,000. For each experiment, they generated 100 requests to get the average access latency and number of switchings during data retrieval. Lv et al. [7, 8] constructed two types of broadcast programs: special data broadcast without channel switching time (SDB) and general data broadcast with channel switching time (GDB). In both types of programs, they simulated a base station with n broadcast channels; the bandwidth of each channel is 1Mbit/sec. The database to be broadcasted has N data items, each of size 512 bytes. The time duration is denoted by t. The data items of query data set D

is generated with access probabilities following the Zipf distribution.

URLs to Code and Data Sets

Shi et al. [6] provided the program for users to test parameter setting for their own data sets and available channels (http://theory.utdallas.edu/dataengineering).

Cross-References

- Efficient Polynomial Time Approximation Scheme for Scheduling Jobs on Uniform Processors

Recommended Reading

1. Acharya S, Alonso R, Franklin M, Zdonik S (1995) Broadcast disks: data management for asymmetric communication environments. In: The ACM special interest group on management of data conference (SIGMOD), San Jose, 22–25 May 1995, pp 199–210
2. Yee W, Navathe S, Omiecinski E, Jermaine C (2002) Efficient data allocation over multiple channels at broadcast servers. IEEE Trans Comput 51(10):1231–1236
3. Ardizzoni E, Bertossi A, Pinotti M, Ramaprasad S, Rizzi R, Shashanka M (2005) Optimal skewed data allocation on multiple channels with flat broadcast per channel. IEEE Trans Comput 54(5):558–572
4. Anticaglia S, Barsi F, Bertossi A, Iamele L, Pinotti M (2008) Efficient heuristics for data broadcasting on multiple channels. Wirel Netw 14(2):219–231
5. Kenyon C, Schabanel N (1999) The data broadcast problem with non-uniform transmission times. In: Proceedings of the tenth annual ACM-SIAM symposium on discrete algorithms (SODA), Baltimore, 17–19 Jan 1999, pp 547–556
6. Shi Y, Gao X, Zhong J, Wu W (2010) Efficient parallel data retrieval protocols with MIMO antennae for data broadcast in 4G wireless communications. In: The 21st international conference on database and expert systems applications (DEXA), Bilbao, 30 Aug–3 Sept 2010, pp 80–95
7. Lu Z, Shi Y, Wu W, Fu B (2012) Efficient data retrieval scheduling for multi-channel wireless data broadcast. In: International conference on computer communications (INFOCOM), Orlando, 25–30 Mar 2012, pp 891–899
8. Lu Z, Shi Y, Wu W, Fu B (2014) Data retrieval scheduling for multi-Item requests in multi-channel wirelessBroadcast environments. IEEE Trans Mobile Comput 13(4):752–765
9. Gao X, Lu Z, Wu W, Fu B (2011) Algebraic algorithm for scheduling data retrieval in multi-channel wireless data broadcast environments. In: The 6th annual international conference on combinatorial optimization and applications (COCOA), Zhangjiajie, 4–6 Aug 2011, pp 74–81
10. Gao X, Lu Z, Wu W, Fu B (2013) Algebraic data retrieval algorithms for multi-channel wireless data broadcast. Theor Comput Sci 497:123–130
11. Williams R (2009) Finding paths of length k in $O*(2^k)time$. Inf Process Lett 109(6):315–318
12. Koutis I (2008) Faster algebraic algorithms for path and packing problems. In: The 2008 international colloquium on automata, languages and programming (ICALP), Reykjavik,6– 13 July 2008, pp 575–586

Scheduling with a Reordering Buffer

Matthias Englert[1] and Matthias Westermann[2]
[1]Department of Computer Science, University of Warwick, Coventry, UK
[2]Department of Computer Science, TU Dortmund University, Dortmund, Germany

Keywords

Machine scheduling; Minimum makespan scheduling; Online algorithms; Reordering buffer; Sorting buffer

Years and Authors of Summarized Original Work

2002; Räcke, Sohler, Westermann
2009; Gamzu, Segev
2010; Englert, Räcke, Westermann
2011; Adamaszek, Czumaj, Englert, Räcke
2011; Dósa, Epstein
2013; Avigdor-Elgrabli, Rabani
2014; Englert, Özmen, Westermann

Problem Definition

The problem known as the reordering buffer problem or as the sorting buffer problem is concerned with sorting a sequence of colored

items according to their color using a limited size buffer. More precisely the items are to be processed and arrive one by one. Arriving items must first be placed into a buffer which can hold up to k items. Once the buffer is completely filled, an algorithm has to free space by selecting one of the items in the buffer for processing and removing that item from the buffer. After items stop arriving, the remaining items in the buffer may be processed in any order. Whenever an item is processed that has a different color than the item processed in the step before, this generates a cost of 1. The objective is to minimize the total cost.

Metric Space Generalization

This problem can be further generalized. Items, instead of having a color, correspond to points in a metric space. A single server must process all items. In order to process an item, the server has to move to the corresponding point in the metric space. At every point, the server has to chose one of the first k as of yet unprocessed items for processing and move the server accordingly. The goal is to minimize the total distance the server travels.

The uniform metric in which any two points either have distance 0 or distance 1 from one another corresponds to the original "color sorting" setting. Other metrics studied include line metrics and "star" metrics which are the distance metrics over weighted undirected trees of diameter 2.

Block Operation Setting

Another variant is the so-called block operation setting. Once again, the input consists of a sequence of colored items. The first k items are placed in a buffer. In each step, an algorithm selects one of the colors and processes all items of that color currently stored in the buffer, incurring a cost of 1. This is called a block operation. The processed items are removed from the buffer and replaced with the next items from the input sequence (if there are any). The goal is once again to minimize the total cost.

The difference between this block operation setting and the original setting is most pronounced for an input sequence consisting of ℓ items of a single color. While in the original setting such a sequence would not produce any cost, the cost in the block device setting would be ℓ/k since only k items can be processed per block operation.

Minor Variants Found in the Literature

In some cases, there are slight differences in which these problems are defined in the literature. Does a cost incur for the first ever processed item or, similarly, is the first position of the server in the metric space part of the input or does the algorithm get to chose that position (without incurring any cost)? Does an item first have to be placed in the buffer or can the algorithm process an arriving item directly, thereby bypassing the buffer? Do we need to remove the remaining items in the buffer once new items stopped arriving? It turns out however that these details are inconsequential for most of the results we are interested in.

Key Results

The main focus of study in the area of scheduling with a reordering buffer has been on online algorithms. In the online setting, the algorithm's decisions have to be based solely on the items that arrived in the past and must not depend on items arriving in the future. An online algorithm is called c-competitive if the cost of the algorithm is at most c times that of an optimal off-line solution.

The Online Problem

Deterministic Algorithms

Räcke, Sohler, and Westermann [29] first introduced the problem for the uniform metric

and gave a $O(\log^2 k)$-competitive online algorithm. After further improvements to $O(\log k)$ [18] and $O(\log k / \log\log k)$ [6] eventually, an $O(\sqrt{\log k})$-competitive online algorithms was designed [2]. This is almost optimal since a lower bound of $\Omega(\sqrt{\log k / \log\log k})$ is known [2]. Many of these upper bounds also generalize from the uniform metric to star metrics.

While the proof techniques for some of these results differ significantly, the basic idea behind all the algorithms is the same. As long as the buffer contains an item of the same color as the previously processed item, such an item is processed next. Otherwise, the algorithm has to pick a different color and performs a color switch which incurs a cost of 1. In order to decide which color to switch to, each color is assigned a "penalty" counter which is initially set to 0 and is reset to 0 whenever the color is selected for processing. If there is a color with penalty at least k, then an item with that color is selected next. Otherwise, an arbitrary color is selected and the penalty counters for each color are increased proportional to the number of items of that color that are stored in the buffer. For the $O(\sqrt{\log k})$-competitive algorithm, instead of picking an arbitrary color, a more sophisticated rule is used.

Randomized Algorithms

Randomized algorithms can achieve much smaller competitive ratios. The first randomized algorithm with a competitive ratio of $O(\log\log k)$ was given for the block operation model [3]. Shortly afterward, a randomized algorithm with the same competitive ratio was presented for the original model [8]. This is best possible since a matching lower bound is known [2]. These randomized algorithms are based on online primal-dual LP schemes [11].

Other Metric Spaces

Apart from the uniform metric, line metrics have received some attention. After a randomized $O(\log^2 n)$-competitive online algorithm for n equally spaced points on a line [27], an improved deterministic $O(\log n)$-competitive algorithm was given [24]. A deterministic $O(\log N \log\log N)$-competitive algorithm for a line metric with not necessarily equally spaced points was also given, but here N refers to the number of items in the input sequence [24]. An easy observation, however, shaves off the $\log\log N$ factor and improves the analysis to show $O(\log N)$ competitiveness (Cygan, Mucha, Private communication, 2011). There is still a significant gap between this upper bound and the best known lower bound of about 2.154 [24].

For general metric spaces, a randomized $O(\log^2 k \log n)$-competitive online algorithm is known, where n is the number of points in the metric space [19]. This result is based on a deterministic algorithm for trees that is turned into an algorithm for general metrics by using a metric embedding [23].

Stochastic Inputs

In a setting where the input is not adversarial constructed but where the colors of the items are drawn i.i.d. from an unknown distribution, a constant competitive ratio is achievable [22]. This result also holds when the colors of the items are fixed by an adversary but the order in which the items arrive is random. The proof is based on the fact that a constant competitive online algorithm is known for adversarial inputs, if the online algorithm can use a buffer that is four times as large as the one used by the optimal off-line algorithm. In the stochastic input setting, this difference in buffer size does not lead to significantly different cost, i.e., the cost of an optimal algorithm with buffer size k is only by a constant factor larger than the cost of an optimal algorithm with buffer size $4k$. This is not true for adversarial inputs [1].

The Off-Line Problem

The reordering buffer problem is NP-hard [5, 12] for the uniform metric, and the complexity for line metrics is unknown. Therefore, several papers focus on approximating the off-line scenario.

A constant factor approximation is known for the uniform metric [7]. For star metrics, the best known approximation factor of $O(\log\log k\gamma)$ is

achieved by a randomized algorithm, where γ denotes the ratio of the maximum to the minimum weight [25]. Both results are based on the intricate rounding of the solution to an LP relaxation of the corresponding problem.

Bicriteria Approximations

For more general metric spaces, the best approximation ratios are achieved by bicriteria approximations, i.e., the approximation algorithm can make use of more buffer capacity than an optimal algorithm. For metric spaces given by the distance metric over a weighted undirected tree, a bicriteria approximation with approximation factor 9 to cost and $4 + 1/k$ to buffer size is known [10]. Using metric embeddings [23], this implies a randomized bicriteria approximation with approximation factor $O(\log n)$ to cost and $O(1)$ to buffer size, where n denotes the number of points in the metric space.

The Maximization Problem

In the maximization version of the problem, the goal is to maximize the total cost savings that result from reordering the input sequence. In terms of an optimal solution, the minimization and maximization scenario are identical. However, in terms of approximation, they behave quite differently in the sense that a c-approximate solution for the maximization problem usually has very different cost from a c-approximate solution for the minimization problem. For the uniform metric, the first result was an approximation algorithm with an approximation factor of 20 [28]. This was later improved to a factor of 9 [9].

Online Minimum Makespan Scheduling

Reordering buffers have also been studied in connection with other scheduling problems, in particular online minimum makespan scheduling. As in the classic problem without reordering, the input consists of a sequence of jobs with processing times, and a scheduling algorithm has to assign the jobs to m parallel machines, with the objective to minimize the makespan, which is the time it takes until all jobs are processed. However, it is not required that each arriving job has to be assigned immediately to one of the machines. A reordering buffer can be used to reorder the input sequence of jobs. At each point in time, the reordering buffer contains the first k jobs of the input sequence that have not been assigned so far. An online scheduling algorithm has to decide which job to assign to which machine next. Upon its decision, the corresponding job is removed from the buffer and assigned to the corresponding machine, and thereafter the next job in the input sequence takes its place.

Non-preemptive Scheduling

For non-preemptive scheduling, Englert, Özmen, and Westermann [20] give, for m identical machines, a tight bound on the competitive ratio. Depending on m, the achieved competitive ratio lies between 4/3 and 1.4659. This optimal ratio is achieved with a buffer of size of at most $\lceil 2.5 \cdot m \rceil + 2$. They show that larger buffer sizes do not result in an additional advantage and that a buffer of size $\Omega(m)$ is necessary to achieve this competitive ratio. This improves upon an optimal algorithm for two identical machines [26].

Further, they present several algorithms for different buffer sizes. In addition, for m uniformly related machines, they give a scheduling algorithm that achieves a competitive ratio of 2 with a reordering buffer of size m.

Subsequently to [20], a variety of related papers appeared (compare, e.g., [4, 14–16, 21]). For 2 uniformly related machines with speed ratio $s \geq 1$, it is shown that, for any $s > 1$, a buffer of size 3 is sufficient to achieve an optimal competitive ratio, and in the case $s \geq 2$, a buffer of size 2 already allows to achieve an optimal ratio [15].

Job Migrations

The results of [20] can be generalized to the problem of online minimum makespan scheduling with job migrations, i.e., where no reordering buffer is available, but a limited number of job reassignments may be performed. For m identical

machines, the same competitive ratio as in [20] can be achieved [4]. The algorithm uses, for $m \geq 11$, at most $7m$ migration operations and, for smaller m, $8m$ to $10m$ migration operations. A number of papers consider similar models (compare, e.g., [13, 17, 30, 31]).

Preemptive Scheduling

For preemptive scheduling on m identical machines, tight bounds on the competitive ratio can be achieved for any m. This bound is $4/3$ for even values of m and slightly lower for odd values of m [16]. A buffer of size $\Theta(m)$ is sufficient to achieve this bound, but a buffer of size $o(m)$ does not reduce the best overall competitive ratio $e/(e-1)$ that is known for the case without reordering [16].

Cross-References

- Approximation Schemes for Makespan Minimization
- Efficient Polynomial Time Approximation Scheme for Scheduling Jobs on Uniform Processors
- Online Preemptive Scheduling on Parallel Machines

Recommended Reading

1. Aboud A (2008) Correlation clustering with penalties and approximating the reordering buffer management problem. Master's thesis, Computer Science Department, The Technion—Israel Institute of Technology
2. Adamaszek A, Czumaj A, Englert M, Räcke H (2011) Almost tight bounds for reordering buffer management. In: Proceedings of the 43rd ACM symposium on theory of computing (STOC), San Jose, pp 607–616
3. Adamaszek A, Czumaj A, Englert M, Räcke H (2012) Optimal online buffer scheduling for block devices. In: Proceedings of the 44th ACM symposium on theory of computing (STOC), New York, pp 589–598
4. Albers S, Hellwig M (2012) On the value of job migration in online makespan minimization. In: Proceedings of the 20th European symposium on algorithms (ESA), Ljubljana, pp 84–95
5. Asahiro Y, Kawahara K, Miyano E (2012) NP-hardness of the sorting buffer problem on the uniform metric. Discret Appl Math 160(10–11):1453–1464
6. Avigdor-Elgrabli N, Rabani Y (2010) An improved competitive algorithm for reordering buffer management. In: Proceedings of the 21st ACM-SIAM symposium on discrete algorithms (SODA), Austin, pp 13–21
7. Avigdor-Elgrabli N, Rabani Y (2013) A constant factor approximation algorithm for reordering buffer management. In: Proceedings of the 24th ACM-SIAM symposium on discrete algorithms (SODA), New Orleans, pp 973–984
8. Avigdor-Elgrabli N, Rabani Y (2013) An optimal randomized online algorithm for reordering buffer management. In: Proceedings of the 54th IEEE symposium on foundations of computer science (FOCS), Berkeley, pp 1–10
9. Bar-Yehuda R, Laserson J (2007) Exploiting locality: approximating sorting buffers. J Discret Algorithms 5(4):729–738
10. Barman S, Chawla S, Umboh S (2012) A bicriteria approximation for the reordering buffer problem. In: Proceedings of the 20th European symposium on algorithms (ESA), Ljubljana, pp 157–168
11. Buchbinder N, Naor J (2009) The design of competitive online algorithms via a primal-dual approach. Found Trends Theor Comput Sci 3(2–3):93–263
12. Chan H, Megow N, Sitters R, van Stee R (2012) A note on sorting buffers offline. Theor Comput Sci 423:11–18
13. Chen X, Lan Y, Benko A, Dósa G, Han X (2011) Optimal algorithms for online scheduling with bounded rearrangement at the end. Theor Comput Sci 412(45):6269–6278
14. Ding N, Lan Y, Chen X, Dósa G, Guo H, Han X (2014) Online minimum makespan scheduling with a buffer. Int J Found Comput Sci 25(5):525–536
15. Dósa G, Epstein L (2010) Online scheduling with a buffer on related machines. J Comb Optim 20(2):161–179
16. Dósa G, Epstein L (2011) Preemptive online scheduling with reordering. SIAM J Discret Math 25(1):21–49
17. Dósa G, Wang Y, Han X, Guo H (2011) Online scheduling with rearrangement on two related machines. Theor Comput Sci 412(8–10):642–653
18. Englert M, Westermann M (2005) Reordering buffer management for non-uniform cost models. In: Proceedings of the 32nd international colloquium on automata, languages and programming (ICALP), Lisbon, pp 627–638
19. Englert M, Räcke H, Westermann M (2010) Reordering buffers for general metric spaces. Theory Comput 6(1):27–46
20. Englert M, Özmen D, Westermann M (2014) The power of reordering for online minimum makespan scheduling. SIAM J Comput 43(3):1220–1237
21. Epstein L, Levin A, van Stee R (2011) Max-min online allocations with a reordering buffer. SIAM J Discret Math 25(3):1230–1250

22. Esfandiari H, Hajiaghayi M, Khani MR, Liaghat V, Mahini H, Räcke H (2014) Online stochastic reordering buffer scheduling. In: Proceedings of the 41st international colloquium on automata, languages and programming (ICALP), Copenhagen, pp 465–476
23. Fakcharoenphol J, Rao SB, Talwar K (2004) A tight bound on approximating arbitrary metrics by tree metrics. J Comput Syst Sci 69(3):485–497
24. Gamzu I, Segev D (2009) Improved online algorithms for the sorting buffer problem on line metrics. ACM Trans Algorithms 6(1):15:1–15:14
25. Im S, Moseley B (2014) New approximations for reordering buffer management. In: Proceedings of the 25th ACM-SIAM symposium on discrete algorithms (SODA), Portland, pp 1093–1111
26. Kellerer H, Kotov V, Speranza MG, Tuza Z (1997) Semi on-line algorithms for the partition problem. Oper Res Lett 21(5):235–242
27. Khandekar R, Pandit V (2010) Online and offline algorithms for the sorting buffers problem on the line metric. J Discret Algorithms 8(1):24–35
28. Kohrt JS, Pruhs K (2004) A constant factor approximation algorithm for sorting buffers. In: Proceedings of the 6th Latin American symposium on theoretical informatics (LATIN), Buenos Aires, pp 193–202
29. Räcke H, Sohler C, Westermann M (2002) Online scheduling for sorting buffers. In: Proceedings of the 10th European symposium on algorithms (ESA), Rome, pp 820–832
30. Tan Z, Yu S (2008) Online scheduling with reassignment. Oper Res Lett 36(2):250–254
31. Wang Y, Benko A, Chen X, Dósa G, Guo H, Han X, Sik-Lányi C (2012) Online scheduling with one rearrangement at the end: revisited. Inform Process Lett 112(16):641–645

Secretary Problems and Online Auctions

MohammadHossein Bateni
Google Inc., New York, NY, USA

Keywords

Competitive analysis; Knapsack constraint; Matroid constraint; Mechanism design; Online algorithm; Secretary problem; Strategyproof; Submodularity; Truthfulness

Years and Authors of Summarized Original Work

2004; Hajiaghayi, Kleinberg, Parkes
2008; Babaioff, Immorlica, Kempe, Kleinberg
2013; Bateni, Hajiaghayi, Zadimoghaddam

Problem Definition

The classic secretary problem, a prime example of stopping theory, has been studied extensively in the computer science literature. Consider the scenario where an employer is interested in hiring one secretary out of a pool of candidates. The difficulty is that, although the employer does not know the utility of a candidate before she is interviewed, the irrevocable hiring decision for each candidate has to be made right after the interview and prior to interviewing the subsequent candidates. The goal is nonetheless to pick the best candidate or maximize the probability of achieving this.

Optimization Angle

The above scenario is hopeless from an algorithmic point of view since an adversarial input makes it impossible to hire the best candidate. We can take either of two paths to make the problem tractable: restrict the set of utilities or the arrival order of candidates. The former path yields, for instance, the stochastic variant of the problem. However, we follow the second idea here that leads to the classic secretary problem. The extra assumption, then, is that the candidates arrive in a random error; i.e., although each candidate may have an arbitrary adversarial utility, every permutation of the candidates is equally likely to be the arrival order.

A folklore solution to the problem, often attributed to [3], is to look into the first $\frac{1}{e}$ fraction of the candidates (called the "tuning set"), without giving them any offers, and then hire the first candidate with utility more than every one in the tuning set. It is not difficult to show that this approach hires the best candidate with probability

at least $\frac{1}{e}$. Indeed, it is known that this is the best possible performance.

There are two questions to be answered, once we extend the problem to multiple secretaries.

1. *What subsets of secretaries can be hired together?* The simplest answer is to allow at most k secretaries to be hired. Alternately, we can place (several) knapsack and/or matroid constraints on the feasible set. The former assigns a cost to each hire – say, the requested salary – that is to be paid out of a given budget. The latter permits only those combinations that form an independent set according to a given matroid. It is easy to see that both generalize the cardinality constraint.
2. *How do we compute the utility of a set?* The utility of a set can be defined as the sum of the utilities of individual secretaries in the set. More generally, a submodular or subadditive function may be employed to describe the utility of a set.

We then attempt to hire a feasible set of secretaries of maximum expected utility.

Mechanism Design Angle

Mechanism design literature has looked at this problem from a slightly different angle. In this setting, the players that arrive in a random order declare a *bid* – i.e., how much they value the item being sold – and then the seller decides who should get the item (or items) and how much they should be charged. Such decisions are to be taken irrevocably as in the optimization problem discussed above.

The players can play strategically, though, by declaring higher or lower bids in order to increase their chances of winning the item or to reduce the price they pay. In addition, they may declare their arrival/departure time untruthfully to achieve a better result. We want to design a "truthful" auction that precludes such undesirable outcomes. Although we allow the player to declare any nonnegative bid (if it is in her favor), we do not let them state an arrival time that is earlier than their actual one. (Presence intervals may be overlapping and/or nested.) We say that a mechanism is value-strategyproof if no player can benefit from declaring a bid different from her real value. Similarly the mechanism is called time-strategyproof if there is no benefit in stating the arrival/departure times untruthfully. We look for mechanisms that are both time- and value-strategyproof.

Key Results

Optimization

Kleinberg [6] studies the multiple-choice generalization where the goal is to hire k candidates, whose total utility (defined as the sum of the individual utilities) is maximized. He presents a tight performance guarantee of $1 + \Theta\left(\frac{1}{\sqrt{k}}\right)$ for the problem. In the case of $k = 1$, this is equivalent to the classic secretary problem. (The nontrivial direction follows from a construction where the utilities are hugely different.) Kleinberg's algorithm partitions the set of candidates into two (almost) equal pieces, recursively hires $\frac{k}{2}$ secretaries in the first, sets the threshold for the second piece by looking at the solution to the first piece, and picks as many as $\frac{k}{2}$ secretaries in the second piece who are better than threshold.

Babaioff et al. [1] look at the generalization where there is a restriction on the set of candidates that can be hired together; the restriction is in the form of a matroid. They present an $O(\log n)$ competitive ratio in this case along with improved bounds when the matroid has a special form. Their general matroid algorithm partitions the items into logarithmically many sets of almost equal utility and focuses (randomly) on one such set, which reduces the problem into that of maximizing the cardinality of the solution (solved via the greedy method).

The case of submodular utilities is discussed in Bateni et al. [2]: several matroid or knapsack constraints can be placed on the set of feasible candidates, and the total utility of a set is computed by a submodular function of the participating candidates. They provide constant competitive ratios as long as a fixed number of knapsack

constraints are present. When (a constant number of) matroid constraints are involved, too, their performance guarantees grow to $O(\log^2 k)$ where k is the rank of the matroid. They divide the input into different pieces where at most one secretary should be picked from each, not losing too much utility in the process. As a result, the submodular function collapses to an additive one within each piece (by taking the marginal values of secretaries with respect to the current solution). The classic algorithm is then used inside each piece. The main idea behind the matroid algorithm is that we only need to show that, whatever choices we have already committed to, there are enough options left that can appropriately augment the current solution. The argument goes by proving the existence of a magical solution with k' secretaries any of whose $\frac{k'}{2}$-size subsets has significant contribution (say, at least a $\frac{1}{\log k}$ fraction of the optimum) in the submodular function. Had we known k', a simple greedy algorithm would have sufficed to find a solution similar to the magical set. At the cost of another factor $O(\log k)$, we can guess k'.

Furthermore, Bateni et al. show that subadditive utility functions make the problem much more difficult. In particular, they provide matching $\Theta(\sqrt{k})$ competitive ratios.

Mechanism Design

The Dynkin's algorithm for the classic secretary problem can be readily turned into an auction: set the price after observing the tuning set, and then sell to anyone with a higher bid. This mechanism is not truthful, though, since high-bid players spanning across the time threshold have an incentive to declare later arrival time (i.e., after the threshold); this way, they will win the item but do not set the price.

Nevertheless, Hajiaghayi et al. [5] show how one can modify the mechanism slightly to make it truthful: after the threshold, consider the option of selling the item to the agent with the highest bid so far – if she is still present – and charge her the second-highest bid so far. Their method achieves constant competitiveness for both efficiency and revenue. Their $1/e$ competitiveness for efficiency is best possible since it generalizes the optimization problem; however, when comparing the revenue to that achieved by the Vickrey auction, their upper bound of $1/e^2$ for competitiveness fares against a lower bound of $1/e$. (It is possible, they show, to modify the mechanism slightly to trade efficiency loss for revenue gain; for instance, simultaneous 4 competitiveness for both objectives is possible.)

The general idea for the transformation is to define a "tuning period" where the price is set for everyone. Then, not only a simple auction-like mechanism is employed in the "hiring phase" to obtain a strategyproof mechanism, but also extra care should be given to the "transition phase" (from tuning to hiring) so as not to incentivize untruthful declaration of arrival time for those whose presence spans the transition. The same approach can be applied to the multiple-choice secretary problem to obtain constant-factor competitive mechanisms (for efficiency and revenue), but this bound is far from the one achieved in the optimization setting by Kleinberg [6].

Open Problems

Though there has been some improvements on the matroid case, we still do not know which cases are hard and admit no constant-factor competitive ratio. For submodular utilities (and simple cardinality constraints), in particular, there is a gap between $\left(1-\frac{1}{e}\right)/(e+1)$ algorithmic result [4] and the $1-\frac{1}{e}$ $\left(\text{or } 1-\frac{1}{\sqrt{k}}\right)$ target known for linear utilities.

Cross-References

▸ Algorithmic Mechanism Design

Recommended Reading

1. Babaioff M, Immorlica N, Kempe D, Kleinberg R (2008) Online auctions and generalized secretary problems. SIGecom Exch 7(2):1–11. doi:http://doi.acm.org/10.1145/1399589.1399596

2. Bateni M, Hajiaghayi MT, Zadimoghaddam M (2013) Submodular secretary problem and extensions. ACM Trans Algorithms 9(4):32
3. Dynkin EB (1963) The optimum choice of the instant for stopping a markov process. Sov Math Dokl 4:627–629
4. Feldman M, Naor J, Schwartz R (2011) Improved competitive ratios for submodular secretary problems (extended abstract). In: APPROX, Princeton, pp 218–229
5. Hajiaghayi MT, Kleinberg R, Parkes DC (2004) Adaptive limited-supply online auctions. In: EC, New York, pp 71–80
6. Kleinberg R (2005) A multiple-choice secretary algorithm with applications to online auctions. In: SODA, Vancouver, pp 630–631

Self-Assembly at Temperature 1

Pierre-Étienne Meunier
Le Laboratoire d'Informatique Fondamentale de Marseille (LIF), Aix-Marseille Université, Marseille, France

Keywords

Computational geometry; Concurrency; Self-assembly

Years and Authors of Summarized Original Work

2000; Rothemund, Winfree
2009; Doty, Patitz, Summers
2011; Cook, Fu, Schweller
2014; Meunier, Patitz, Summers, Theyssier, Winslow, Woods

Problem Definition

Temperature 1 (also called *noncooperative*) self-assembly is a model of the formation of structures by growing and branching tips. Despite its ubiquity in nature (in systems such as plants and mycelium or percolation processes) and apparent dynamic simplicity, it is one of the least understood models of self-assembly.

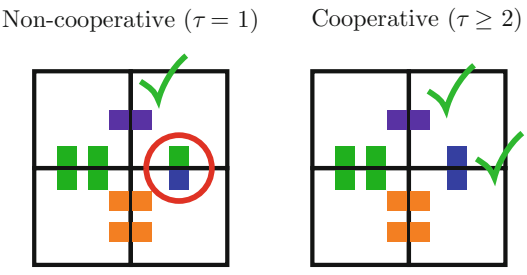
This model was introduced in a broader framework called the *abstract Tile Assembly Model* (aTAM) [10]. In the aTAM, we consider *tile assembly systems*, which are defined by a finite set T of square or cubic *tile types*, an initial *seed assembly* σ (one or more tiles stuck together), and an integer temperature $\tau = 1, 2, 3, \ldots$. All tiles, on each of their sides, have *glues* with an integer *color* and an integer *strength*.

The dynamics of tile self-assembly starts from the seed assembly and proceeds one tile at a time, asynchronously and nondeterministically. A tile can stick to an existing assembly if it can be placed so that the sum of the strengths on its sides matching the existing assembly is at least the temperature. In the case of temperature 1, this means that tiles can be placed as soon as one of their sides matches the existing assembly. At higher temperatures, we can require that newly placed tiles match *several* of their neighbors to attach.

Ultimately, after a countable (potentially infinite) number of steps, no tile can be added to the assembly, in which case we call it *terminal*. Like in Wang tilings, tiles cannot overlap, be rotated, or be flipped. However, tiles can have *mismatches* with their neighbors (Fig. 1).

Key Results

The first comparison between temperatures 1 and 2 was shown by Rothemund and Winfree [9], with the motivation of computing and efficiently



Self-Assembly at Temperature 1, Fig. 1 In the non-cooperative model, tiles can attach as soon as one side matches the neighborhood

building arbitrary shapes at the nanoscale. In this context, the generally accepted definition of "efficient" is *with significantly less tile types than the size of the output*.

Assembling Simple Shapes Efficiently

The first step toward these goals is the programming of simple shapes like squares or trees. At temperature ≥ 2, constructions with Turing machines can be used to show the following bound:

Theorem 1 (from [9]) *The smallest two-dimensional tileset T_n producing only squares of size $n \times n$ from a single-tile seed is of size $\theta\left(\frac{\log n}{\log \log n}\right)$.*

The smallest number of tile types that can assemble exactly a set of shapes is called the *tile complexity* of that set. In the noncooperative model, the following upper bound is known:

Theorem 2 (from [9]) *For all integer n, there is a tileset T_n, of size $2n - 1$, that produces only squares of size $n \times n$ from a single-tile seed.*

Whether this upper bound is optimal is still one of the major open problems of the model, and little progress has been made since its identification. The real motivation behind this question is whether we (or natural systems) can perform useful computations with this model.

Finding the smallest tileset for assembling an input shape can also be treated as an optimization problem: see Adleman et al. [1] for the case of tree shapes.

The Role of Geometry

A partial answer to this question was found by Cook, Fu, and Schweller [2], who tried to "fake" cooperation by blocking the growth of some parts of the assembly. They introduced two different ways to do this: removing the planarity constraint and allowing errors.

In both cases, "faking" cooperation means producing the same assemblies as a temperature 2 tile assembly systems up to rescaling by a constant factor.

Three-Dimensional Noncooperative Self-Assembly

In three dimensions, temperature 1 self-assembly is able to simulate Turing computations:

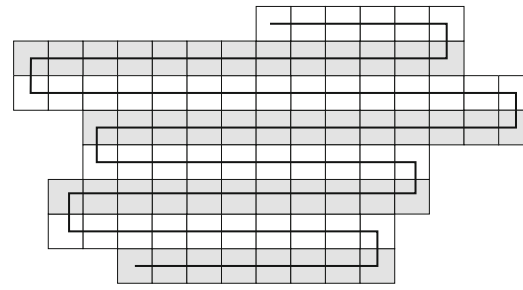
Theorem 3 (from [2]) *There is a three-dimensional tileset T such that for all Turing machine $\mathcal{M}$ and input $x \in \mathbb{N}$, there is a computable seed assembly $\sigma_{\mathcal{M},x}$ and a tile $t \in T$, such that all terminal assemblies of $(T, \sigma_{\mathcal{M},x}, 1)$ contain t if and only if $\mathcal{M}$ accepts input x.*

The construction simulates a Turing-universal cooperative tile assembly system called a *zigzag system*, in which rows grow on top of each other, alternatively to the left and to the right, using cooperation to copy and update the previous row (Fig. 2).

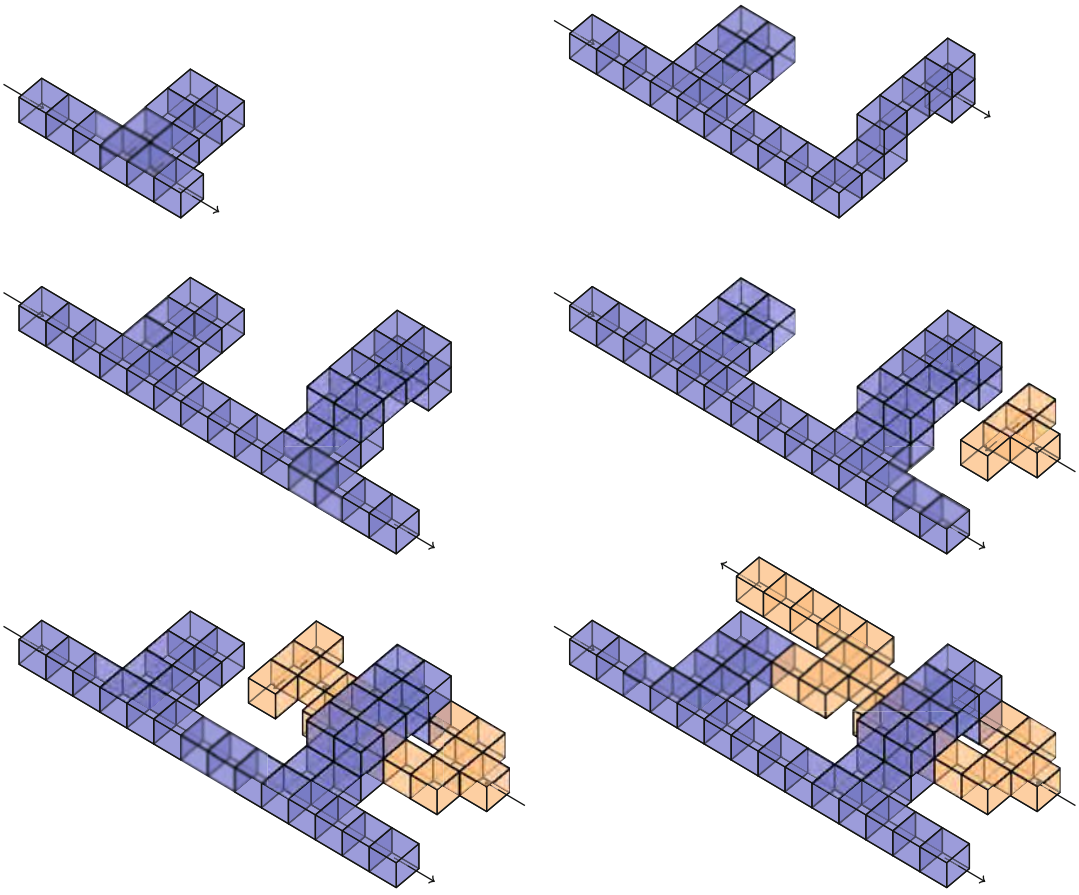
The idea is pictured on Fig. 3. A "main" path grows on each row, building "bridges" and "blockers" (in blue on Fig. 3) that encode bits. These bits can be read by the next row: before reading a bit, the main path (in orange on Fig. 3) of the row forks into two branches, respectively probing for a bridge (encoding a 1) and a blocker (encoding a 0). Exactly one branch passes through and can accumulate successive bits in its state, until a full tile has been read. Then, it rewrites bits encoding the next tile for the row above.

Allowing Erroneous Blocking

Adapting the mechanism used in the 3D construction to the planar case is widely conjectured impossible [9], because allowing the "wrong" branch to grow and collide against a previous



Self-Assembly at Temperature 1, Fig. 2 An example zigzag system (Figure from [2])



Self-Assembly at Temperature 1, Fig. 3 Bit selection in 3d

part of the assembly, in Fig. 3, *encloses* the other "correct" branch inside a finite portion of the plane.

However, it becomes possible if we consider a stochastic assembly schedule, where at each time step, exactly one tile attaches, and all tiles that can attach do so with equal probability. If we repeat the above construction k times consecutively, only one needs to succeed. We can therefore lower the probability of failure of each bit selection to 2^{-k}:

Theorem 4 (from [2]) *For all $\varepsilon > 0$ and all zigzag tile systems $\mathcal{T} = (T, s, 2)$, whose producible assemblies have size at most some constant r, there is a planar temperature 1 probabilistic tile assembly system $\mathcal{S}$ that simulates $\mathcal{T}$ without error with probability at least $1 - \varepsilon$.*

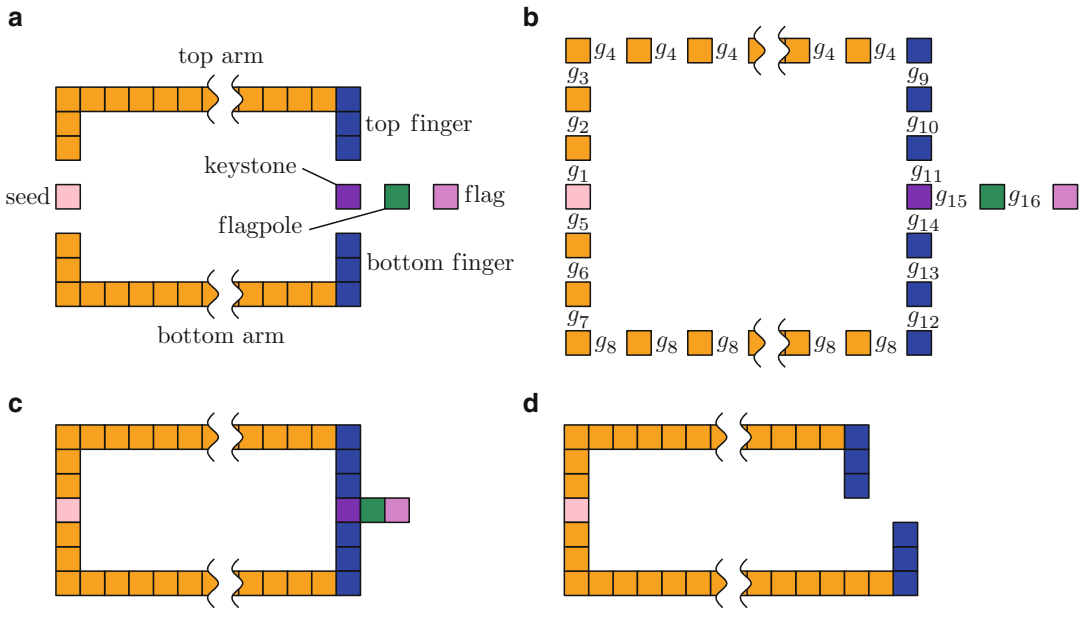
Of course, this construction means that the number of tile types and scaling factor will increase by a factor depending on ε and r.

Simulation up to Rescaling

One of the latest developments of tile assembly is the notion of *intrinsic universality* [3, 4, 11], a notion of simulation by rescaling only between tile assembly systems.

This idea is useful in particular to compare different models, because it provides qualitative properties to check, as opposed to quantitative properties such as tile complexity. The general argument is:

- At temperature 2 in two dimensions, there is a tileset known from [4] to be able to simu-



Self-Assembly at Temperature 1, Fig. 4 Tile assembly system $\mathcal{T}$ (Figure from [7])

late any other tile assembly system, modulo rescaling.

- However, there is a tile assembly system $\mathcal{T}$ that no tileset in model X (in our case, temperature 1) can simulate without errors.
- Therefore, model X is not as powerful as planar temperature 2.

This argument was used, for instance, to prove the first separation result between temperature 2 and the fully general model of temperature 1 [7]:

Theorem 5 (from [7]) *There is a planar (temperature 2) tile assembly system $\mathcal{T}$ (whose productions are pictured on Fig. 4) that no (two- or three- dimensional) tile assembly system $(A, \alpha, 1)$ can simulate up to rescaling.*

The proof uses a combinatorial argument (called the *window movie lemma*) to show that if there were a tile assembly system simulating all productions of $\mathcal{T}$, then it would also be able to produce other "illegal" assemblies (see Fig. 5) that do not represent any of $\mathcal{T}$'s producible assemblies.

Important Particular Cases

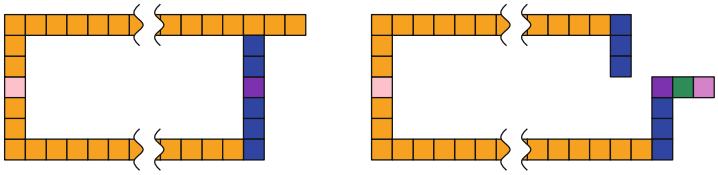
Noncooperative self-assembly, when restricted to dimension one, is similar to nondeterministic finite automata. It is therefore natural to look for a *pumping lemma*.

The first result in this direction was proven by Doty, Patitz, and Summers [5], who introduced the notion of *pumpable paths*: a path P is *pumpable* if it contains a subsegment $P_{i,i+1,\ldots,j}$ that can be repeated arbitrarily many (consecutive) times along $\overrightarrow{P_i P_j}$ while remaining self-avoiding.

Theorem 6 (from [5]) *Let $\mathcal{T}$ be a tile assembly system that assembles exactly one (potentially infinite) terminal assembly α. If any path in α, longer than a constant c, is pumpable, then there are finite families of vectors $\mathbf{b_1}, \ldots, \mathbf{b_n}$, $\mathbf{u_1}, \ldots, \mathbf{u_n}$, and $\mathbf{v_1}, \ldots, \mathbf{v_n} \in \mathbb{Z}^2$, such that:*

$$\mathrm{dom}(\alpha) = \bigcup_{1 \le i \le n} \{\mathbf{b_i} + j\,\mathbf{u_i} + k\,\mathbf{v_i} | j, k \in \mathbb{N}\}$$

In [5], examples were identified, of paths with segments that could be repeated, but only finitely many times due to collisions. The formalization of these examples was later done by Manuch,



Self-Assembly at Temperature 1, Fig. 5 Illegal productions, that a temperature 1 system that can simulate all productions of $\mathcal{T}$ must also be able to simulate (Figure modified from [7])

Stacho, and Stoll [6], proving lower bounds under the hypothesis that no mismatches can occur.

This approach was then extended by Reif and Song [8], to show that tile assembly systems without mismatches have a recursive set of productions. However, the decidability of the "no mismatches" hypothesis is still an open problem.

Applications

Given the successful experimental applications of tile self-assembly, particularly in the field of DNA nanotechnologies, it seems natural to try to implement them: indeed, intuition suggests that they would make no errors in cooperation tiles. However, no successful construction of noncooperative experiments has been reported; the reason might be that ensuring uniqueness of the seed is impossible, as any two tiles in solution together might bind, without any of them being bound to the seed.

Open Problems

Aside from understanding the exact geometric requirements for Turing universality, a number of open problems have been identified in this model:

1. From [9]: What is the tile complexity of squares of size $n \times n$ in the planar, temperature 1 model?

   A related problem, which has been in the folklore for some time, is the existence of a shape of tile complexity arbitrarily smaller than its Manhattan diameter.
2. From [5]: If $\mathcal{T}$ is a tile assembly system with exactly one terminal assembly, is there a constant c such that any path longer than c is pumpable?
3. From [7]: Is there a temperature 1 tile assembly system with a non-recursive set of productions?
4. From [7]: Is there a single tileset able to simulate any temperature 1 tile assembly system up to rescaling, using only noncooperative bindings?

Cross-References

- Experimental Implementation of Tile Assembly
- Intrinsic Universality in Self-Assembly

Recommended Reading

1. Adleman LM, Cheng Q, Goel A, Huang MDA, Kempe D, de Espanés PM, Rothemund PWK (2002) Combinatorial optimization problems in self-assembly. In: Proceedings of the thirty-fourth annual ACM symposium on theory of computing (STOC), Montréal, pp 23–32
2. Cook M, Fu Y, Schweller RT (2011) Temperature 1 self-assembly: deterministic assembly in 3D and probabilistic assembly in 2D. In: Proceedings of the 22nd annual ACM-SIAM symposium on discrete algorithms (SODA), San Francisco, pp 570–589, arxiv preprint: arXiv:0912.0027
3. Doty D, Lutz JH, Patitz MJ, Summers SM, Woods D (2009) Intrinsic universality in self-assembly. In: Proceedings of the 27th international symposium on theoretical aspects of computer science (STACS), Nancy, pp 275–286. arxiv preprint: arXiv:1001.0208
4. Doty D, Lutz JH, Patitz MJ, Schweller RT, Summers SM, Woods D (2012) The tile assembly model is intrinsically universal. In: Proceedings of the 53rd

annual IEEE symposium on foundations of computer science (FOCS), New Brunswick, pp 439–446. arxiv preprint: arXiv:1111.3097
5. Doty D, Patitz MJ, Summers SM (2009) Limitations of self-assembly at temperature 1. In: Proceedings of the fifteenth international meeting on DNA computing and molecular programming, Fayetteville, 8–11 June 2009, pp 283–294. arxiv preprint: arXiv:0906.3251
6. Maňuch J, Stacho L, Stoll C (2010) Two lower bounds for self-assemblies at temperature 1. J Comput Biol 17(6):841–852
7. Meunier PE, Patitz MJ, Summers SM, Theyssier G, Winslow A, Woods D (2014) Intrinsic universality in tile self-assembly requires cooperation. In: Proceedings of the 25th annual ACM-SIAM symposium on discrete algorithms (SODA), Portland, pp 752–771. arxiv preprint: arXiv:1304.1679
8. Reif JH, Song T (2013) Complexity and computability of temperature-1 tilings. In: FNANO 2013, poster abstract
9. Rothemund PWK, Winfree E (2000) The program-size complexity of self-assembled squares (extended abstract). In: Proceedings of the thirty-second annual ACM symposium on theory of computing (STOC), Portland. ACM, pp 459–468. doi:http://doi.acm.org/10.1145/335305.335358
10. Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology
11. Woods D (2013) Intrinsic universality and the computational power of self-assembly. In: Neary T, Cook M (eds) MCU, Zürich. EPTCS, vol 128, pp 16–22

Self-Assembly of Fractals

Matthew J. Patitz
Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR, USA

Keywords

Discrete self-similar fractals; Fractal dimension; Self-assembly; Tile Assembly Model

Years and Authors of Summarized Original Work

2009; Kautz, Lathrop
2009; Lathrop, Lutz, Summers
2010; Patitz, Summers
2012; Lutz, Shutters
2013; Kautz, Shutters
2014; Barth, Furcy, Summers, Totzke

Problem Definition

This problem is concerned with the self-assembly fractal patterns and structures. More specifically, it deals with discrete self-similar fractals and different notions of them self-assembling from tiles in the abstract Tile Assembly Model (aTAM) and derivative models. The self-assembly of fractals and fractal-like structures is particularly interesting due to their pervasiveness in nature, as well their complex aperiodic structures which result in them occupying less dimensional space than the space they are embedded within.

Using the terminology from [1], we define $\mathbb{N}_g$ as the subset $\{0, 1, \ldots, g-1\}$ of $\mathbb{N}$, and if $A, B \subseteq \mathbb{N}^2$ and $k \in \mathbb{N}$, then $A + kB = \{\mathbf{m} + k\mathbf{n} | \mathbf{m} \in A$ and $\mathbf{n} \in B\}$. We then define discrete self-similar fractals as follows.

We say that $\mathbf{X} \subset \mathbb{N}^2$ is a *discrete self-similar fractal* (or *dssf* for short) if there exist $1 < g \in \mathbb{N}$ and a set $\{(0,0)\} \subset G \subset \mathbb{N}_g^2$ with at least one point in every row and column, such that $\mathbf{X} = \bigcup_{i=1}^{\infty} X_i$, where X_i, the ith stage of $\mathbf{X}$, is defined by $X_1 = G$ and $X_{i+1} = X_i + g^i G$. We say that G is the generator of $\mathbf{X}$.

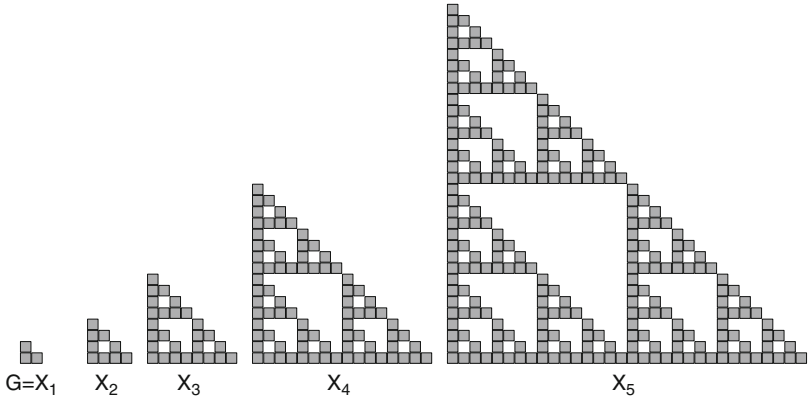
Figure 1 shows, as an example, the first 5 stages of the discrete self-similar fractal known as the Sierpinski triangle. In this example, $G = \{(0,0), (1,0), (0,1)\}$.

In general, we ask whether or not a given dssf $\mathbf{X}$ can self-assemble within a given model.

Variants

The general problem of determining whether or not a discrete self-similar fractal self-assembles within a given model has several variants, which determine the way in which the fractal shape is represented within a resulting assembly.

1. **Weak self-assembly.** If a dssf $\mathbf{X}$ weakly self-assembles using a tile set T, then there exists a subset of tile types $B \subseteq T$ such that, in



Self-Assembly of Fractals, Fig. 1 Example discrete self-similar fractal: the first 5 stages of the Sierpinski triangle

the terminal assembly α, for every point $\mathbf{p} \in \text{dom}\,\alpha$ such that $\mathbf{p} \in \mathbf{X}$, the tile type at location $\mathbf{p}$ in α is a type within B, and for every point $\mathbf{p} \in \text{dom}\,\alpha$ such that $\mathbf{p} \notin \mathbf{X}$, the tile type at location $\mathbf{p}$ in α is not within B. That is, the tile types in the subset B precisely "paint a picture" of $\mathbf{X}$, while tiles of types not in B may appear in locations outside of $\mathbf{X}$.

2. **Strict self-assembly.** If a dssf $\mathbf{X}$ strictly self-assembles, then it weakly self-assembles with $B = T$, i.e., the locations that exist in the domain of the terminal assembly are exactly those of $\mathbf{X}$.
3. **Approximate self-assembly.** A dssf $\mathbf{X}$, and thus a strictly self-assembled version of $\mathbf{X}$, has fractal dimension (i.e., zeta-dimension [3]) <2, and a weakly self-assembled version has dimension 2. Since it appears to be difficult if not impossible to strictly self-assemble many (or all) dssf's, it is interesting to consider if an approximation of a dssf $\mathbf{X}$ which retains the same fractal dimension as $\mathbf{X}$ can strictly self-assemble.

Key Results

Self-assembly of dssf's has been studied in all of the above variants and within the aTAM, 2HAM, and STAM [9]. As previously mentioned, the complexity of dssf's makes them interesting to study since they are infinite, aperiodic structures. This requires any system in which they self-assemble to rely on algorithmic self-assembly (rather than unique tile types hard coded to each position of the shape), and for this reason early experimental results even included the weak self-assembly of the initial few stages of the Sierpinski triangle [11] as a proof of concept that DNA-based tile implementations of the aTAM are capable of algorithmic self-assembly. Nonetheless, as infinite structures, dssf's are more often the focus of theoretical studies.

Weak Self-Assembly

As seen in [11], it is possible for a very simple tile set of only 7 tile types to weakly self-assemble the Sierpinski triangle. This tile set can essentially be thought of as computing the `xor` function on two inputs (i.e., $00 \to 0$, $01 \to 1$, $10 \to 1$, and $11 \to 0$), with the glues with which a tile initially binds to an assembly encoding the input bits and those to which tiles later attach encoding the output bits.

In [4] it was noted that another characterization of the Sierpinski triangle is as the nonzero residues modulo 2 of Pascal's triangle. They then provided a characterization of an infinite class of dssf's, known as *generalized Sierpinski carpets*, which can be defined as the residues, modulo a prime number, of the entries in a two-dimensional matrix generated by a simple recursive equation. (A well-known example among this class of

dssf's is the Sierpinski carpet.) They then proved that all generalized Sierpinski carpets weakly self-assemble in the aTAM.

Strict Self-Assembly

Although weak self-assembly of many dssf's can be achieved with very simple tile sets, it turns out that strict self-assembly is an entirely different, and much more difficult, problem. In fact, in [6] they proved that it is impossible for the Sierpinski triangle to strictly self-assemble in the aTAM. Furthermore, their proof showed that to be the case regardless of the temperature parameter. This result was extended in [10] to a proof that an infinite class of "pinch-point" dssf's, which includes the Sierpinski triangle, cannot strictly self-assemble in the aTAM at any temperature. Pinch-point fractals are those whose generators have exactly one point in their topmost row, the leftmost, and one in their eastmost column, the bottommost. Yet another extension was provided in [1], where the authors defined "tree" fractals, which again include the Sierpinski triangle, as those with generators which are trees and which have a single point in their topmost row and a single point in their rightmost column. They then proved that, regardless of the temperature or even of the scale factor, no tree fractal strictly self-assembles in the aTAM.

Additional results related to strict self-assembly of dssf's include the proof in [10] that in the aTAM at temperature 1 (i.e., systems with $\tau = 1$), it is impossible for any dssf to self-assemble within a locally deterministic system (see [12] for a definition of local determinism), and in [2] it was proven that the Sierpinski triangle also cannot self-assemble in the 2-Handed Assembly Model, at any temperature.

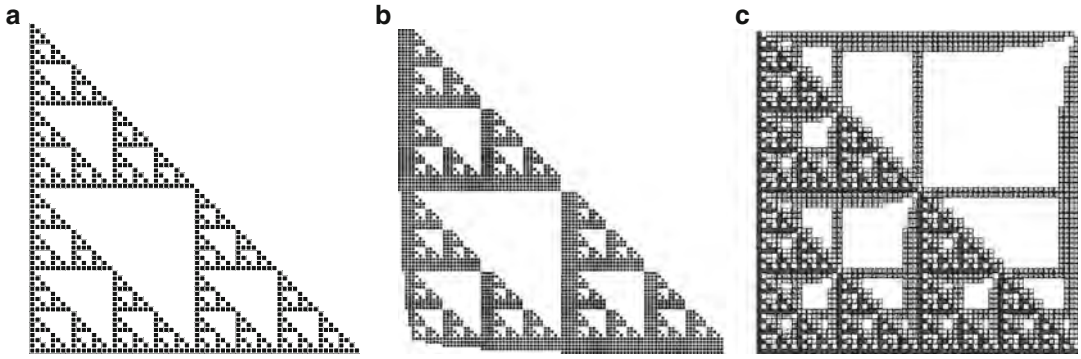
To date, the single positive result related to the strict self-assembly of a dssf is for an "active" model of self-assembly, where tiles are allowed to change the states of their glues during assembly, called the Signal-passing Tile Assembly Model (STAM). In [9] they gave a construction proving that the Sierpinski triangle can self-assemble within the STAM at temperature 1 and scale factor 2. By the result of [1], this is impossible in the aTAM and demonstrates the power of the active nature of the STAM, as that construction essentially builds stages of the Sierpinski triangle in a manner analogous to weak self-assembly, but then causes the unwanted interior portions to dissociate and then break apart.

Approximate Self-Assembly

It has been shown that an infinite subset of dssf's can weakly self-assemble in the aTAM, while another infinite subset cannot strictly self-assemble. Recall also that dssf's have fractal dimension <2, and since their strictly self-assembled versions retain their original fractal dimensions, so do they. However, their weakly self-assembled versions have dimension 2. Therefore, the question arises about whether or not some transformation of a dssf (especially, a dssf which cannot strictly self-assemble), which visually approximates the original dssf while retaining its fractal dimension, can strictly self-assemble in the aTAM.

This question was first answered positively in [6], where they defined a transformation for the Sierpinski triangle which they called "fibering," and they then gave a construction proving that the so-called fibered Sierpinski triangle does strictly self-assemble in the aTAM while maintaining the Sierpinski triangle's fractal dimension of ≈ 1.585. An example can be seen in Fig. 2b, showing how the fibering consists an additional row of tiles added to the south and west borders of each copy of each subsequent stage of the fractal. In [10] they extended the technique of fibering to include an infinite subclass of dssf's (which again includes the Sierpinski triangle) which they called "nice" dssf's. Nice dssf's are those whose generators are connected and contain all points on the west and south boundaries.

While the fibering technique creates visual approximations of fractals, it results in subsequent stages being further and further separated from each other. To counter this drawback, in [8] they introduced a technique for fibering the Sierpinski triangle "in place." An example can be seen in Fig. 2c, showing how this version of fibering only uses space on the interior of each stage of the fractal, thus allowing the stages to remain in the same positions relative to each other.



Self-Assembly of Fractals, Fig. 2 Various patterns corresponding to the Sierpinski triangle. (**a**) A portion of the discrete Sierpinski triangle. (**b**) A portion of the fibered Sierpinski triangle of [7] (Figure from [7]). (**c**) A portion of the in-place fibered Sierpinski triangle of [8] (Figure from [8])

Furthermore, this technique retains the same fractal dimension as the Sierpinski triangle, and they showed that it is impossible to use asymptotically less space than their construction while strictly self-assembling a shape which contains the Sierpinski triangle as a subset. In [5] this technique was extended to strictly self-assemble approximations for every generalized Sierpinski carpet.

Open Problems

1. Does there exist a discrete self-similar fractal which can strictly self-assemble in the aTAM, or conversely, can it be shown that none does?
2. What is the class of discrete self-similar fractals for which an approximation, such as fibering or in-place fibering, which maintains the original fractal dimension, strictly self-assembles in the aTAM?

URLs to Code and Data Sets

ISU TAS simulation software for the aTAM, kTAM, and 2HAM (http://self-assembly.net/wiki/index.php?title=ISU_TAS) and the Fibered Fractal Tiler for defining discrete self-similar fractals which can be fibered and generating the corresponding aTAM tile sets (http://self-assembly.net/wiki/index.php?title=Fibered_Fractal_Tiler).

Cross-References

- ▸ Experimental Implementation of Tile Assembly
- ▸ Self-Assembly at Temperature 1
- ▸ Self-Assembly of Squares and Scaled Shapes

Recommended Reading

1. Barth K, Furcy D, Summers SM, Totzke P (2014) Scaled tree fractals do not strictly self-assemble. In: Unconventional computation & natural computation (UCNC) 2014, University of Western Ontario, London, 14–18 July 2014 (to appear)
2. Cannon S, Demaine ED, Demaine ML, Eisenstat S, Patitz MJ, Schweller R, Summers SM, Winslow A (2012) Two hands are better than one (up to constant factors). Technical report 1201.1650, Computing Research Repository. http://arxiv.org/abs/1201.1650
3. Doty D, Gu X, Lutz JH, Mayordomo E, Moser P (2005) Zeta-dimension. In: Proceedings of the thirtieth international symposium on mathematical foundations of computer science, Gdansk. Springer, pp 283–294
4. Kautz SM, Lathrop JI (2009) Self-assembly of the Sierpinski carpet and related fractals. In: Proceedings of the fifteenth international meeting on DNA computing and molecular programming, Fayetteville, 8–11 June 2009, pp 78–87
5. Kautz S, Shutters B (2013) Self-assembling rulers for approximating generalized Sierpinski

carpets. Algorithmica 67(2):207–233. doi:10.1007/s00453-012-9691-x, http://dx.doi.org/10.1007/s00453-012-9691-x
6. Lathrop JI, Lutz JH, Summers SM (2007) Strict self-assembly of discrete Sierpinski triangles. In: Proceedings of the third conference on computability in Europe, Siena, 18–23 June 2007
7. Lathrop JI, Lutz JH, Summers SM (2009) Strict self-assembly of discrete Sierpinski triangles. Theor Comput Sci 410:384–405
8. Lutz JH, Shutters B (2012) Approximate self-assembly of the Sierpinski triangle. Theory Comput Syst 51(3):372–400
9. Padilla JE, Patitz MJ, Pena R, Schweller RT, Seeman NC, Sheline R, Summers SM, Zhong X (2013) Asynchronous signal passing for tile self-assembly: fuel efficient computation and efficient assembly of shapes. In: UCNC, Milan, pp 174–185
10. Patitz MJ, Summers SM (2008) Self-assembly of discrete self-similar fractals (extended abstract). In: Proceedings of the fourteenth international meeting on DNA computing, Prague, 2–6 June 2008 (to appear)
11. Rothemund PWK, Papadakis N, Winfree E (2004) Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol 2(12):e424. doi:10.1371/journal.pbio.0020424, http://dx.doi.org/10.1371
12. Soloveichik D, Winfree E (2007) Complexity of self-assembled shapes. SIAM J Comput 36(6):1544–1569

Self-Assembly of Squares and Scaled Shapes

Robert Schweller
Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, TX, USA

Keywords

Algorithmic self-assembly; Kolmogorov complexity; Scaled shapes; Self-assembly; Tile assembly model; Tile complexity

Years and Authors of Summarized Original Work

2000; Rothemund, Winfree
2001; Adleman, Cheng, Goel, Huang
2005; Cheng, Aggarwal, Goldwasser, Kao, Schweller, Espanes
2007; Soloveichik, Winfree

Problem Definition

Abstract Tile Assembly Model

The abstract Tile Assembly Model (aTAM) [3] is a mathematical model of self-assembly in which system components are four-sided Wang tiles with glue types assigned to each tile edge. Any pair of glue types are assigned some nonnegative interaction strength denoting how strongly the pair of glues bind. An aTAM system is an ordered triplet (T, τ, σ) consisting of a set of tiles T, a positive integer threshold parameter τ called the system's *temperature*, and a special tile $\sigma \in T$ denoted as the *seed* tile. Assembly proceeds by attaching copies of tiles from T to a growing seed assembly whenever the placement of a tile on the 2D grid achieves a total strength of attachment from abutting edges, determined by the sum of pairwise glue interactions, that meets or exceeds the temperature parameter τ. The pairwise strength assignment between glues on tile edges is often restricted to be "linear" in that identical glue pairs may be assigned arbitrary positive values, while non-equal pairs are required to have interaction strengths of 0. We denote this restricted version of the model as the *standard* aTAM. When this restriction is not applied, i.e., any pair of glues may be assigned any positive integer strength, we call the model the *flexible glue* aTAM.

Given the aTAM's model of growth, we may consider the problem of designing an aTAM system which is guaranteed to grow into a target shape S, given by a set of 2D integer coordinates, and stop growing. Such systems are guaranteed to exist for any finite shape S, but solutions will typically vary in the number of tiles $|T|$ used. For a given shape S, an interesting problem is to design a system that assembles S while using the fewest, or close to the fewest, number of tiles $|T|$ possible. This fewest possible number of tiles required for the assembly of a given shape S is termed the *program-size* complexity of S.

Problem 1 Let $K_{SA}(n)$ and $K_{SA}^{\sim}(n)$ denote the program-size complexity of an $n \times n$ square for the standard aTAM and the flexible glue aTAM, respectively. What are $K_{SA}(n)$ and $K_{SA}^{\sim}(n)$?

Problem 2 Let $K_{SA}(n,k)$ and $K_{SA}^{\sim}(n,k)$ denote the program-size complexity of a $k \times n$ rectangle for the standard aTAM and the flexible glue aTAM, respectively. What are $K_{SA}(n,k)$ and $K_{SA}^{\sim}(n,k)$?

Problem 3 For an arbitrary given shape S, what is the program-size complexity of S? Let the *scale-free* program size of S be the smallest tile set system that uniquely builds some scaled-up version S. Let $K_{SA}(S)$ and $K_{SA}^{\sim}(S)$ denote the *scale-free* program size of S for the standard aTAM and the flexible glue aTAM, respectively. What are $K_{SA}(S)$ and $K_{SA}^{\sim}(S)$?

Key Results

The best known bounds for program-size complexity for squares, rectangles, and general scaled shapes are presented in this section.

$n \times n$ Squares

The efficient self-assembly of $n \times n$ squares has served as a benchmark for self-assembly algorithms within the aTAM and more general tile assembly models. Within the aTAM, the problem is well understood up to constant factors. The first result states a general upper bound for the program size of self-assembled squares for general n, which is matched by an information-theoretic lower bound that holds for almost all integers n. The precise bounds differ between the standard and flexible glue models but are tight in both cases. The lower bound of inequality (1) is proven in [3] and is based on the Kolmogorov complexity of the integer n. The lower bound of (2) is proven in [2] by the same approach. The upper bound of (1) is proven in [1] and offers an improvement over the initial upper bound of $O(\log n)$ from [3]. The $O(\log n)$ result of [3] is achieved by implementing a key primitive in tile self-assembly: a binary counter of $\log n$ tiles that grows to length n. The improvement of [1] is achieved by modifying the counter concept to work with an optimal, variable base. The upper bound of (2) is proven in [2] and is obtained by combining the aTAM counter primitives with a scheme for efficiently seeding the counter by extracting bits from the values of the flexible glue interactions.

Theorem 1 *There exist positive constants c_1 and c_2 such that for almost all integers $n \in \mathbb{N}$, the following inequalities hold. Moreover, the upper bounds hold for all $n \in \mathbb{N}$.*

$$c_1 \frac{\log n}{\log \log n} \leq K_{SA}(n) \leq c_2 \frac{\log n}{\log \log n}. \quad (1)$$

$$c_1 \sqrt{\log n} \leq K_{SA}^{\sim}(n) \leq c_2 \sqrt{\log n}. \quad (2)$$

While the above theorem presents a tight understanding of the program-size complexity for most self-assembled squares, the information-theoretic lower bound allows for special values of n to be assembled with a much smaller program size. The program size is in fact as small as one could reasonably hope for. In [3], a tile system is presented that simulates a Busy Beaver Turing Machine and assembles correspondingly large squares for each tile set size. This construction yields the following theorem implying that the largest self-assembled square for a given number of tiles grows faster than any computable function!

Theorem 2 *There exists a positive constant c such that for infinitely many n, $K_{SA}(n) \leq c f(n)$ for $f(n)$ any nondecreasing unbounded computable function.*

Thin Rectangles

The program size of self-assembled squares and other thick rectangles is dictated by information-theoretic bounds which stem from the aTAM's ability to simulate arbitrary Turing machines given enough geometric space to work within. When this space is cut down, such as in the case of building a thin $k \times n$ rectangle, the program size is limited by geometric factors. The following upper and lower bounds are shown in [2] and represent the best known bounds for *thin* $k \times n$ rectangles in which $k = O(\log / \log \log n)$. The lower bound is achieved by a pigeon-hole pumping argument on the types of tiles placed, along with their order of placement, along a

width k column of the target rectangle. The upper bound is based on the construction of a general-base, general-width counter, which generalizes the binary counter concept of [3].

Theorem 3 *There exist positive constants c_1 and c_2 such that for any $n, k \in \mathbb{N}$, the following inequalities hold.*

$$c_1 \frac{n^{1/k}}{k} \leq K^{\sim}_{SA}(n,k) \leq K_{SA}(n,k) \leq c_2(n^{1/k}+k).$$

Scaled Shapes

The program size of general shapes is difficult to analyze as it is highly dependent on geometric features of the target shape. However, if we consider the assembly of an arbitrarily scaled-up version of a target shape, these geometric difficulties can be eliminated and a very general result can be achieved. The next result from [4] shows that the scale-free program size of S is closely related to the Kolmogorov complexity of S. In particular, the scale-free program-size complexity of S is a log factor less than the Kolmogorov complexity of S for the standard model, and the scale-free program size complexity of S is the square root of the Kolmogorov complexity of S for the flexible glue model. The standard model result is shown in [4] and is achieved by encoding a compressed description of S in a small tile set which is extracted by a set of tiles simulating a Turing machine that extracts the pixels of S from this compressed representation. The need for the scale factor increase of S is to allow room for the Turing machine simulation. In fact, the required scale factor is the run time of the Turing machine that decompresses the optimal encoding of S. The flexible glue result is achieved by combining portions of the flexible glue construction for squares [2] with the construction of [4]. In the following theorem, $K(S)$ denotes the Kolmogorov complexity of S with respect to some fixed universal Turing machine.

Theorem 4 *For any shape S, there exist positive constants c_1 and c_2 such that*

$$c_1 \frac{K(S)}{\log K(S)} \leq K_{SA}(S) \leq c_2 \frac{K(S)}{\log K(S)}. \quad (3)$$

$$c_1 \sqrt{K(S)} \leq K^{\sim}_{SA}(S) \leq c_2 \sqrt{K(S)}. \quad (4)$$

Open Problems

A few important open problems in this area are as follows. In the case of squares, the program size is well understood as long as the temperature of the system is at least two. A long-standing open problem has been to determine the program-size complexity of $n \times n$ squares for temperature-1 self-assembly in which each positive glue force alone is sufficient to cause a tile attachment. To date, no known method is able to achieve $o(n)$ tile complexity at temperature-1 for an $n \times n$ square, but no proof exists that this cannot be done. With respect to thin $k \times n$ rectangles, the best upper and lower bound have a gap with respect to variable k. Does there exist a more efficient rectangle construction, or can a higher lower bound be derived? Finally, while the scale-free program-size complexity of general shapes is well understood, little is known about the (unscaled) program size of general shapes. What new tools and geometric classifications can be developed to analyze and bound this complexity for general shapes?

Cross-References

- Active Self-Assembly and Molecular Robotics with Nubots
- Combinatorial Optimization and Verification in Self-Assembly
- Intrinsic Universality in Self-Assembly
- Patterned Self-Assembly Tile Set Synthesis
- Randomized Self-Assembly
- Robustness in Self-Assembly
- Self-Assembly at Temperature 1
- Self-Assembly of Fractals
- Self-Assembly of Squares and Scaled Shapes
- Self-Assembly with General Shaped Tiles
- Temperature Programming in Self-Assembly

Recommended Reading

1. Adleman L, Cheng Q, Goel A, Huang, M-D (2001) Running time and program size for self-assembled squares. In Proceedings of the thirty-third annual ACM symposium on theory of computing, New York. ACM, pp 740–748
2. Cheng Q, Aggarwal G, Goldwasser MH, Kao M-Y, Schweller RT, de Espanés PM (2005) Complexities for generalized models of self-assembly. SIAM J Comput 34:1493–1515
3. Rothemund PWK, Winfree E (2000) The program-size complexity of self-assembled squares (extended abstract). In Proceedings of the 32nd ACM symposium on theory of computing, STOC'00, Portland, pp 459–468
4. Soloveichik D, Winfree E (2007) Complexity of self-assembled shapes. SIAM J Comput 36(6):1544–1569

Self-Assembly with General Shaped Tiles

Andrew Winslow
Department of Computer Science, Tufts University, Medford, MA, USA

Keywords

Cellular automata; DNA computing; Geometric computing; Natural computing; Turing universality

Years and Authors of Summarized Original Work

2012; Fu, Patitz, Schweller, Sheline
2014; Demaine, Demaine, Fekete, Patitz, Schweller, Winslow, Woods

Problem Definition

Self-assembly is an asynchronous, decentralized process in which particles aggregate to form superstructures according to localized interactions. The most well-studied models of these particle systems, e.g., the abstract Tile Assembly Model of Winfree [11], utilize square-shaped particles arranged on a lattice by attaching edgewise. Particles attach to form larger *assemblies*, and a pair of assemblies or tiles can attach if they can translate to a nonoverlapping configuration with a set of k coincident edges, where $k \geq \tau$, a parameter of the system called the *temperature*.

In *seeded* assembly, individual particles attach to a growing *seed assembly*. This assembly may begin as a single-tile or a multi-tile assembly. In *unseeded* assembly (also called *hierarchical* [3], *two-handed* [2], or *polyomino* [7] assembly), there is no such restriction. The set of assemblies to which a single tile cannot attach (in seeded assembly) or that cannot attach to any other assembly (in unseeded assembly) are the *terminal assemblies* of the system.

Objectives In general, the goal is to design a system of minimal complexity that assembles into a unique terminal assembly with a desired shape or property. In models using square tiles, this is equivalent to designing a system using the fewest tile types. When tiles are allowed to be more general shapes, then the option of trading tile types for tile complexity becomes available. The motivation for this work is to understand how more complex tile shapes can be used to reduce the number of tile types in a system, and two benchmark problems regarding the computational power and efficiency of tile systems are considered in the context of systems of non-square tiles:

Problem 1 (Square Assembly)

INPUT*: A natural number n.*
OUTPUT*: A self-assembly system with a unique terminal assembly consisting of n^2 tiles in a $n \times n$ square shape.*

Problem 2 (Computational Power) *What systems of non-square tiles are capable of simulating computation, and to what extent?*

Key Results

In general, it is the case that allowing non-square tiles permits an asymptotic reduction in the number of tile types, and systems of very

few non-square tiles are capable of universal computation. At a high-level, such reductions are achieved by simulating many tiles via translations and rotations of a single tile type.

Models

Fu, Patitz, Schweller, and Sheline [6] introduce two models of general shaped types. The first, called the *geometric Tile Assembly Model (gTAM)*, is a model of seeded, translation-only assembly where tiles are polyomino-shaped – equivalent to prebuilt assemblies of square tiles. The second is an unseeded version they call the *Two-Handed Planar Geometric Tile Assembly Model (2GAM)*, which has the added restriction that assemblies can only attach if there exists a continuous motion bringing the two assemblies together during which they remain disjoint. This can be thought of as a restriction that the assemblies live in the plane and do not make use of the third dimension to maneuver into place.

Demaine et al. [4] introduce the *polygonal free-body Tile Assembly Model (pfbTAM)* in which tiles may have arbitrary simple polygonal shapes, attaching edgewise along equal-length edges. Systems without and without rotation are both permitted – we note that rotation is forbidden in the gTAM and 2GAM (as well as the aTAM).

Efficient Construction

Fu, Patitz, Schweller, and Sheline [6] prove that both the gTAM and 2GAM allow an asymptotic reduction in the number of tiles needed to assemble an $n \times n$ square of tiles. For the gTAM, they prove that such a square can be assembled using a temperature-1 system of $O(\sqrt{\log n})$ tile types, beating the optimal (temperature-2) system of $\Omega(\log n / \log \log n)$ square tiles by Adleman et al. [1]. This is a reduction in both the number of tile types (by a quadratic factor) and temperature. The temperature reduction is especially significant, as a lower bound of $\Omega(n)$ for temperature-1 aTAM systems is a widely believed conjecture [8–10].

For the 2GAM, they reduce the number of tile types even further, using a temperature-2 system $O(\log \log n)$ tile types to assemble an $n \times n$ square. However, this system comes with the caveat that system makes use of either a disconnected tile shape or a slightly three-dimensional shape.

Computational Power

Positive results on the computational power of general shaped tile systems fall into two categories: Turing universality and bounded-time computation. Fu, Patitz, Schweller, and Sheline [6] prove that any Turing machine computation can be carried out by a temperature-1 gTAM system. As with the temperature-1 construction of squares, this result is surprising due to the open conjecture regarding the computational power of square tile systems at temperature 1.

Demaine et al. [4] prove that any Turing machine computation can be carried out by a temperature-2 pfbTAM system (with rotation) consisting of a single tile. Their result actually proves that any aTAM system can be simulated by such a system, and thus Turing universality is achieved by simulating aTAM systems carrying out computation. Combined with the intrinsic universality result of Doty et al. [5], this result can be extended to prove that a *single* temperature-2 pfbTAM system (with rotation) consisting of a single tile can carry out any Turing machine computation, given an appropriate seed assembly consisting of copies of this tile.

Finally, Demaine et al. also prove that temperature-3 pfbTAM systems (*without* rotation) consisting of a single tile can carry out simulation of computationally universal cellular automata for a number of steps limited by the size of the seed assembly. Specifically, they prove that n steps can be carried out using a seed assembly of $O(n)$ tiles. A loose lower bound is also proved, namely, that more than three tiles are needed to carry out any computation.

Applications

The generic ability to reduce the number of tile types in a system by increasing the geometric complexity of these tiles extends many other

constructions in theoretical tile assembly. Additionally, there may be practical barriers to systems of many tile types, e.g., additional cost of manufacturing or longer assembly time due to heterogenous combinations of many particle types, that can be reduced or eliminated by replacing these systems with systems of fewer, more complex tile.

Open Problems

Obtaining an upper bound on the number of steps of a cellular automaton simulable by single-tile translation-only systems remains open. It is conjectured that a seed assembly of size n can only carry out $O(n^2)$ steps.

Cross-References

- Combinatorial Optimization and Verification in Self-Assembly
- Experimental Implementation of Tile Assembly
- Self-Assembly at Temperature 1

Recommended Reading

1. Adleman L, Cheng Q, Goel A, Huang MD (2001) Running time and program size for self-assembled squares. In: Proceedings of symposium on theory of computing (STOC), Heraklion
2. Cannon S, Demaine ED, Demaine ML, Eisenstat S, Patitz MJ, Schweller RT, Summers SM, Winslow A (2013) Two hands are better than one (up to constant factors): self-assembly in the 2HAM vs. aTAM. In: STACS 2013, Kiel, LIPIcs, vol 20. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp 172–184
3. Chen H, Doty D (2012) Parallelism and time in hierarchical self-assembly. In: Proceedings of the 23rd annual ACM-SIAM symposium on discrete algorithms (SODA), Kyoto, pp 1163–1182
4. Demaine ED, Demaine ML, Fekete SP, Patitz MJ, Schweller RT, Winslow A, Woods D (2014) One tile to rule them all: simulating any tile assembly system with a single universal tile. In: Esparza J, Fraigniaud P, Husfeldt T, Koutsoupias E (eds) Automata, languages and programming (ICALP), Copenhagen. LNCS, vol 8572. Springer, Berlin/Heidelberg, pp 368–379
5. Doty D, Lutz JH, Patitz MJ, Schweller RT, Summers SM, Woods D (2012) The tile assembly model is intrinsically universal. In: Proceedings of the 53rd annual symposium on foundations of computer science (FOCS), New Brunswick, pp 302–310
6. Fu B, Patitz MJ, Schweller RT, Sheline B (2012) Self-assembly with geometric tiles. In: Czumaj A, Mehlhorn K, Pitts A, Wattenhofer R (eds) Automata, languages and programming (ICALP), Warwick. LNCS, vol 7391. Springer, Berlin/New York, pp 714–725
7. Luhrs C (2010) Polyomino-safe DNA self-assembly via block replacement. Nat Comput 9(1):97–109
8. Meunier PE, Patitz MJ, Summers SM, Theyssier G, Winslow A, Woods D (2014) Intrinsic universality in tile self-assembly requires cooperation. In: Proceedings of the 25th annual ACM-SIAM symposium on discrete algorithms (SODA), Portland, pp 752–771
9. Rothemund PWK, Winfree E (2000) The program-size complexity of self-assembled squares (extended abstract). In: Proceedings of ACM symposium on theory of computing (STOC), Portland, pp 459–468
10. Summers SM (2010) Universality in algorithmic self-assembly. PhD thesis, Iowa State University
11. Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, Caltech

Selfish Bin Packing Problems

Leah Epstein
Department of Mathematics, University of Haifa, Haifa, Israel

Keywords

Bin packing; Price of anarchy; Selfish agents

Years and Authors of Summarized Original Work

2006; Bilò
2008, 2011; Epstein, Kleiman

Problem Definition

In bin packing games with selfish items, n items are to be packed into (at most) n bins, where each item chooses a bin that it wishes to be packed

into. The cost of an item i of size $0 < s_i \leq 1$ is defined based on its size and the contents of its bin. Nash equilibria (NE) are defined as solutions where there is no item that can change its choice unilaterally and gain from this change. Bin packing games were inspired by the well-known bin packing problem [2]. In this problem, a set of items, each of size in $(0, 1]$, is given. The goal is to partition (or pack) the items into a minimum number of subsets that are called *bins*. Each bin has unit capacity, and the load of a bin is defined to be the total size of items packed into it (where the load cannot exceed 1). The problem is NP-hard in the strong sense, and thus theoretical research has focused on studying and developing approximation algorithms, which allow to design nearly optimal solutions, and on online algorithms, which receive the items one by one and must assign each item to a bin immediately and irrevocably (without any information on further items).

In a bin packing game, every item is operated by a selfish player. There are n bins, and the strategy of a player is the bin that it selects. If the resulting packing is valid (i.e., the load of no bin exceeds 1), then the set of items sharing a bin share its cost proportionally, i.e., let B be a bin (a subset of items). The cost of $i \in B$ is $s_i / \left(\sum_{j \in B} s_j \right)$. If the resulting packing is invalid, any item packed into an invalid bin has infinite cost. We are interested in pure Nash equilibria, and by the term NE, we refer to such an equilibrium. The problem was presented by Bilò [1].

There are several directions which can be explored. First, one would like to find out if any bin packing game has an NE. If this is the case, other kinds of equilibria might be of interest as well. For a class of games (such that each of them has an NE), a process of convergence is defined as follows. The process starts with an arbitrary configuration, and at each time, an item that can reduce its cost is selected and moved to another bin (where the cost of this item will be smaller than its cost before it is moved). Such a process can also be seen as local search. Items are moved one at a time; a single move (for one item) is called a step. Note that one item can participate in multiple steps. The questions which can be asked are whether the process converges for any initial packing (i.e., reaches a state that no further step can be applied) and how large can the number of steps be. As it turns out, any bin packing game has at least one NE, and the processes described here always converge [1, 8]. Since it is possible that the process converges in exponential time, it is of interest to develop a polynomial time algorithm that computes NE packings. Such an algorithm for this problem defined above was designed by Yu and Zhang [9]. Finally, once the existence of NE packing has been established, the primary goal becomes the study of the quality of worst-case equilibria. This concept is called *price of anarchy*. For a given game G (i.e., a set of items which is an input for bin packing), the price of anarchy of this game, denoted by $POA(G)$, is the ratio between the maximum number of nonempty bins in any NE packing and the minimum number of bins in any packing (the number of bins in an optimal packing, also called the social optimum, denoted by $OPT(G)$). The price of stability is similar, but best-case equilibria are studied, and as Bilò [1] proved that any game has a social optimum that is an NE, the price of stability is 1 for any game.

The price of anarchy (POA) of a class of games (here, the class of all bin packing games) is defined to be the supremum POA over all games in the class. However, as bin packing is typically studied with respect to the asymptotic approximation ratio, the POA for the bin packing class of games is defined, similarly to the asymptotic approximation ratio, as $\lim_{M \to \infty} \sup_{\{G : OPT(G) \geq M\}} POA(G)$.

Key Results

The POA was studied already in [1], where Bilò provided the first bounds on it, a lower bound of $\frac{8}{5}$ and an upper bound of $\frac{5}{3}$. The quality of NE solutions was further investigated in [4], where nearly tight bounds for the PoA were given, an

upper bound of 1.6428 and a lower bound of 1.6416 (see also [9]). The parametric POA, which is the POA for subclasses of games where the size of no item exceeds a given value, was considered as well [5].

NE packings are related to outputs of the algorithm First Fit (FF) for bin packing [7]. FF is in fact an online algorithm that packs each item, in turn, into a minimum index bin where it fits (using an empty bin if there is no other option). It is not difficult to see that every NE is an output of FF; sort the bins of the NE by non-increasing loads, and create a list of items according to the ordering of bins. FF will create exactly the bins of the original packing. Interestingly, the POA is significantly smaller than the asymptotic approximation ratio of FF (which is equal to 1.7 [7]). Note that the PoA is not equal to the approximation ratio of any natural algorithm for bin packing.

Some intuition regarding the difference between the asymptotic approximation ratio of First Fit and the POA of this class of games can be shown using a small example. Consider items of the following sizes (for a sufficiently small $\varepsilon > 0$): $\frac{1}{6} - 2\varepsilon$ (small items), $\frac{1}{3} + \varepsilon$ (medium items), and $\frac{1}{2} + \varepsilon$ (large items). The worst-case examples for FF are similar to this example, though the items of the first two types have a number of different sizes; small items can be slightly smaller or slightly larger than $\frac{1}{6}$, and medium items can be slightly smaller or slightly larger than $\frac{1}{3}$. Given the item types defined above, assume that there are $6N$ items of each type (for some positive integer N), when FF receives this input (sorted by non-decreasing size), it creates N bins with six small items packed into each bin, $3N$ bins with two medium items packed into each bin, and the remaining items are packed into dedicated bins. This packing is not an NE, as a medium item reduces its cost from $\frac{1}{2}$ to approximately $\frac{2}{5}$ if it joins a large item. Indeed, roughly speaking, if an NE packing consists of a large number of bins (compared to an optimal solution), a bin of this NE packing either has an item whose size exceeds $\frac{1}{2}$ or its load cannot be as small as approximately $\frac{2}{3}$. This allows a tighter analysis. Interestingly, in worst-case examples for the POA, medium items have sizes that are close to $\frac{1}{4}$ instead of $\frac{1}{3}$.

Related Results

Bin packing games, where the cost of an item is defined differently, were studied. One option is to assign equal costs to all players (which are packed together into a valid bin) [3, 6]. A generalized version where each item has a positive weight, and costs are based on cost sharing proportional to the weights of items that share a bin [3] was studied as well. The weights of items in the games described above (those of [1, 4]) are equal to their sizes. These are two classes of games, for which the POA turns out to be of interest. The POA for the class of games with equal weights is slightly (strictly) below 1.7, and in the case of general weights, the POA is equal to 1.7 [3]. Another topic of interest is the quality of other kinds of equilibria. Those are strong equilibria, which are solutions that are also resilient to deviations of subsets of items reducing their costs, and Pareto optimal equilibria, where the solution is required to be weakly (or strictly) Pareto optimal, that is, there is no alternative packing where all items reduce their costs (or a packing where no item increases its cost and at least one item reduces it) [3]. For these last kinds of equilibria, the POA is still above 1.6 (but at most 1.7).

Cross-References

▸ Subset Sum Algorithm for Bin Packing

Recommended Reading

1. Bilò V (2006) On the packing of selfish items. In: Proceedings of the 20th international parallel and distributed processing symposium (IPDPS2006). IEEE, Rhodes, Greece, 9pp
2. Coffman E Jr, Csirik J (2007) Performance guarantees for one-dimensional bin packing. In: Gonzalez TF (ed) Handbook of approximation algorithms and metaheuristics, chap 32. Chapman & Hall/CRC, Boca Raton, pp (32–1)–(32–18)

3. Dósa Gy, Epstein L (2012) Generalized selfish bin packing. CoRR, abs/1202.4080
4. Epstein L, Kleiman E (2011) Selfish bin packing. Algorithmica 60(2):368–394
5. Epstein L, Kleiman E, Mestre J (2009) Parametric packing of selfish items and the subset sum algorithm. In: Proceedings of the 5th international workshop on internet and network economics (WINE2009), Rome, Italy, pp 67–78
6. Han X, Dósa Gy, Ting HF, Ye D, Zhang Y (2013) A note on a selfish bin packing problem. J Glob Optim 56(4):1457–1462
7. Johnson DS, Demers AJ, Ullman JD, Garey MR, Graham RL (1974) Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM J Comput 3(4):299–325
8. Miyazawa FK, Vignatti AL (2009) Convergence time to Nash equilibrium in selfish bin packing. Electron Notes Discret Math 35:151–156
9. Yu G, Zhang G (2008) Bin packing of selfish items. In: The 4th international workshop on internet and network economics (WINE2008), Shanghai, China, pp 446–453

Selfish Unsplittable Flows: Algorithms for Pure Equilibria

Paul (Pavlos) Spirakis
Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece
Computer Science, University of Liverpool, Liverpool, UK
Computer Technology Institute (CTI), Patras, Greece

Keywords

Atomic network congestion games; Cost of anarchy

Years and Authors of Summarized Original Work

2005; Fotakis, Kontogiannis, Spirakis

Problem Definition

Consider having a set of resources E in a system. For each $e \in E$, let $d_e(\cdot)$ be the delay per user that requests its service, as a function of the total usage of this resource by all the users. Each such function is considered to be non−decreasing in the total usage of the corresponding resource. Each resource may be represented by a pair of points: an entry point to the resource and an exit point from it. So, each resource is represented by an arc from its entry point to its exit point and the model associates with this arc the cost (e.g., the delay as a function of the load of this resource) that each user has to pay if she is served by this resource. The entry/exit points of the resources need not be unique; they may coincide in order to express the possibility of offering joint service to users, that consists of a sequence of resources. Here, denote by V the set of all entry/exit points of the resources in the system. Any nonempty collection of resources corresponding to a directed path in $G \equiv (V, E)$ comprises an *action* in the system.

Let $N \equiv [n]$ be a set of users, each willing to adopt some action in the system. $\forall i \in N$, let w_i denote user i's *demand* (e.g., the flow rate from a source node to a destination node), while $\Pi_i \subseteq 2^E \setminus \emptyset$ is the collection of actions, any of which would satisfy user i (e.g., alternative routes from a source to a destination node, if G represents a communication network). The collection Π_i is called the *action set* of user i and each of its elements contains at least one resource. Any vector $\boldsymbol{r} = (r_1, \ldots, r_n) \in \Pi \equiv \times_{i=1}^{n} \Pi_i$ is a *pure strategies profile*, or a *configuration* of the users. Any vector of real functions $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)$ s.t. $\forall i \in [n]$, $p_i : \Pi_i \to [0, 1]$ is a probability distribution over the set of allowable actions for user i (i.e., $\sum_{r_i \in \Pi_i} p_i(r_i) = 1$), and is called a *mixed strategies profile* for the n users.

A congestion model typically deals with users of identical demands, and thus, user cost function depending on the *number* of users adopting each action [1, 4, 6]. In this work the more general case is considered, where a *weighted congestion model* is the tuple $((w_i)_{i \in N}, (\Pi_i)_{i \in N}, (d_e)_{e \in E})$. That is, the users are allowed to have different demands for service from the whole system, and thus affect the resource delay functions in a different way, depending on their own weights.

A *weighted congestion game* associated with this model, is a game in strategic form with the set of users N and user demands $(w_i)_{i\in N}$, the action sets $(\Pi_i)_{i\in N}$ and cost functions $(\lambda^i_{r_i})_{i\in N, r_i\in\Pi_i}$ defined as follows: For any configuration $\boldsymbol{r}\in\Pi$ and $\forall e\in E$, let $\Lambda_e(\boldsymbol{r})=\{i\in N : e\in r_i\}$ be the set of users exploiting resource e according to $\boldsymbol{r}$ (called the *view* of resource e wrt configuration $\boldsymbol{r}$). The *cost* $\lambda^i(\boldsymbol{r})$ *of user* i *for adopting strategy* $r_i\in\Pi_i$ in a given configuration $\boldsymbol{r}$ is equal to the cumulative *delay* $\lambda_{r_i}(\boldsymbol{r})$ along this path:

$$\lambda^i(\boldsymbol{r})=\lambda_{r_i}(\boldsymbol{r})=\sum_{e\in r_i} d_e(\theta_e(\boldsymbol{r})) \qquad (1)$$

where, $\forall e\in E, \theta_e(\boldsymbol{r})\equiv\sum_{i\in\Lambda_e(\boldsymbol{r})} w_i$ is the load on resource e wrt the configuration $\boldsymbol{r}$.

On the other hand, for a mixed strategies profile $\boldsymbol{p}$, the *expected cost of user* i *for adopting strategy* $r_i\in\Pi_i$ is

$$\lambda^i_{r_i}(\boldsymbol{p})=\sum_{\boldsymbol{r^{-i}}\in\Pi^{-i}} P(\boldsymbol{p^{-i}},\boldsymbol{r^{-i}})\cdot \sum_{e\in r_i} d_e\left(\theta_e(\boldsymbol{r^{-i}}\oplus r_i)\right) \qquad (2)$$

where, $\boldsymbol{r^{-i}}$ is a configuration of all the users except for user i, $\boldsymbol{p^{-i}}$ is the mixed strategies profile of all users except for i, $\boldsymbol{r^{-i}}\oplus r_i$ is the new configuration with user i choosing strategy r_i, and $P(\boldsymbol{p^{-i}},\boldsymbol{r^{-i}})\equiv\prod_{j\in N\setminus\{i\}} p_j(r_j)$ is the occurrence probability of $\boldsymbol{r^{-i}}$.

Remark 1 Here notation is abused a little bit and the model considers the user costs $\lambda^i_{r_i}$ as functions whose exact definition depends on the other users' strategies: In the general case of a mixed strategies profile $\boldsymbol{p}$, (2) is valid and expresses the expected cost of user i wrt $\boldsymbol{p}$, conditioned on the event that i chooses path r_i. If the other users adopt a pure strategies profile $\boldsymbol{r^{-i}}$, we get the special form of (1) that expresses the exact cost of user i choosing action r_i.

A congestion game in which all users are indistinguishable (i.e., they have the same user cost functions) and have the same action set, is called *symmetric*. When each user's action set Π_i consists of sets of resources that comprise (simple) paths between a unique origin-destination pair of nodes (s_i,t_i) in a network $G=(V,E)$, the model refers to a *network congestion game*. If additionally all origin-destination pairs of the users coincide with a unique pair (s,t) one gets a *single commodity network congestion game* and then all users share exactly the same action set. Observe that a single-commodity network congestion game is not necessarily symmetric because the users may have different demands and thus their cost functions will also differ.

Selfish Behavior

Fix an arbitrary (mixed in general) strategies profile $\boldsymbol{p}$ for a congestion game $\left((w_i)_{i\in N},(\Pi_i)_{i\in N},(d_e)_{e\in E}\right)$. We say that $\boldsymbol{p}$ is a *Nash Equilibrium (NE)* if and only if $\forall i\in N, \forall r_i,\pi_i\in\Pi_i, p_i(r_i)>0\Rightarrow\lambda^i_{r_i}(\boldsymbol{p})\le\lambda^i_{\pi_i}(\boldsymbol{p})$. A configuration $\boldsymbol{r}\in\Pi$ is a *Pure Nash Equilibrium (PNE)* if and only if $(\forall i\in N,\forall\pi_i\in\Pi_i,\lambda_{r_i}(\boldsymbol{r})\le\lambda_{\pi_i}(\boldsymbol{r^{-i}}\oplus\pi_i)$ where, $\boldsymbol{r}^{-i}\oplus\pi_i$ is the same configuration with $\boldsymbol{r}$ except for user i that now chooses action π_i.

Key Results

In this section the article deals with the existence and tractability of PNE in weighted network congestion games. First, it is shown that it is not always the case that a PNE exists, even for a weighted single-commodity network congestion game with only linear and 2-wise linear (e.g., the maximum of two linear functions) resource delays. In contrast, it is well known [1, 6] that any unweighted (not necessarily single-commodity, or even network) congestion game has a PNE, for any kind of nondecreasing delays. It should be mentioned that the same result has been independently proved also by [3].

Lemma 1 *There exist instances of weighted single–commodity network congestion games with resource delays being either linear or 2–wise linear functions of the loads, for which there is no PNE.*

Theorem 2 *For any weighted multi–commodity network congestion game with linear resource delays, at least one PNE exists and can be computed in pseudo-polynomial time.*

Proof Fix an arbitrary network $G = (V, E)$ with linear resource/edge delays $d_e(x) = a_e x + b_e$, $e \in E$, $a_e, b_e \geq 0$. Let $\boldsymbol{r} \in \Pi$ be an arbitrary configuration for the corresponding weighted multi–commodity congestion game on G. For the configuration r consider the potential $\Phi(\boldsymbol{r}) = C(\boldsymbol{r}) + W(\boldsymbol{r})$, where

$$C(\boldsymbol{r}) = \sum_{e \in E} d_e(\theta_e(\boldsymbol{r}))\theta_e(\boldsymbol{r}) = \sum_{e \in E} [a_e \theta_e^2(\boldsymbol{r}) + b_e \theta_e(\boldsymbol{r})],$$

and

$$W(\boldsymbol{r}) = \sum_{i=1}^{n} \sum_{e \in r_i} d_e(w_i) w_i = \sum_{e \in E} \sum_{i \in \widetilde{e}(\boldsymbol{r})} d_e(w_i) w_i = \sum_{e \in E} \sum_{i \in \widetilde{e}(\boldsymbol{r})} (a_e w_i^2 + b_e w_i)$$

one concludes that

$$\Phi(\boldsymbol{r}') - \Phi(\boldsymbol{r}) = 2w_i [\lambda^i(\boldsymbol{r}') - \lambda^i(\boldsymbol{r})] ,$$

Note that the potential is a global system function whose changes are proportional to selfish cost improvements of any user. The global minima of the potential then correspond to configurations in which no user can improve her cost acting unilaterally. Therefore, any weighted multi–commodity network congestion game with linear resource delays admits a PNE. □

Applications

In [5] many experiments have been conducted for several classes of pragmatic networks. The experiments show even faster convergence to pure Nash Equilibria.

Open Problems

The Potential function reported here is polynomial on the loads of the users. It is open whether one can find a purely combinatorial potential, which will allow strong polynomial time for finding Pure Nash equilibria.

Cross-References

- Best Response Algorithms for Selfish Routing
- Computing Pure Equilibria in the Game of Parallel Links
- General Equilibrium

Recommended Reading

1. Fabrikant A, Papadimitriou C, Talwar K (2004) The complexity of pure nash equilibria. In: Proceedings of the 36th ACM symposium on theory of computing (STOC'04). ACM, Chicago
2. Fotakis D, Kontogiannis S, Spirakis P (2005) Selfish unsplittable flows. J Theory Comput Sci 348:226–239
3. Libman L, Orda A (2001) Atomic resource sharing in noncooperative networks. Telecommun Syst 17(4):385–409
4. Monderer D, Shapley L (1996) Potential games. Game Econ Behav 14:124–143
5. Panagopoulou P, Spirakis P (2006) Algorithms for pure nash equilibrium in weighted congestion games. ACM J Exp Algorithms 11:2.7
6. Rosenthal RW (1973) A class of games possessing pure-strategy nash equilibria. Int J Game Theory 2:65–67

Self-Stabilization

Ted Herman
Department of Computer Science, University of Iowa, Iowa City, IA, USA

Keywords

Autonomic system control; Autopoesis; Homeostasis

Years and Authors of Summarized Original Work

1974; Dijkstra

Problem Definition

An algorithm is self-stabilizing if it eventually manifests correct behavior regardless of initial state. The general problem is to devise self-stabilizing solutions for a specified task. The property of self-stabilization is now known to be feasible for a variety of tasks in distributed computing. Self-stabilization is important for distributed systems and network protocols subject to transient faults. Self-stabilizing systems automatically recover from faults that corrupt state.

The operational interpretation of self-stabilization is depicted in Fig. 1. Part (a) of the figure is an informal presentation of the behavior of a self-stabilizing system, with time on the x-axis and some informal measure of correctness on the y-axis. The curve illustrates a system trajectory, through a sequence of states, during execution. At the initial state, the system state is incorrect; later, the system enters a correct state, then returns to an incorrect state, and subsequently stabilizes to an indefinite period where all states are correct. This period of stability is disrupted by a transient fault that moves the system to an incorrect state, after which the scenario above repeats. Part (b) of the figure illustrates the scenario in terms of state predicates. The box represents the predicate *true*, which characterizes all possible states. Predicate $\mathcal{C}$ characterizes the correct states of the system, and $\mathcal{L} \subset \mathcal{C}$ depicts the closed *legitimacy* predicate. Reaching a state in $\mathcal{L}$ corresponds to entering a period of stability in part (a). Given an algorithm A with this type of behavior, it is said that A self-stabilizes to $\mathcal{L}$; when $\mathcal{L}$ is implicitly understood, the statement is simplified to: A is self-stabilizing.

Problem [3]. The first setting for self-stabilization posed by Dijkstra is a ring of n processes numbered 0 through $n-1$. Let the state of process i be denoted by $x[i]$. Communication is unidirectional in the ring using a *shared state* model. An atomic step of process i can be expressed by a guarded assignment of the form $g(x[i \ominus 1], x[i]) \rightarrow x[i] := f(x[i \ominus 1], x[i])$. Here, $\ominus$ is subtraction modulo n, so that $x[i \ominus 1]$ is the state of the previous process in the ring with respect to process i. The guard g is a boolean expression; if $g(x[i \ominus 1], x[i])$ is *true*, then process i is said to be *privileged* (or enabled). Thus in one atomic step, privileged process i reads the state of the previous process and computes a new state. Execution scheduling is controlled by a *central daemon*, which fairly chooses one among all enabled processes to take the next step. The problem is to devise g and f so that, regardless of initial states of $x[i]$, $0 \leq i < n$, eventually there is one privilege and every process enjoys a privilege infinitely often.

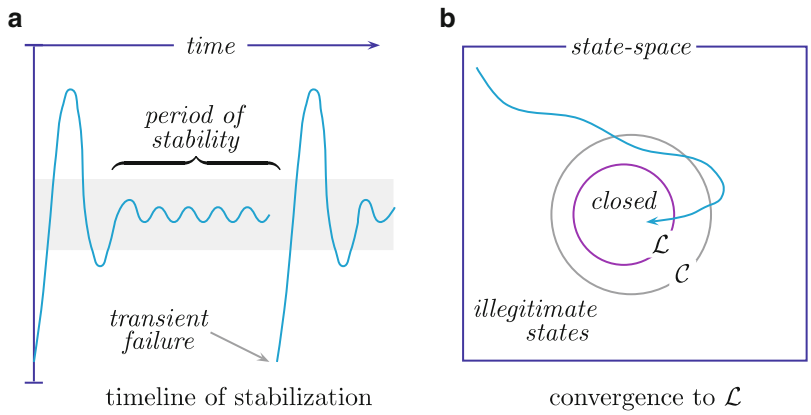
Complexity Metrics

The complexity of self-stabilization is evaluated by measuring the resource needed for convergence from an arbitrary initial state. Most prominent in the literature of self-stabilization are metrics for worst-case time of convergence and space required by an algorithm solving the given task. Additionally, for reactive self-stabilizing algorithms, metrics are evaluated for the stable behavior of the algorithm, that is, starting from a legitimate state, and compared to non-stabilizing algorithms, to measure costs of self-stabilization.

Key Results

Composition

Many self-stabilizing protocols have a layered construction. Let $\{ A_i \}_{i=0}^{m-1}$ be a set programs with the property that for every state variable x, if program A_i writes x, then no program A_j, for $j > i$, writes x. Programs in $\{ A_j \}_{j=i+1}^{m-1}$ may read variables written by A_i, that is, they use the output of A_i as input. Fair composition of programs B and C, written $B \,[]\, C$, assumes fair scheduling of steps of B and C. Let X_j be the set



Self-Stabilization, Fig. 1 Self-stabilization trajectories

of variables read by A_j and possibly written by $\{ A_i \}_{i=0}^{j-1}$.

Theorem 1 (Fair Composition [4]) *Suppose A_i is self-stabilizing to $\mathcal{L}_i$ under the assumption that all variables in X_i remain constant throughout any execution; then $A_0 [] A_1 [] \cdots [] A_{m-1}$ self-stabilizes to $\{ \mathcal{L}_i \}_{i=0}^{m-1}$.*

Fair composition with a layered set $\{ A_i \}_{i=0}^{m-1}$ corresponds to sequential composition of phases in a distributed algorithm. For instance, let B be a self-stabilizing algorithm for mutual exclusion in a network that assumes the existence of a rooted, spanning tree and let algorithm C be a self-stabilizing algorithm to construct a rooted spanning tree in a connected network; then $B [] C$ is a self-stabilizing mutual exclusion algorithm for a connected network.

Synchronization Tasks

One question related to the problem posed in section "Problem Definition" is whether or not there can be a *uniform* solution, where all processes have identical algorithms. Dijkstra's result for the unidirectional ring is a semi-uniform solution (all but one process have the same algorithm), using n states per process. The state of each process is a counter: process 0 increments the counter modulo k, where $k \geq n$ suffices for convergence; the other processes copy the counter of the preceding process in the ring. At a legitimate state, each time process 0 increments the counter, the resulting value is different from all other counters in the ring. This ring algorithm turns out to be self-stabilizing for the *distributed daemon* (any subset of privileged processes may execute in parallel) when $k > n$. Subsequent results have established that mutual exclusion on a unidirection ring is $\Theta(1)$ space per process with a non-uniform solution. Deterministic uniform solutions to this task are generally impossible, with the exceptional case where n is and prime. Randomized uniform solutions are known for arbitrary n, using $O(\lg \alpha)$ space where α is the smallest number that does not divide n. Some lower bounds on space for uniform solutions are derived in [7]. Time complexity of Dijkstra's algorithm is $O(n^2)$ rounds, and some randomized solutions have been shown to have expected $O(n^2)$ convergence time.

Dijkstra also presented a solution to mutual exclusion for a linear array of processes, using $O(1)$ space per process [3]. This result was later generalized to a rooted tree of processes, but with mutual exclusion relaxed to having one privilege along any path from root to leaf. Subsequent research built on this theme, showing how tasks for distributed wave computations have self-stabilizing solutions. Tasks of phase synchronization and clock synchronization have also been solved. See reference [9] for an example of self-stabilizing mutual exclusion in a multiprocessor shared memory model.

Graph Algorithms

Communication networks are commonly represented with graph models and the need for distributed graph algorithms that tolerate

transient faults motivates study of such tasks. Specific results in this area include self-stabilizing algorithms for spanning trees, center-finding, matching, planarity testing, coloring, finding independent sets, and so forth. Generally, all graph tasks can be solved by self-stabilizing algorithms: tasks that have network topology and possibly related factors, such as edge weights, for input, and define outputs to be a function of the inputs, can be solved by general methods for self-stabilization. These general methods require considerable space and time resource, and may also use stronger model assumptions than needed for specific tasks, for instance unique process identifiers and an assumed bound on network diameter. Therefore research continues on graph algorithms.

One discovery emerging from research on self-stabilizing graph algorithms is the difference between algorithms that terminate and those that continuously change state, even after outputs are stable. Consider the task of constructing a spanning tree rooted at process r. Some algorithms self-stabilize to the property that, for every $p \neq r$, the variable u_p refers to p's parent in the spanning tree and the state remains unchanged. Other algorithms are self-stabilizing protocols for token circulation with the side-effect that the circulation route of the token establishes a spanning tree. The former type of algorithm has $O(\lg n)$ space per process, whereas the latter has $O(\lg \delta)$ where δ is the degree (number of neighbors) of a process. This difference was formalized in the notion of *silent* algorithms, which eventually stop changing any communication value; it was shown in [5] for the link register model that silent algorithms for many graph tasks have $\Omega(\lg n)$ space.

Transformation

The simple presentation of [3] is enabled by the abstract computation model, which hides details of communication, program control, and atomicity. Self-stabilization becomes more complicated when considering conventional architectures that have messages, buffers, and program counters. A natural question is how to transform or refine self-stabilizing algorithms expressed in abstract models to concrete models closer to practice. As an example, consider the problem of transforming algorithms written for the central daemon to the distributed daemon model. This transformation can be reduced to finding a self-stabilizing token-passing algorithm for the distributed daemon model such that, eventually, no two neighboring processes concurrently have a token; multiple tokens can increase the efficiency of the transformation.

General Methods

The general problem of constructing a self-stabilizing algorithm for an input nonreactive task can be solved using standard tools of distributed computing: snapshot, broadcast, system reset, and synchronization tasks are building blocks so that the global state can be continuously validated (in some fortunate cases $\mathcal{L}$ can be locally checked and corrected). These building blocks have self-stabilizing solutions, enabling the general approach.

Fault Tolerance

The connection between self-stabilization and transient faults is implicit in the definition. Self-stabilization is also applicable in executions that asynchronously change inputs, silently crash and restart, and perturb communication [10]. One objection to the mechanism of self-stabilization, particularly when general methods are applied, is that a small transient fault can lead to a system-wide correction. This problem has been investigated, for example in [8], where it is shown how convergence can be optimized for a limited number of faults. Self-stabilization has also been combined with other types of failure tolerance, though this is not always possible: the task of counting the number of processes in a ring has no self-stabilizing solution in the shared state model if a process may crash [1], unless a failure detector is provided.

Applications

Many network protocols are self-stabilizing by the following simple strategy: periodically,

they discard current data and regenerate it from trusted information sources. This idea does not work in purely asynchronous systems; the availability of real-time clocks enables the simple strategy. Similarly, watchdogs with hardware clocks can provide an effective basis for self-stabilization [6].

Cross-References

▸ Concurrent Programming, Mutual Exclusion

Recommended Reading

1. Anagnostou E, Hadzilacos V (1993) Tolerating transient and permanent failures. In: Distributed algorithms 7th international workshop. LNCS, vol 725. Springer, Heidelberg, pp 174–188
2. Cournier A, Datta AK, Petit F, Villain V (2002) Snap-stabilizing PIF algorithm in arbitrary networks. In: Proceedings of the 22nd international conference distributed computing systems, Vienna, July 2002, pp 199–206
3. Dijkstra EW (1974) Self stabilizing systems in spite of distributed control. Commun ACM 17(11):643–644. See also EWD391 (1973) In: Selected writings on computing: a personal perspective. Springer, New York, pp 41–46 (1982)
4. Dolev S (2000) Self-stabilization. MIT, Cambridge
5. Dolev S, Gouda MG, Schneider M (1996) Memory requirements for silent stabilization. In: Proceedings of the 15th annual ACM symposium on principles of distributed computing, Philadelphia, May 1996, pp 27–34
6. Dolev S, Yagel R (2004) Toward self-stabilizing operating systems. In: 2nd international workshop on self-adaptive and autonomic computing systems, Zaragoza, Aug 2004, pp 684–688
7. Israeli A, Jalfon M (1990) Token management schemes and random walks yield self-stabilizing mutual exclusion. In: proceedings of the 9th annual ACM symposium on principles of distributed computing, Quebec City, Aug 1990, pp 119–131
8. Kutten S, Patt-Shamir B (1997) Time-adaptive self stabilization. In: Proceedings of the 16th annual ACM symposium on principles of distributed computing, Santa Barbara, Aug 1997, pp 149–158
9. Lamport L (1986) The mutual exclusion problem: part II-statement and solutions. J ACM 33(2):327–348
10. Varghese G, Jayaram M (2000) The fault span of crash failures. J ACM 47(2):244–293

Semi-supervised Learning

Avrim Blum
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

Keywords

Co-training; Learning from labeled and unlabeled data; Semi-supervised SVM

Years and Authors of Summarized Original Work

1998; Blum, Mitchell
1999; Joachims
2010; Balcan, Blum

Problem Definition

Semi-supervised learning [1, 4, 5, 8, 12] refers to the problem of using a large unlabeled data set U together with a given labeled data set L in order to generate prediction rules that are more accurate on new data than would have been achieved using just L alone. Semi-supervised learning is motivated by the fact that in many settings (e.g., document classification, image classification, speech recognition), unlabeled data is plentiful but labeled data is more limited or expensive, e.g., due to the need for human labelers. Therefore, one would like to make use of the unlabeled data if possible.

The general idea behind semi-supervised learning is that unlabeled data, while missing the labels, nonetheless often contains useful information. As an example, suppose one believes the correct decision boundary for some classification problem should be a linear separator that separates most of the data by a large margin. By observing enough unlabeled data to estimate the probability mass near to any given linear separator, one could in principle then discard separators in advance that slice through dense regions and instead focus attention on just

those that indeed separate most of the distribution by a large margin. This is the high-level idea behind semi-supervised SVMs. Alternatively, suppose data objects can be described by two different "kinds" of features, and one believes that each kind should be sufficient to produce an accurate classifier. Then one might want to train a *pair* of classifiers and use unlabeled data for which one classifier is confident but the other is not to bootstrap, labeling such examples with the confident classifier and then feeding them as training data to the less-confident classifier. This is the high-level idea behind Co-Training. Or, if one believes "similar examples should generally have the same label," one might construct a graph with an edge between examples that are sufficiently similar and aim for a classifier that is correct on the labeled data and has a small cut value on the unlabeled data; this is the high-level idea behind graph-based methods. (These will all be discussed in more detail later.) General surveys of semi-supervised learning appear in [5, 12].

A Formal Framework

We now present a formal model for analyzing semi-supervised learning due to Balcan and Blum [1]. This model was developed to provide a unified explanation for a wide range of semi-supervised learning algorithms including the semi-supervised SVMs, Co-Training, and graph-based methods mentioned above. Before describing it, however, we first describe the classic PAC and agnostic learning models for *supervised* learning that this model builds on.

In the PAC and agnostic learning models, data is assumed to be drawn iid from some fixed but initially unknown distribution D over an instance space $\mathcal{X}$ and labeled by some unknown target function $c^* : \mathcal{X} \to \{0, 1\}$. The *error* of some hypothesis function h is defined as $\text{err}(h) = \Pr_{x\sim D}[h(x) \neq c^*(x)]$. In the PAC model (also known as the *realizable case*), we assume that c^* is a member of some known class of functions $\mathcal{C}$, and we say that an algorithm PAC-learns $\mathcal{C}$ if for any given $\epsilon, \delta > 0$, with probability $\geq 1 - \delta$, it produces a hypothesis h such that $\text{err}(h) \leq \epsilon$. In the *agnostic case*, we do not assume that $c^* \in \mathcal{C}$ and instead aim to achieve error close to $\inf_{f\in\mathcal{C}}[\text{err}(f)]$.

The PAC and agnostic learning models in essence assume that one's prior beliefs about the target be described in terms of a class of functions $\mathcal{C}$. In order to capture the reasoning used in semi-supervised learning, however, we need to also describe beliefs about the *relation* between the target function and the data distribution. This is done in the model of Balcan and Blum [1] via a *notion of compatibility* χ between a hypothesis h and a distribution D. Formally, χ maps pairs (h, D) to $[0, 1]$ with $\chi(h, D) = 1$ meaning that h is highly compatible with D and $\chi(h, D) = 0$ meaning that h is very *in*compatible with D. The quantity $1 - \chi(h, D)$ is called the *unlabeled error rate* of h and denoted $\text{err}_{\text{unl}}(h)$. Note that for χ to be useful, it must be estimatable from a finite sample; to this end, χ is further required to be an expectation over individual examples. That is, overloading notation for convenience, we require $\chi(h, D) = \mathbf{E}_{x\sim D}[\chi(h, x)]$, where $\chi : \mathcal{C} \times \mathcal{X} \to [0, 1]$. As with the class $\mathcal{C}$, one can either assume that the target is fully compatible ($\text{err}_{\text{unl}}(c^*) = 0$) or instead aim to do well as a function of how compatible the target is. The case that we assume $c^* \in \mathcal{C}$ and $\text{err}_{\text{unl}}(c^*) = 0$ is termed the "doubly realizable case." The concept class $\mathcal{C}$ and compatibility notion χ are both viewed as *known*.

Examples

Suppose we believe the target should separate most data by a large margin γ. We can represent this belief by defining $\chi(h, x) = 0$ if x is within distance γ of the decision boundary of h and $\chi(h, x) = 1$ otherwise. In this case, $\text{err}_{\text{unl}}(h)$ will denote the probability mass of D within distance γ of h's decision boundary. Alternatively, if we do not wish to commit to a specific value of γ, we could define $\chi(h, x)$ to be a smooth function of the distance of x to the separator defined by h. As a very different example, in co-training (described in more detail below), we assume each example can be described using two "views" that each are sufficient for classification, that is, there exist c_1^*, c_2^* such that for each example $x = \langle x_1, x_2\rangle$, we have $c_1^*(x_1) = c_2^*(x_2)$. We can represent this belief by defining a hypothesis

$h = \langle h_1, h_2 \rangle$ to be compatible with an example $\langle x_1, x_2 \rangle$ if $h_1(x_1) = h_2(x_2)$ and incompatible otherwise; $\text{err}_{\text{unl}}(h)$ is then the probability mass of examples, under D, where the two halves of h disagree.

Intuition

In this framework, the way that unlabeled data helps in learning can be intuitively described as follows. Suppose one is given a concept class $\mathcal{C}$ (such as linear separators) and a compatibility notion χ (such as penalizing h for points within distance γ of the decision boundary). Suppose also that one believes $c^* \in \mathcal{C}$ (or at least is close) and that $\text{err}_{\text{unl}}(c^*) = 0$ (or at least is small). Then, unlabeled data can help by allowing one to estimate the *unlabeled error rate* of all $h \in \mathcal{C}$, thereby in principle reducing the search space from $\mathcal{C}$ (all linear separators) down to just the subset of $\mathcal{C}$ that is highly compatible with D. The key challenge is how this can be done efficiently (in theory, in practice, or both) for natural notions of compatibility, as well as identifying types of compatibility that data in important problems can be expected to satisfy.

Key Results

The following, from [1], illustrate formally how unlabeled data can help in this model. Fix some concept class $\mathcal{C}$ and compatibility notion χ. Given a labeled sample L, define $\widehat{\text{err}}(h)$ to be the fraction of mistakes of h on L. Given an unlabeled sample U, define $\chi(h, U) = \mathbf{E}_{x \sim U}[\chi(h, x)]$ and define $\widehat{\text{err}}_{\text{unl}}(h) = 1 - \chi(h, U)$. That is, $\widehat{\text{err}}(h)$ and $\widehat{\text{err}}_{\text{unl}}(h)$ are the empirical error rate and unlabeled error rate of h, respectively. Finally, given $\alpha > 0$, define $\mathcal{C}_{D,\chi}(\alpha)$ to be the set of functions $f \in \mathcal{C}$ such that $\text{err}_{\text{unl}}(f) \leq \alpha$.

Theorem 1 ([1]) *If $c^* \in \mathcal{C}$ then with probability at least $1 - \delta$, for a random labeled set L and unlabeled set U, the $h \in \mathcal{C}$ that optimizes $\widehat{\text{err}}_{\text{unl}}(h)$ subject to $\widehat{\text{err}}(h) = 0$ will have $\text{err}(h) \leq \epsilon$ for*

$$|U| \geq \frac{2}{\epsilon^2}\left[\ln|\mathcal{C}| + \ln\frac{4}{\delta}\right],$$

$$|L| \geq \frac{1}{\epsilon}\left[\ln|\mathcal{C}_{D,\chi}(\text{err}_{\text{unl}}(c^*) + 2\epsilon)| + \ln\frac{2}{\delta}\right].$$

Equivalently, for $|U|$ satisfying the above bound, for any $|L|$, with probability at least $1 - \delta$, the $h \in \mathcal{C}$ that optimizes $\widehat{\text{err}}_{\text{unl}}(h)$ subject to $\widehat{\text{err}}(h) = 0$ has

$$\text{err}(h) \leq \frac{1}{|L|}\left[\ln|\mathcal{C}_{D,\chi}(\text{err}_{\text{unl}}(c^*) + 2\epsilon)| + \ln\frac{2}{\delta}\right].$$

One can view Theorem 1 as bounding the number of labeled examples needed to learn well as a function of the "helpfulness" of the distribution D with respect to χ, for sufficiently large U. Namely, a helpful distribution is one in which $\mathcal{C}_{D,\chi}(\alpha)$ is small for α slightly larger than the compatibility of the true target function, so we do not need much labeled data to identify a good function among those in $\mathcal{C}_{D,\chi}(\alpha)$.

For infinite hypothesis classes, one needs to consider both the complexity of the class $\mathcal{C}$ and the complexity of the compatibility notion χ. Specifically, given $h \in \mathcal{C}$, define $\chi_h(x) = \chi(h, x)$ and let $\text{VCdim}(\chi(\mathcal{C}))$ denote the VC-dimension of the set $\{\chi_h | h \in \mathcal{C}\}$. A sample complexity bound from [1] based on ϵ-cover size is the following.

Theorem 2 ([1]) *Assume $c^* \in \mathcal{C}$ and let p be the size of the smallest set of functions H such that every function in $\mathcal{C}_{D,\chi}(\text{err}_{\text{unl}}(c^*) + \epsilon/3)$ is $\epsilon/6$-close to some function in H. Then $|U| = O\left(\frac{\max[\text{VCdim}(\mathcal{C}), \text{VCdim}(\chi(\mathcal{C}))]}{\epsilon^2}\ln\frac{1}{\epsilon} + \frac{1}{\epsilon^2}\ln\frac{2}{\delta}\right)$ and $|L| = O\left(\frac{1}{\epsilon}\ln\frac{p}{\delta}\right)$ is sufficient to identify a function $f \in \mathcal{C}$ of error at most ϵ with probability at least $1 - \delta$.*

Finally, for the general (agnostic) case that $c^* \notin \mathcal{C}$, we can define a regularizer based on empirical unlabeled error rates, and then get good bounds for optimizing a combination of the empirical labeled error and the regularization term. Specifically, for a hypothesis h, define $\hat{N}(h)$ to be the number of ways of partitioning the first $|L|$ points in U using $\{f \in \mathcal{C} : \widehat{\text{err}}_{\text{unl}}(f) \leq \widehat{\text{err}}_{\text{unl}}(h)\}$. Then we have

Theorem 3 ([1]) *With probability at least* $1-\delta$, *the hypothesis*

$$h = \arg\min_{h' \in \mathcal{C}} [\widehat{\text{err}}(h') + R(h')], \quad \textit{where}$$

$$R(h') = \sqrt{\frac{24 \ln(\hat{N}(h'))}{|L|}},$$

satisfies

$$\text{err}(h) \leq \min_{h' \in \mathcal{C}} \left[\text{err}(h') + R(h')\right] + 5\sqrt{\frac{\ln(8/\delta)}{|L|}}.$$

Co-Training

Co-Training is a semi-supervised learning method due to [4] for settings in which examples can be thought of as having two "views," that is, two distinct types of information. For example, in classifying webpages (e.g., into student home page, faculty member home page, course home page, etc.), one could use the words on the page itself, but one could also use information from links pointing *to* that page [4]. Or, in classifying visual images, one might have two cameras or even two different filters or preprocessing steps on images from the same camera [9]. Or, in understanding video, one can use visual images and spoken dialogue [7]. In such settings, one can think of an example x as a pair $x = \langle x_1, x_2 \rangle$. The idea of Co-Training is that if each view is in principle enough to achieve a good classification by itself, but each provides somewhat different information, then one can hope to improve performance using unlabeled data. Specifically, in Co-Training, one maintains two hypotheses, one for each view (e.g., a hypothesis that classifies webpages based on the text on the page itself and one that classifies webpages based on information from links pointing to the page). A hypothesis pair $h = \langle h_1, h_2 \rangle$ is compatible with an example $\langle x_1, x_2 \rangle$ if $h_1(x_1) = h_2(x_2)$ and is incompatible otherwise. So the unlabeled error rate of a hypothesis (pair) $h = \langle h_1, h_2 \rangle$ is the probability mass of examples $\langle x_1, x_2 \rangle$ on which the two parts of h disagree.

In practice, there are two primary ways that this notion of compatibility is used to learn from a small amount of labeled data and a large amount of unlabeled data. The first is *iterative co-training*, introduced in [4]. In iterative co-training, a small labeled sample L is used to produce predictors for each view that are confident in some part of their respective input spaces and not confident in other parts. Then, the algorithm searches through the (large) unlabeled set U to find examples $x = \langle x_1, x_2 \rangle$ for which one classifier is confident and the other is not. These examples are labeled by the confident classifier and handed to the less-confident classifier to improve its predictor. The other primary method is to optimize a global objective that combines accuracy over the labeled sample L with agreement over the unlabeled sample U. That is, one searches for the hypothesis pair h that minimizes $\widehat{\text{err}}(h) + \lambda \widehat{\text{err}}_{\text{unl}}(h)$ for some regularization parameter λ [6, 10]. This is generally a non-convex optimization problem, and so various heuristics are typically applied to perform the optimization.

Theoretically, the guarantees for Co-Training are strongest when the data satisfies independence given the label (with some probability p, a random positive example $\langle x_1, x_2 \rangle$ is drawn from $D_1^+ \times D_2^+$, and with probability $1-p$, a random negative example is drawn from $D_1^- \times D_2^-$) and in the realizable case (there exist targets $c_1^*, c_2^* \in \mathcal{C}$ such that all examples $\langle x_1, x_2 \rangle$ in the support of the distribution satisfy $c_1^*(x_1) = c_2^*(x_2)$). Specifically, two key results are

Theorem 4 ([4]) *Any class* $\mathcal{C}$ *that is efficiently PAC-learnable from random classification noise is efficiently learnable from unlabeled data alone in the realizable Co-Training setting, if data satisfies independence given the label and one is given an initial weakly useful predictor* $h_1(x_1)$.

Here, h is a *weakly useful predictor* of a function f if for some $\epsilon > 1/poly(n)$ we have both (a) $\Pr_{x \sim D}[h(x) = 1] \geq \epsilon$ and (b) $\Pr_{x \sim D}[f(x) = 1 | h(x) = 1] \geq \Pr_{x \sim D}[f(x) = 1] + \epsilon$. Theorem 4 implies that if one is able to use a small labeled sample to produce an initial hypothesis that gives a slight "edge" in predicting the target beyond just the overall class probabilities, then under independence given the label one can boost that to a high-accuracy predictor from just unlabeled data.

Furthermore, ignoring computation time, under independence given the label, any class of finite VC-dimension is learnable from a single labeled example. In the case of linear separators, this can be done computationally efficiently.

Theorem 5 ([1]) *Any class $\mathcal{C}$ of finite VC-dimension is learnable from polynomially many unlabeled examples and a single labeled example if D satisfies independence given the label. Furthermore, for linear separators this can be done in polynomial time.*

Semi-supervised SVMs

Semi-supervised SVMs (also called transductive SVMs) [8, 11] aim to find a linear separator that separates both the labeled sample L and the unlabeled sample U by the largest possible margin. That is, one wants to find a separator such that for γ as large as possible, all labeled examples are on the correct side of the separator by distance at least γ and all unlabeled examples are on *some* side of the separator by distance at least γ. In practice, one combines a large-margin objective with a hinge-loss penalty for labeled examples that fail to satisfy the condition, and a "hat-loss" penalty for unlabeled examples that fail to satisfy the condition. Formally, the goal is to minimize $c_1 w^T w + c_2 \sum_{(x_i,y_i)\in L} \alpha_i + c_3 \sum_{x_j \in U} \beta_j$ subject to $(w^T x_i) y_i \geq 1 - \alpha_i$ for all $(x_i, y_i) \in L$ and $(w^T x_j)\tilde{y}_j \geq 1 - \beta_j$ for all $x_j \in U$ (and $\alpha_i, \beta_j \geq 0$), where $y_i \in \{-1, 1\}$ is the (known) label of $x_i \in L$ and $\tilde{y}_j \in \{-1, 1\}$ is a variable representing the algorithm's guess of the label of $x_j \in U$. While the optimization problem is NP-hard, a number of heuristics have been developed. For example, Joachims [8] uses an iterative labeling heuristic to approximately optimize the objective. Semi-supervised SVMs have been shown to achieve high accuracy in a number of text classification domains where unlabeled data is plentiful [8].

Graph-Based Methods

Graph-based methods [3, 13] can be viewed as a (transductive) semi-supervised version of nearest-neighbor learning. In these methods, one creates a graph with a vertex for each example in $L \cup U$ and an edge between two examples x, x' if they are deemed to be sufficiently "similar" (or with edge weights based on *how* similar they are deemed to be). Similarity can be directly based on distance between the examples in the input space or given by some provided kernel function $k(x, x')$. Given the labels for the examples in L, one then finds a "most compatible" labeling for the examples in U, based on the belief that similar examples will typically have the same label. Specifically, in the mincut approach of [3], the labeling h produced is the cut of least total weight subject to agreeing with the known labels on examples in L or equivalently the cut that agrees with L minimizing $\sum_{e=(x,x')} w_e |h(x) - h(x')|$. In the algorithm of [13], in order to produce a smoother solution, the algorithm instead views the graph as an electrical network, finding the cut agreeing with L that minimizes $\sum_{e=(x,x')} w_e (h(x) - h(x'))^2$.

Open Problems

There are a number of open problems in developing computationally efficient semi-supervised learning algorithms. For example, can one extend the algorithm of Theorem 5 for Co-Training with linear separators to weaker conditions than independence given the label, while maintaining computational efficiency? (Note: A number of weaker conditions are known to produce good sample bounds if computational considerations are ignored [2].) More broadly, can one develop efficient algorithms for other classes or notions of compatibility that meet the cover-based sample complexity bounds of Theorem 2? Additional open problems are given in [1].

Recommended Reading

1. Balcan MF, Blum A (2010) A discriminative model for semi-supervised learning. J ACM

57(3):19:1–19:46. doi:10.1145/1706591.1706599. http://doi.acm.org/10.1145/1706591.1706599
2. Balcan MF, Blum A, Yang K (2004) Co-training and expansion: towards bridging theory and practice. In: Proceedings of 18th conference on neural information processing systems, Vancouver
3. Blum A, Chawla S (2001) Learning from labeled and unlabeled data using graph mincuts. In: Proceedings of 18th international conference on machine learning, Williams College
4. Blum A, Mitchell TM (1998) Combining labeled and unlabeled data with co-training. In: Proceedings of the 11th annual conference on computational learning theory, Madison, pp 92–100
5. Chapelle O, Schölkopf B, Zien A (eds) (2006) Semi-supervised learning. MIT, Cambridge. http://www.kyb.tuebingen.mpg.de/ssl-book
6. Collins M, Singer Y (1999) Unsupervised models for named entity classification. In: Proceedings of the joint SIGDAT conference on empirical methods in natural language processing and very large corpora, College Park, pp 189–196
7. Gupta S, Kim J, Grauman K, Mooney R (2008) Watch, listen & learn: co-training on captioned images and videos. In: Machine learning and knowledge discovery in databases (ECML PKDD). Lecture notes in computer science, vol 5211. Springer, Berlin/Heidelberg, pp 457–472. 10.1007/978-3-540-87479-9_48. http://dx.doi.org/10.1007/978-3-540-87479-9_48
8. Joachims T (1999) Transductive inference for text classification using support vector machines. In: Proceedings of 16th international conference on machine learning, Bled, pp 200–209
9. Levin A, Viola P, Freund Y (2003) Unsupervised improvement of visual detectors using co-training. In: Proceedings of the ninth IEEE international conference on computer vision, ICCV '03, vol 2, Nice. IEEE Computer Society, Washington, DC, pp 626–633. http://dl.acm.org/citation.cfm?id=946247.946615
10. Nigam K, Ghani R (2000) Analyzing the effectiveness and applicability of co-training. In: Proceedings of ACM CIKM international conference on information and knowledge management, McLean, pp 86–93
11. Vapnik V (1998) Statistical learning theory, vol 2. Wiley, New York
12. Zhu X (2006) Semi-supervised learning literature survey Computer sciences TR 1530 University of Wisconsin, Madison
13. Zhu X, Ghahramani Z, Lafferty J (2003) Semi-supervised learning using Gaussian fields and harmonic functions. In: Proceedings of 20th international conference on machine learning, Washington, DC, pp 912–919

Separators in Graphs

Goran Konjevod
Department of Computer Science and Engineering, Arizona State University, Tempe, AZ, USA

Keywords

Balanced cuts

Years and Authors of Summarized Original Work

1998; Leighton, Rao
1999; Leighton, Rao

Problem Definition

The (balanced) separator problem asks for a cut of minimum (edge)-weight in a graph, such that the two shores of the cut have approximately equal (node)-weight.

Formally, given an undirected graph $G = (V, E)$, with a nonnegative edge-weight function $c : E \to \mathbb{R}_+$, a nonnegative node-weight function $\pi : V \to \mathbb{R}_+$, and a constant $b \leq 1/2$, a cut $(S : V \setminus S)$ is said to be *b-balanced*, or a *$(b, 1-b)$-separator*, if $b\pi(V) \leq \pi(S) \leq (1-b)\pi(V)$ (where $\pi(S)$ stands for $\sum_{v \in S} \pi(v)$).

Problem 1 (b-balanced separator)
INPUT: Edge- and node-weighted graph $G = (V, E, c, \pi)$, constant $b \leq 1/2$.
OUTPUT: A b-balanced cut $(S : V \setminus S)$. Goal: minimize the edge weight $c(\delta(S))$.

Closely related is the *product sparsest cut problem.*

Problem 2 ((Product) Sparsest cut)
INPUT: Edge- and node-weighted graph $G = (V, E, c, \pi)$.
OUTPUT: A cut $(S : V \setminus S)$ minimizing the ratio-cost $\frac{c}{c}(\delta(S)))/(\pi(S)\pi(V \setminus S))$.

Problem 2 is the most general version of sparsest cut solved by Leighton and Rao. Setting all

S

node weights are equal to 1 leads to the uniform version, Problem 3.

Problem 3 ((Uniform) Sparsest cut)
INPUT: Edge-weighted graph $G = (V, E, c)$.
OUTPUT: A cut $(S : V \setminus S)$ minimizing the ratio-cost $(c(\delta(S)))/(|S||V \setminus S|)$.

Sparsest cut arises as the (integral version of the) linear programming dual of *concurrent multicommodity flow* (Problem 4). An instance of a multicommodity flow problem is defined on an edge-weighted graph by specifying for each of k *commodities* a *source* $s_i \in V$, a *sink* $t_i \in V$, and a *demand* D_i. A feasible solution to the multicommodity flow problem defines for each commodity a flow function on E, thus routing a certain amount of flow from s_i to t_i. The edge weights represent capacities, and for each edge e, a capacity constraint is enforced: the sum of all commodities' flows through e is at most the capacity $c(e)$.

Problem 4 (Concurrent multicommodity flow)
INPUT: Edge-weighted graph $G = (V, E, c)$, commodities $(s_1, t_1, D_1), \ldots (s_k, t_k, D_k)$.
OUTPUT: A multicommodity flow that routes fD_i units of commodity i from s_i to t_i for each i simultaneously, without violating the capacity of any edge. Goal: maximize f.

Problem 4 can be solved in polynomial time by linear programming, and approximated arbitrarily well by several more efficient combinatorial algorithms (section "Implementation"). The maximum value f for which there exists a multicommodity flow is called the *max-flow* of the instance. The *min-cut* is the minimum ratio $(c(\delta(S)))/(D(S, V \setminus S))$, where $D(S, V \setminus S) = \sum_{i:|\{s_i,t_i\}\cap S|=1} D_i$. This dual interpretation motivates the most general version of the problem, the *nonuniform sparsest cut* (Problem 5).

Problem 5 ((Nonuniform) Sparsest cut)
INPUT: Edge-weighted graph $G = (V, E, c)$, commodities $(s_1, t_1, D_1), \ldots (s_k, t_k, D_k)$.
OUTPUT: A min-cut $(S : V \setminus S)$, that is, a cut of minimum ratio-cost $(c(\delta(S)))/(D(S, V \setminus S))$.

(Most literature focuses on either the uniform or the general nonuniform version, and both of these two versions are sometimes referred to as just the "sparsest cut" problem.)

Key Results

Even when all (edge- and node-) weights are equal to 1, finding a minimum-weight b-balanced cut is NP-hard (for $b = 1/2$, the problem becomes *graph bisection*). Leighton and Rao [23, 24] give a pseudo-approximation algorithm for the general problem.

Theorem 1 *There is a polynomial-time algorithm that, given a weighted graph $G = (V, E, c, \pi)$, $b \leq 1/2$ and $b' < \min\{b, 1/3\}$, finds a b'-balanced cut of weight $O((\log n)/(b - b'))$ times the weight of the minimum b-balanced cut.*

The algorithm solves the sparsest cut problem on the given graph, puts aside the smaller-weight shore of the cut, and recurses on the larger-weight shore until both shores of the sparsest cut found have weight at most $(1-b')\pi(G)$. Now the larger-weight shore of the last iteration's sparsest cut is returned as one shore of the balanced cut, and everything else as the other shore. Since the sparsest cut problem is itself NP-hard, Leighton and Rao first required an approximation algorithm for this problem.

Theorem 2 *There is a polynomial-time algorithm with approximation ratio $O(\log p)$ for product sparsest cut (Problem 2), where p denotes the number of nonzero-weight nodes in the graph.*

This algorithm follows immediately from Theorem 3.

Theorem 3 *There is a polynomial-time algorithm that finds a cut $(S : V \setminus S)$ with ratio-cost $(c(\delta(S)))/(\pi(S)\pi(V \setminus S)) \in O(f \log p)$, where*

f is the max-flow for the product multicommodity flow and p the number of nodes with nonzero weight.

The proof of Theorem 3 is based on solving a linear programming formulation of the multicommodity flow problem and using the solution to construct a sparse cut.

Related Results

Shahrokhi and Matula [27] gave a max-flow min-cut theorem for a special case of the multicommodity flow problem and used a similar LP-based approach to prove their result. An $O(\log n)$ upper bound for arbitrary demands was proved by Aumann and Rabani [6] and Linial et al. [26]. In both cases, the solution to the dual of the multicommodity flow linear program is interpreted as a finite metric and embedded into ℓ_1 with distortion $O(\log n)$, using an embedding due to Bourgain [10]. The resulting ℓ_1 metric is a convex combination of cut metrics, from which a cut can be extracted with sparsity ratio at least as good as that of the combination.

Arora et al. [5] gave an $O(\sqrt{\log n})$ pseudo-approximation algorithm for (uniform or product-weight) balanced separators, based on a semidefinite programming relaxation. For the nonuniform version, the best bound is $O(\sqrt{\log n}\log\log n)$ due to Arora et al. [4]. Khot and Vishnoi [18] showed that, for the nonuniform version of the problem, the semidefinite relaxation of [5] has an integrality gap of at least $(\log\log n)^{1/6-\delta}$ for any $\delta > 0$, and further, assuming their Unique Games Conjecture, that it is NP-hard to (pseudo)-approximate the balanced separator problem to within any constant factor. The SDP integrality gap was strengthened to $\Omega(\log\log n)$ by Krauthgamer and Rabani [20]. Devanur et al. [11] show an $\Omega(\log\log n)$ integrality gap for the SDP formulation even in the uniform case.

Implementation

The bottleneck in the balanced separator algorithm is solving the multicommodity flow linear program. There exists a substantial amount of work on fast approximate solutions to such linear programs [19, 22, 25]. In most of the following results, the algorithm produces a $(1+\epsilon)$-approximation, and its hidden constant depends on ϵ^{-2}. Garg and Könemann [15], Fleischer [14] and Karakostas [16] gave efficient approximation schemes for multicommodity flow and related problems, with running times $\tilde{O}((k+m)m)$ [15] and $\tilde{O}(m^2)$ [14, 16]. Benczúr and Karger [7] gave an $O(\log n)$ approximation to sparsest cut based on randomized minimum cut and running in time $\tilde{O}(n^2)$. The current fastest $O(\log n)$ sparsest cut (balanced separator) approximation is based on a primal-dual approach to semidefinite programming due to Arora and Kale [3], and runs in time $O(m+n^{3/2})(\tilde{O}(m+n^{3/2})$, respectively). The same paper gives an $O(\sqrt{\log n})$ approximation in time $O(n^2)(\tilde{O}(n^2)$, respectively), improving on a previous $\tilde{O}(n^2)$ algorithm of Arora et al. [2]. If an $O(\log^2 n)$ approximation is sufficient, then sparsest cut can be solved in time $\tilde{O}(n^{3/2})$, and balanced separator in time $\tilde{O}(m+n^{3/2})$ [17].

Applications

Many problems can be solved by using a balanced separator or sparsest cut algorithm as a subroutine. The approximation ratio of the resulting algorithm typically depends directly on the ratio of the underlying subroutine. In most cases, the graph is recursively split into pieces of balanced size. In addition to the $O(\log n)$ approximation factor required by the balanced separator algorithm, this leads to another $O(\log n)$ factor due to the recursion depth. Even et al. [12] improved many results based on balanced separators by using *spreading metrics*, reducing the approximation guarantee to $O(\log n\log\log n)$ from $O(\log^2 n)$.

Some applications are listed here; where no reference is given, and for further examples, see [24].

- Minimum cut linear arrangement and minimum feedback arc set. One single algorithm provides an $O(\log^2 n)$ approximation for both of these problems.

- Minimum chordal graph completion and elimination orderings [1]. Elimination orderings are useful for solving sparse symmetric linear systems. The $O(\log^2 n)$ approximation algorithm of [1] for chordal graph completion has been improved to $O(\log n \log\log n)$ by Even et al. [12].
- Balanced node cuts. The cost of a balanced cut may be measured in terms of the weight of nodes removed from the graph. The balanced separator algorithm can be easily extended to this node-weighted case.
- VLSI layout. Bhatt and Leighton [8] studied several optimization problems in VLSI layout. Recursive partitioning by a balanced separator algorithm leads to polylogarithmic approximation algorithms for crossing number, minimum layout area and other problems.
- Treewidth and pathwidth. Bodlaender et al. [9] showed how to approximate treewidth within $O(\log n)$ and pathwidth within $O(\log^2 n)$ by using balanced node separators.
- Bisection. Feige and Krauthgamer [13] gave an $O(\alpha \log n)$ approximation for the minimum bisection, using any α-approximation algorithm for sparsest cut.

Experimental Results

Lang and Rao [21] compared a variant of the sparsest cut algorithm from [24] to methods used in graph decomposition for VLSI design.

Cross-References

- ▸ Fractional Packing and Covering Problems
- ▸ Minimum Bisection
- ▸ Sparsest Cut

Recommended Reading

Further details and pointers to additional results may be found in the survey [28].

1. Agrawal A, Klein PN, Ravi R (1993) Cutting down on fill using nested dissection: provably good elimination orderings. In: Brualdi RA, Friedland S, Klee V (eds) Graph theory and sparse matrix computation. IMA volumes in mathematics and its applications. Springer, New York, pp 31–55
2. Arora S, Hazan E, Kale S (2004) $O(\sqrt{\log n})$ approximation to sparsest cut in $\tilde{O}(n^2)$ time. In: FOCS '04: proceedings of the 45th annual IEEE symposium on foundations of computer science (FOCS'04). IEEE Computer Society, Washington, pp 238–247
3. Arora S, Kale S (2007) A combinatorial, primal-dual approach to semidefinite programs. In: STOC '07: proceedings of the 39th annual ACM symposium on theory of computing. ACM, pp 227–236
4. Arora S, Lee JR, Naor A (2005) Euclidean distortion and the sparsest cut. In: STOC '05: proceedings of the thirty-seventh annual ACM symposium on theory of computing. ACM, New York, pp 553–562
5. Arora S, Rao S, Vazirani U (2004) Expander flows, geometric embeddings and graph partitioning. In: STOC '04: proceedings of the thirty-sixth annual ACM symposium on theory of computing. ACM, New York, pp 222–231
6. Aumann Y, Rabani Y (1998) An (log) approximate min-cut maxflow theorem and approximation algorithm. SIAM J Comput 27(1):291–301
7. Benczúr AA, Karger DR (1996) Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In: STOC '96: proceedings of the twenty-eighth annual ACM symposium on theory of computing. ACM, New York, pp 47–55
8. Bhatt SN, Leighton FT (1984) A framework for solving vlsi graph layout problems. J Comput Syst Sci 28(2):300–343
9. Bodlaender HL, Gilbert JR, Hafsteinsson H, Kloks T (1995) Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. J Algorithms 18(2):238–255
10. Bourgain J (1985) On Lipshitz embedding of finite metric spaces in Hilbert space. Isr J Math 52:46–52
11. Devanur NR, Khot SA, Saket R, Vishnoi NK (2006) Integrality gaps for sparsest cut and minimum linear arrangement problems. In: STOC '06: proceedings of the thirty-eighth annual ACM symposium on theory of computing. ACM, New York, pp 537–546
12. Even G, Naor JS, Rao S, Schieber B (2000) Divide-and-conquer approximation algorithms via spreading metrics. J ACM 47(4):585–616
13. Feige U, Krauthgamer R (2002) A polylogarithmic approximation of the minimum bisection. SIAM J Comput 31(4):1090–1118
14. Fleischer L (2000) Approximating fractional multicommodity flow independent of the number of commodities. SIAM J Discret Math 13(4):505–520
15. Garg N, Könemann J (1998) Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: FOCS '98: proceedings of the 39th annual symposium on foundations of computer science. IEEE Computer Society, Washington, p 300

16. Karakostas G (2002) Faster approximation schemes for fractional multicommodity flow problems. In: SODA '02: proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, pp 166–173
17. Khandekar R, Rao S, Vazirani U (2006) Graph partitioning using single commodity flows. In: STOC '06: proceedings of the thirty-eighth annual ACM symposium on theory of computing. ACM, New York, pp 385–390
18. Khot S, Vishnoi NK (2005) The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into l_1. In: FOCS '07: proceedings of the 46th annual IEEE symposium on foundations and computer science. IEEE Computer Society, pp 53–62
19. Klein PN, Plotkin SA, Stein C, Tardos É (1994) Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. SIAM J Comput 23(3):466–487
20. Krauthgamer R, Rabani Y (2006) Improved lower bounds for embeddings into l_1. In: SODA '06: proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithm. ACM, New York, pp 1010–1017
21. Lang K, Rao S (1993) Finding near-optimal cuts: an empirical evaluation. In: SODA '93: proceedings of the fourth annual ACM SIAM symposium on discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, pp 212–221
22. Leighton FT, Makedon F, Plotkin SA, Stein C, Stein É, Tragoudas S (1995) Fast approximation algorithms for multicommodity flow problems. J Comput Syst Sci 50(2):228–243
23. Leighton T, Rao S (1988) An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In: Proceedings of the 29th annual symposium on foundations of computer science. IEEE Computer Society, Washington, DC, pp 422–431
24. Leighton T, Rao S (1999) Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. J ACM 46(6):787–832
25. Leong T, Shor P, Stein C (1991) Implementation of a combinatorial multicommodity flow algorithm. In: Johnson DS, McGeoch CC (eds) Network flows and matching. DIMACS series in discrete mathematics and theoretical computer science, vol 12, AMS, Providence, pp 387–406
26. Linial N, London E, Rabinovich Y (1995) The geometry of graphs and some of its algorithmic applications. Combinatorica 15(2):215–245
27. Shahrokhi F, Matula DW (1990) The maximum concurrent flow problem. J ACM 37(2):318–334
28. Shmoys DB (1997) Cut problems and their applications to divide-and-conquer. In: Hochbaum DS (ed) Approximation algorithms for NP-hard problems. PWS Publishing Company, pp 192–235

Sequence and Spatial Motif Discovery in Short Sequence Fragments

Jie Liang[1] and Ronald Jackups[2]
[1]Department of Bioengineering, University of Illinois, Chicago, IL, USA
[2]Department of Pediatrics, Washington University, St. Louis, MO, USA

Keywords

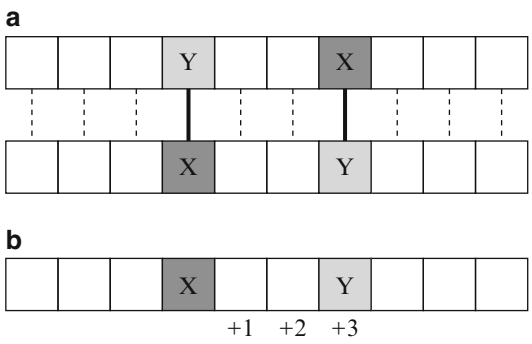
Internally random model; Permutation model; Positional null model; Sequence motif; Spatial motif; String pairing pattern; String pattern

Years and Authors of Summarized Original Work

2005; Jackups, Jr and Liang
2006; Jackups, Jr, Cheng, Liang
2010; Jackups, Jr and Liang

Problem Definition

The problem is to detect specific patterns in string and patterns in string pairs for discovery of sequence and spatial motifs in membrane proteins. A *spatial interaction* motif of residue pair X-Y is defined as a pattern in which a character or residue of type X is found interacting with a residue of type Y on two strings or sequences (Fig. 1a). We define a *sequence pair* XYk as a pattern in which a residue of type Y is found at the k-th position from a residue of type X along a single sequence (Fig. 1b). The propensity $P(X, Y)$ of residue pairing XY is $P(X, Y) = \frac{f_{\text{obs}}(X,Y)}{\mathbb{E}[f(X,Y)]}$, where $f_{\text{obs}}(X, Y)$ is the observed count of XY patterns and $\mathbb{E}[f(X, Y)]$ is the expected count of XY patterns according to some random null model. We define a *motif* as a residue pair with propensity >1.0 (or greater than some other predefined limit) and statistically significant.



Sequence and Spatial Motif Discovery in Short Sequence Fragments, Fig. 1 Examples of spatial and sequence patterns. (**a**) Two X-Y spatial patterns on interacting sequences. (**b**) An $XY3$ sequence pattern

The null model for calculating $\mathbb{E}[f(X,Y)]$ is critical for motif detection. For short sequence fragments, the null model for spatial motif detection cannot be the χ^2 distribution as was used in [13], since the assumption of Gaussian distribution is not valid for short sequences. The null model for sequence motif detection cannot be the binomial distribution as was used in [4, 10], since the assumption of drawing from a universal population with replacement is unrealistic for short sequence fragments. Instead, we use a combinatorial model called the *permutation model* more effective for discoveries of motifs [5–7]. This null model is similar for both pair types: the residues within each sequence are exhaustively and independently permuted *without replacement*, and each permutation occurs with equal probability. This model has been called the *internally random* model [6]. This permutation model is further extended to positional null model to correct position-specific bias in residue distributions [6].

Objective. Our task is to determine explicit formulas to calculate $\mathbb{E}[f(X,Y)]$ for each pair type under different conditions. Explicit probability distributions for $f(X,Y)$ can also be found for many special cases, which will allow for the calculation of statistical significance p-values. These formulas can also be used to study whole datasets of short sequences.

Key Results

Spatial Motifs by Permutation Model

Expectation for Interacting Residues of the Same Type

For cases in which X is the same as Y (i.e., X-X pairs), let x_1 be the number of residues of type X in the first sequence, x_2 the number of residues of type X in the second sequence, and l the common length of the sequence pair. The probability $\mathbb{P}_{XX}(i)$ of exactly $i = f(X,X)$ number of X-X contacts follows a hypergeometric distribution: $\mathbb{P}_{XX}(i) = \binom{x_1}{i}\binom{l-x_1}{x_2-i}/\binom{l}{x_2}$. Its expectation $\mathbb{E}[f(X,X)]$ is then:

$$\mathbb{E}[f(X,X)] = \frac{x_1 x_2}{l}.$$

Expectation for Interacting Residues of Different Types

When $X \neq Y$, the number of X-Y contacts in the permutation model for one sequence pair is the sum of two dependent hypergeometric variables, one variable for type X residues in the first sequence s_1 and type Y in the second sequence s_2, and another variable for type Y residues in s_1 and type X in s_2. The expected number of X-Y contacts $\mathbb{E}[f(X,Y)]$ is the sum of the two expected values $\mathbb{E}[f(X,Y|X \in s_1, Y \in s_2)] + \mathbb{E}[f(X,Y|Y \in s_1, X \in s_2)]$:

$$\mathbb{E}[f(X,Y)] = \frac{x_1 y_2}{l} + \frac{y_1 x_2}{l},$$

where x_1 and x_2 are the numbers of residues of type X in the first and second sequence, respectively, y_1 and y_2 are the numbers of residues of type Y in the first and second sequence, respectively, and l is the length of the sequence pair.

Significance of Spatial Motifs

To calculate the statistical significance in the form of p-value of interacting residues of the same type, two-tailed p-values can be calculated using the hypergeometric distribution for a dataset of sequence pairs.

For interacting residues of different types, the formula to determine the p-value for a specific

observed number of X-Y contacts is more complex because of the dependency. We define a 3-element multinomial function $M(a,b,c) \equiv \frac{a!}{b!c!(a-b-c)!}$, where $M(a,b,c) = 0$ if $a-b-c < 0$. This represents the number of distinct permutations, without replacement, in a multiset of size a containing three different types of elements, with number count b, c, and $a-b-c$ of each of the three element types.

The probability $\mathbb{P}(h,i,j,k)$ of inter-sequence matches, namely, the probability of h X-X contacts, i X-Y contacts, j Y-X contacts, and k Y-Y contacts occurring in a random permutation is (Fig. 2)

$$\mathbb{P}(h,i,j,k) = \frac{M(x_1,h,i)\cdot M(y_1,j,k)\cdot M(l-x_1-y_1, x_2-h-j, y_2-i-k)}{M(l,x_2,y_2)}.$$

The marginal probability $\mathbb{P}_{XY}(m)$ that there are a total of $i+j=m$ X-Y contacts is

$$\mathbb{P}_{XY}(m) = \sum_{h=0}^{x_1} \sum_{i=0}^{x_1-h} \sum_{k=0}^{y_1-(m-i)} \mathbb{P}(h,i,m-i,k).$$

There are x_1 possible values for h, one for each residue of type X on sequence 1; x_1-h possible values for i, once h has been determined; and $y_1-j = y_1-(m-i)$ possible values for k, once i has been determined. The i number of X-Y contacts plus the $m-i$ number of Y-X contacts will sum to the m number of contacts desired.

This closed-form formula allows calculation of p-values analytically. The running time is $O(l^4)$, due to the presence of 3 summations and $l!$ in the summand. For short sequences, the computing cost is not prohibitive.

Sequences of Different Lengths

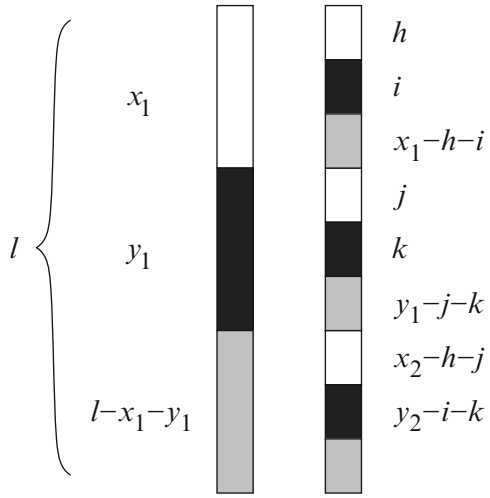
The requirement for interacting sequences to be of the same length may be relaxed by introducing a 21st "dummy" amino acid type. All unpaired residues in the longer member of a sequence pair will be paired to this extra amino acid type, and our standard method can be applied to determine the propensity of unpaired amino acids (i.e., residues paired with the "dummy" amino acid type).

Sequence Motifs by Permutation Model

The propensity $P(X,Y|k)$ for the XYk pattern of two ordered intrasequence residues of type X and type Y that are k positions away on the same sequence (Fig. 1b) is $P(X,Y|k) = \frac{f_{\text{obs}}(X,Y|k)}{\mathbb{E}[f(X,Y|k)]}$, where $f_{\text{obs}}(X,Y|k)$ is the observed count of XYk patterns, and $\mathbb{E}[f(X,Y|k)]$ is the expected count of XYk patterns.

Expectation of XYk and XXk Two-Residue Motifs

We can regard $f(X,Y|k)$ as the sum of identical Bernoulli variables $f_t(X,Y|k)$, each of which equals 1 if one of the x number of residues of type X occurs at position t in the sequence and one of the y number of residues of type Y occurs



Sequence and Spatial Motif Discovery in Short Sequence Fragments, Fig. 2 Division of residues in spatial motif analysis when $X \neq Y$. *White* = X, *black* = Y, *gray* = "neither" X or Y

at position $t + k$ or equals 0 otherwise. Since an XYk pattern cannot occur if $t > l - k$, we concern ourselves only with the first $l - k$ positions. We have: $\mathbb{E}[f_t(X, Y|k)] = \mathbb{P}[f_t(X, Y|k) = 1] = \frac{x}{l} \cdot \frac{y}{(l-1)}$ if $t \leq l-k$. There are $l-k$ such identical variables, and their expectations may be summed as

$$\mathbb{E}[f(X, Y|k)] = (l - k)\frac{xy}{l(l-1)}, \quad (1)$$

where l is the length of the sequence, x is the number of residues of type X, and y is the number of residues of type Y.

For XXk patterns, the expectation is calculated as

$$\mathbb{E}[f(X, X|k)] = (l - k)\frac{x(x-1)}{l(l-1)}, \quad (2)$$

as there will be $x - 1$ residues available to place the second X residue at position $t + k$ after the first X residue is placed at t. Although these Bernoulli random variables are dependent (i.e., the placement of one XYk pattern will affect the probability of another XYk pattern), their expectations may be summed, because expectation is a linear operator.

Significance of XYk and XXk Two-Residue Sequence Motifs

To calculate statistical significance p-values, several formulas have been derived to determine $\mathbb{P}_{XYk}(i)$, the probability of the occurrence of $i = f(X, Y|k)$ XYk patterns for different k values.

1. **Sequence motifs when $k = 1$.** We have

$$\mathbb{P}_{XY1}(i) = \frac{\binom{l-y}{x}\binom{x}{i}\binom{l-x}{y-i}}{\frac{l!}{x!y!(l-x-y)!}} = \frac{\binom{x}{i}\binom{l-x}{y-i}}{\binom{l}{y}}, \quad \text{and}$$

$$\mathbb{P}_{XX1}(i) = \frac{\binom{l-x+1}{x-i}\binom{x-1}{i}}{\binom{l}{x}},$$

with the convention that $\binom{n}{r} = 0$ if $n < r$.

2. **Sequence motifs with residues of different types and if $x \leq 2$ or $y \leq 2$.**
   - If either $x = 1$ or $y = 1$, we have

$$\mathbb{P}_{XYk}(1) = (l - k)\frac{xy}{l(l-1)}.$$

   For $i = 0$, we have simply $\mathbb{P}_{XYk}(0) = 1 - \mathbb{P}_{XYk}(1)$.
   - If $x = 2$ or $y = 2$, the probability of two XYk patterns is

$$\mathbb{P}_{XYk}(2) = \frac{\left[\binom{l-k}{2} - (l - 2k)\right]}{\frac{l(l-1)(l-2)(l-3)}{x(x-1)y(y-1)}}.$$

   We also have for the probabilities of exactly one XYk pattern or zero pattern:

$$\mathbb{P}_{XYk}(1) = \mathbb{E}[f(XYk)] - 2\mathbb{P}_{XYk}(2) \quad \text{and}$$

$$\mathbb{P}_{XYk}(0) = 1 - [\mathbb{P}_{XYk}(1) + \mathbb{P}_{XYk}(2)].$$

3. **Sequence motifs with residues of the same type if $x \leq 3$.**
   - If $x = 2$, the probability of one XXk pattern is

$$\mathbb{P}_{XXk}(1) = \mathbb{E}[f(XXk)] = (l-k)\frac{x(x-1)}{l(l-1)},$$

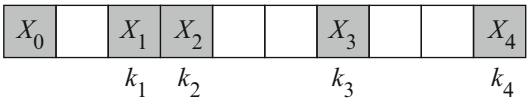
   The probability of no XXk pattern is

$$\mathbb{P}_{XXk}(0) = 1 - \mathbb{P}_{XXk}(1).$$

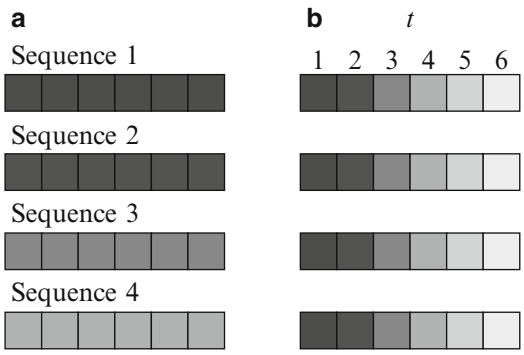
   - If $x = 3$, the probability of exactly two XXk patterns is

$$\mathbb{P}_{XXk}(2) = \frac{l - 2k}{\binom{l}{x}},$$

4. **Sequence motifs with $k > 1$, $x > 2$, and $y > 2$.** When $k > 1$, $x > 2$, and $y > 2$, the analytical formulas for $\mathbb{P}_{XYk}(i)$ become very complicated. However, when the sequences in the dataset used are short, it is possible to fully enumerate all permutations of a sequence and calculate $\mathbb{P}_{XYk}(i)$ and p-values exactly, as shown by Senes et al. [11]. Because x and y are usually small in short sequences, the computation time needed for motif analysis of short sequences is not prohibitive.



Sequence and Spatial Motif Discovery in Short Sequence Fragments, Fig. 3 Example of a multi-residue sequence pattern as described in the text. This pattern contains five specified residues in a span of ten residues. Here, X_0, X_1, X_2, X_3, and X_4 are specified amino acid types, and the corresponding k values are counted as the distance from the first position of the sequence (i.e., the position occupied by X_0). Thus, $k_1 = 2$, $k_2 = 3$, $k_3 = 6$, and $k_4 = 9$. All other residues (in *white*) are unspecified and may be any amino acid type. This pattern is written as $(X_0, X_1, X_2, X_3, X_4 \mid 2, 3, 6, 9)$



Sequence and Spatial Motif Discovery in Short Sequence Fragments, Fig. 4 Difference between (**a**) a *permutation* null model for sequence motif analysis and (**b**) a *position-dependent* null model. In both cases, only residues of the same shade are permuted with each other. In (**a**), residues are permuted only within each sequence individually, while in (**b**), residues are permuted across sequences but only within their specified position t

Propensity of Multi-residue Sequence Motifs

We now discuss the expected number $\mathbb{E}[f(X_0, X_1, X_2, \ldots, X_n | k_1, k_2, \ldots, k_n)]$ of a specific pattern containing $n+1$ residues placed in a contiguous subsequence of $k_n + 1$ residues ($k_n \geq n$). Here X_i is the residue type of the i-th fixed residue in the pattern and k_i is the position of this residue from the 0-th residue ($k_0 = 0$). Positions not specified by k_i can be any residue type. For example, the pattern $(A, L, Y | 2, 4)$ is written as AL2Y4 and represents AxLxY. A graphic example is shown in Fig. 3. Many examples of these multi-residue sequence motifs in proteins have been discovered, including the GxGxxG NADH binding motif [1] and the RSxSxP 14-3-3 binding motif [14].

The expected value can be calculated as:

$$\mathbb{E}[f(X_0, X_1, X_2, \ldots, X_n | k_1, k_2, \ldots, k_n)] = (l - k_n) \frac{\prod_{i=0}^{n} [x_i - \#(\mathbb{I}(X_i))]}{\frac{l!}{(l-n-1)!}}, \quad (3)$$

where x_i is the number of residues of type X_i, l is the length of the sequence, and $\#(\mathbb{I}(X_i))$ is the number of times residue type X_i appears in the "subpattern" $\{X_0, X_1, X_2, \ldots, X_{i-1}\}$.

Remark

The above discussions are for determining motifs in a single short sequence or sequence pair. This can be extended so analysis can be performed on a dataset of multiple short sequences to attain sufficient statistical significance. This has the advantage of capturing within-sequence relationships on a scale large enough to obtain reliable p-values. Details can be found in [6].

Spatial Motifs by Positional Null Model

When there are significant biases in residue preferences for certain positions in a sequence known a priori, e.g., the enrichment of aromatic residues at either end of a transmembrane α-helix or β-strand [12], these single-residue biases may confound two-residue propensities. The *positional null model* should be used for motif detection in such cases [6]. Instead of permuting residues across all positions within individual sequences, we permute residues across all sequences in a dataset within specific positions (Fig. 4).

Expectation and Significance of Interacting Residue Pairs

We allocate residues into regions, which do not overlap. Regions may have different lengths along the sequences. Interacting regions within a sequence pair are assumed to have equal length. If a residue in region r interacts with a residue in region s on a spatially adjacent sequence fragment, all residues in region r in the dataset

must only interact with residues in region s. For example, for interacting antiparallel β-strands, we divide each strand into three regions, the N-terminal, central core, and C-terminal regions, and all interacting strand pairs into two spatial pair types, N-terminal with C-terminal and core with core. We require that no core residue interact with an N-terminal or C-terminal residue.

The null model for position-dependent spatial motifs differs depending on whether paired residues are from the same region ($r = s$) or different regions ($r \neq s$), and whether the residue types in the pair are the same ($X = Y$) or different ($X \neq Y$).

1. **When $r = s$ and $X = Y$.** The expected value of X-X pairs in region r is

$$\mathbb{E}(X, X|rr) = \frac{\binom{x_r}{2}}{\binom{n_r}{2}} \cdot \frac{n_r}{2} = \frac{x_r(x_r - 1)}{2(n_r - 1)},$$

where n_r is the number of residues in region r. The probability $\mathbb{P}_{XX|rr}(i)$ of i X-X interacting pairs in region r in the dataset for p-values calculation calculated is

$$\mathbb{P}_{XX|rr}(i) = \frac{M(\frac{n_r}{2}, i, x_r - 2i) \cdot 2^{x_r - 2i}}{\binom{n_r}{x_r}},$$

where the 3-element multinomial function $M(a, b, c)$ is as defined before.

2. **When $r = s$ and $X \neq Y$.** The expected value when $X \neq Y$ is

$$\mathbb{E}(XY|rr) = \frac{x_r y_r}{\binom{n_r}{2}} \cdot \frac{n_r}{2} = \frac{x_r y_r}{n_r - 1}.$$

The probability $\mathbb{P}(i, j, k)$ of each combination of i, j, and k pairs of type X-Y, X-X, and Y-Y interactions, respectively, is

$$\mathbb{P}(i, j, k) = \frac{M(\frac{n_r}{2}, i, j, k, x_r - i - 2j, y_r - i - 2k) \cdot 2^{x_r + y_r - i - 2j - 2k}}{M(n_r, x_r, y_r)},$$

where the 6-variable multinomial function $M(a,b,c,d,e,f) \equiv \frac{a!}{b!c!d!e!f!(a-b-c-d-e-f)!}$. The probability $\mathbb{P}_{XY|rr}(i)$ of i X-Y pairs in the dataset is then

$$\mathbb{P}_{XY|rr}(i) = \sum_{j=0}^{\frac{x_r - i}{2}} \sum_{k=0}^{\frac{y_r - i}{2}} \mathbb{P}(i, j, k).$$

3. **When $r \neq s$.** We distinguish X_r, a residue of type X occurring in region r in one sequence, and X_s, a residue of type X occurring in region s in the other sequence. Thus, an X-Y pair, which we define as an $X_r - Y_s$ pair, is different from a Y-X pair, which is $Y_r - X_s$. Because there is a one-to-one correspondence between residues in region r and region s, $n_r = n_s$ is the total number of $r - s$ pairs.

In order for exactly i X-Y pairs to occur, i X_r residues must be drawn from a possible x_r residues of type X to match i Y_s residues drawn from a possible y_s residues of type Y. This can be modeled with a simple hypergeometric distribution. The expected value can be calculated as

$$\mathbb{E}(XY|rs) = \frac{x_r y_s}{n_r}.$$

The $\mathbb{P}_{XY|rs}(i)$ of i X-Y pairs is

$$\mathbb{P}_{XY|rs}(i) = \frac{\binom{x_r}{i}\binom{n_r - x_r}{y_s - i}}{\binom{n_r}{y_s}}.$$

Expectation and Significance of Sequence Motifs

We define the *positional residue frequency* x_t as the number of residues of type X occupying the t-th position of all sequences in the dataset. If sequences of different lengths are represented in the dataset, it is necessary to normalize t to be within an appropriate range $[1, l]$, to approximate

an average or predetermined sequence length of l:

$$t = \lceil \frac{l(t_{\text{obs}} - 0.5)}{l_{\text{obs}}} \rceil,$$

where $t_{\text{obs}} \in \{1, 2, 3, \cdots, l_{\text{obs}}\}$ is the actual position of the residue within its sequence, l_{obs} is the actual length of the sequence, $\lceil x \rceil$ represents the ceiling function, equal to the lowest integer greater than or equal to x, and the 0.5 factor is a correction for continuity to round to the next integer. This ensures that $1 \leq t \leq l$, no residues are removed from the model by truncation, and each position t will be represented by nearly the same number of residues.

For sequence motif, we use the model of permutation within each position in a sequence *with replacement* across all sequences. Although all other null models in this study rely on permutation without replacement, this model is based on datasets of multiple sequences instead of individual sequences, and the approximation of sampling without replacement will not be problematic once a sufficiently large sample of sequences is assembled.

1. XYk **motif at position** t**.** When $t \leq 1-k$, the probability of an XYk pattern at position t is

$$\mathbb{P}(X, Y|k, t) = \frac{x_t}{n_t} \cdot \frac{y_{t+k}}{n_{t+k}},$$

where x_t is the number of residues of type X in position t on all sequences, y_t is the number of residues of type Y in position t, and n_t is the number of all residues of all types in position t. This null model can be represented as a binomial distribution.

The expected frequency of XYk patterns at position t is

$$\mathbb{E}[f(X, Y|k, t)] = n_t \cdot \mathbb{P}(X, Y|k, t).$$

The probability of i XYk patterns at position t in the dataset is

$$\mathbb{P}_{XYk|t}(i) = \binom{n_t}{i} \mathbb{P}(X, Y|k, t)^i [1 - \mathbb{P}(X, Y|k, t)]^{n_t - i}.$$

Note that the probability that an XYk pattern appears at position t is 0 if $t > l - k$, as an XYk pattern would span across the end of a sequence of length l.

2. XYk **motif at any arbitrary position.** To calculate the dataset-wide probability of an XYk pattern at any arbitrary position of the sequence, we average $\mathbb{P}(X, Y|k, t)$ over all $l - k$ possible positions:

$$\mathbb{P}(X, Y|k) = \frac{1}{l-k} \sum_{t=1}^{l-k} \mathbb{P}(X, Y|k, t).$$

This can similarly be represented as a binomial distribution with probability distribution function: $\mathbb{P}_{XYk}(i) = \binom{n_k}{i} \mathbb{P}(X, Y|k)^i [1 - \mathbb{P}(X, Y|k)]^{n_k - i}$, where n_k is the number of all pairs of all residue types k residues apart in the dataset. The expected value is

$$\mathbb{E}[f(X, Y|k)] = n_k \cdot \mathbb{P}(X, Y|k).$$

Unlike the situation where only one position t is concerned, this distribution represents the sum of dependent Bernoulli variables. Methods of accounting for this dependence can be found in Robin et al. [10].

Applications

Several spatial and sequence motifs have been discovered using the approach discussed here [5–7]. The estimated propensities have also been used to develop empirical potential function for prediction of oligomerization stated [8], protein-protein interaction interfaces [3, 8], engineering of thermal resistance [2], and in predicting structures of β-barrel membrane proteins [9].

Open Problems

General analytical formulas for calculating probabilities of two-residue and multi-residue motifs under the permutation model are unknown.

Recommended Reading

1. Baker PJ, Britton KL, Rice DW, Rob A, Stillman TJ (1992) Structural consequences of sequence patterns in the fingerprint region of the nucleotide binding fold. Implications for nucleotide specificity. J Mol Biol 228(2):662–671
2. Gessmann D, Mager F, Naveed H, Arnold T, Weirich S, Linke D, Liang J, Nussberger S (2011) Improving the resistance of a eukaryotic beta-barrel protein to thermal and chemical perturbations. J Mol Biol 413(1):150–161. doi:10.1016/j.jmb.2011.07.054
3. Geula S, Naveed H, Liang J, Shoshan-Barmatz V (2012) Structure-based analysis of VDAC1 protein: defining oligomer contact sites. J Biol Chem 287(3):2179–2190
4. Hart R, Royyuru A, Stolovitzky G, Califano A (2000) Systematic and fully automated identification of protein sequence patterns. J Comput Biol 7(3–4):585–600
5. Jackups R Jr, Liang J (2005) Interstrand pairing patterns in beta-barrel membrane proteins: the positive-outside rule, aromatic rescue, and strand registration prediction. J Mol Biol 354(4):979–993
6. Jackups R Jr, Liang J (2010) Combinatorial analysis for sequence and spatial motif discovery in short sequence fragments. IEEE/ACM Trans Comput Biol Bioinform 7(3):524–536
7. Jackups R Jr, Cheng S, Liang J (2006) Sequence motifs and antimotifs in beta-barrel membrane proteins from a genome-wide analysis: the ala-tyr dichotomy and chaperone binding motifs. J Mol Biol 363(2):611–623
8. Naveed H, Jackups R Jr, Liang J (2009) Predicting weakly stable regions, oligomerization state, and protein-protein interfaces in transmembrane domains of outer membrane proteins. Proc Natl Acad Sci USA 106(31):12735–12740. doi:10.1073/pnas.0902169106
9. Naveed H, Xu Y, Jackups R, Liang J (2012) Predicting three-dimensional structures of transmembrane domains of β-barrel membrane proteins. J Am Chem Soc 134(3):1775–1781
10. Robin S, Rodophe F, Schbath S (2005) DNA, words and models: statistics of exceptional words. Cambridge University Press, Cambridge/New York
11. Senes A, Gerstein M, Engelman DM (2000) Statistical analysis of amino acid patterns in transmembrane helices: the GxxxG motif occurs frequently and in association with β-branched residues at neighboring positions. J Mol Biol 296:921–936
12. Wimley WC (2002) Toward genomic identification of β-barrel membrane proteins: composition and architecture of known structures. Protein Sci 11:301–312
13. Wouters MA, Curmi PM (1995) An analysis of side chain interactions and pair correlations within antiparallel β-sheets: the differences between backbone hydrogen-bonded and non-hydrogen-bonded residue pairs. Proteins 22:119–131
14. Yaffe MB, Rittinger K, Volinia S, Caron PR, Aitken A, Leffers H, Gamblin SJ, Smerdon SJ, Cantley LC (1997) The structural basis for 14-3-3:phosphopeptide binding specificity. Cell 91(7):961–971

Sequential Circuit Technology Mapping

Peichen Pan
Xilinx, Inc., San Jose, CA, USA

Keywords

Boolean network; EDA; FPGA; Retiming; Technology mapping; VLSI CAD

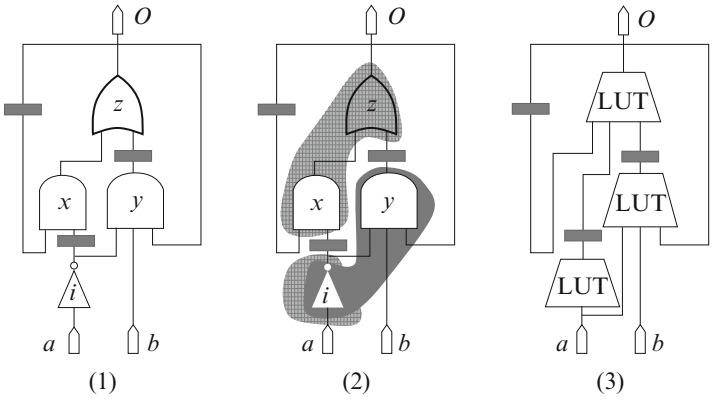
Years and Authors of Summarized Original Work

1996; Pan, Liu
1998; Pan, Liu
1998; Pan, Lin

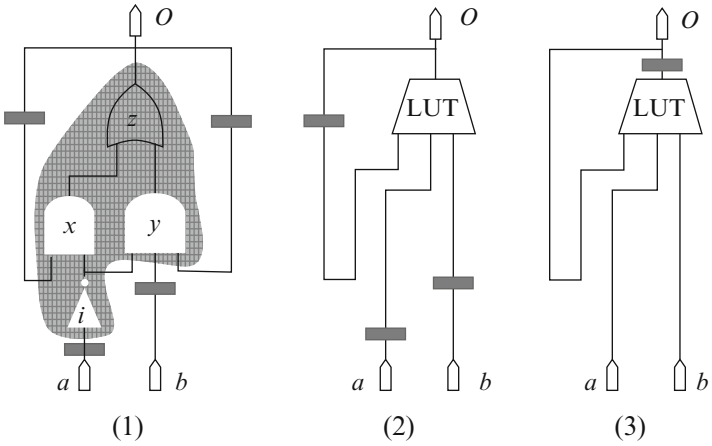
Problem Definition

One of the key steps in a VLSI design flow is technology mapping that converts a Boolean network of technology-independent logic gates and D-flipflops (FFs) into an equivalent one comprised of cells from a technology library [1, 4]. Technology mapping can be formulated as a covering problem where logic gates are covered by cells from the technology library. For ease of discussion, it is assumed that the cell library contains only one cell, a K-input lookup table (K-LUT) with one unit of delay. A K-LUT can implement any Boolean function with up to K inputs as is the case in field-programmable gate arrays (FPGAs) [1, 3].

Figure 1 shows an example of technology mapping. The original network in (1) with three FFs and four gates is covered by three 3-input cones as indicated in (2). The corresponding



Sequential Circuit Technology Mapping, Fig. 1 Technology mapping: (**1**) original network, (**2**) covering, (**3**) mapping solution



Sequential Circuit Technology Mapping, Fig. 2 Retiming and mapping: (**1**) retiming and covering, (**2**) mapping solution, (**3**) retimed solution

mapping solution using 3-LUTs is shown in (3). Note that gate i is covered by two cones so it will be replicated. The mapping solution has a *cycle time* (or *clock period*) of two units, which is the maximum number of LUTs on all paths without FFs.

Retiming relocates FFs in a network by moving FFs across logic nodes backward or forward [5]. Retiming does not alter the functionality of a network. Figure 2 (1) shows the network obtained from the one in Fig. 1 (1) by moving the FFs at the output of gates y and i to their inputs. It can now be covered with just one 3-input cone as indicated in (1). The corresponding mapping solution shown in (2) is better in both cycle time and area than the one in Fig. 1 (3) obtained without retiming.

A *K-bounded* network is one in which each gate has at most K inputs. The sequential circuit technology mapping problem can be defined as follows: *Given a K-bounded Boolean network N and a target cycle time ϕ, find a mapping solution with a cycle time of ϕ, assuming FFs can be relocated using retiming*.

Key Results

The first polynomial time algorithm for the problem was proposed in [9, 10]. An improved algorithm was proposed in [2] to reduce runtime. Both algorithms are based on min-cost flow computation.

In [8], an efficient algorithm was proposed to take advantage of the fact that K is a small integer usually between 3 and 6. The algorithm is based

Sequential Circuit Technology Mapping, Fig. 3 Cut enumeration procedure

FindAllCuts(N, K)

foreach node v in N **do** C(v) $\Leftarrow \{\{v^0\}\}$

while (*new cuts discovered*) **do**

foreach node v in N **do** $C(v) \Leftarrow merge(C(u_1),....,C(u_t))$

on enumerating all K-input cones for each gate and will be described next.

Cut Enumeration

A Boolean network can be represented as an edge-weighted directed graph where the nodes denote logic gates and primary inputs/outputs. There is a directed edge (u, v) with weight d if u, after going through d FFs, drives v.

A *cone* for a node can be captured by a *cut* consisting of inputs to the cone. An element in a cut for v consists of the driving node u and the total weight d on the paths from u to v, denoted by u^d. If u can reach v on several paths with different FF counts, u will appear in the cut multiple times with different ds. For the cone for z in Fig. 2 (2), the corresponding cut is $\{z^1, a^1, b^1\}$. A cut of size K is called a K-cut.

Let (u_i, v), where $i = 1, \ldots, t$, be all input edges to v in N. Further assume the weight of (u_i, v) is d_i and $C(u_i)$ is a set of K-cuts for u_i. Let $merge(C(u_1), \ldots, C(u_t))$ denote the following set operation:

$$\{\{v^0\}\} \cup \{c_1^{d_1} \cup \cdots \cup c_t^{d_t} | c_1 \in (u_1), \ldots, c_t \in C(u_t), |c_1^{d_1} \cup \cdots \cup c_t^{d_t}| \leq K\}$$

where $c_i^{d_i} = \{u^{d+d_i} | u^d \in c_i\}$ for $i = 1, \ldots, t$. It is obvious that $merge(C(u_1), \ldots, C(u_t))$ is a set of K-cuts for v.

If the network N does not contain cycles, the K-cuts of all nodes can be determined using the merge operation in a topological order starting from the PIs. For general networks, Fig. 3 outlines the iterative cut computation procedure proposed in [8].

Figure 4 depicts the iterations in enumerating the 3-cuts for the network in Fig. 1 (1) where cuts are merged in the order i, x, y, z, and o. At the beginning, every node has a trivial cut formed by itself (Row 0). Row 1 shows the new cuts discovered in the first iteration. In second iteration, two more cuts are discovered (for x). After that, further merging does not yield any new cut and the procedure stops.

Lemma 1 *After at most Kn iterations, the cut enumeration procedure will find all the K-cuts for every node in N.*

Techniques have been proposed to speed up the procedure [8]. For practical networks, the cut enumeration procedure typically converges in just a few iterations.

Label Computation

After obtaining all K-cuts, the cuts are evaluated based on sequential arrival times (or l-values), which is an extension of traditional arrival times, to consider the effect of retiming [7, 9].

The labeling procedure tries to find a label for each node as outlined in Fig. 5, where w_v denotes the weight of the shortest paths from PIs to node v.

Figure 6 shows the iterations for label computation for the network in Fig. 1 (1), assuming that the target cycle time $\phi = 1$ and the nodes are evaluated in the order of i, x, y, z, and o. In the table, the current label as well as a corresponding cut for each node is listed. In this example, after the first iteration, none of the labels will change and the procedure stops.

It can be shown that the labeling procedure will stop after at most $n(n-1)$ iterations [10]. The following lemma relates labels to mapping:

Iter	*a*	*B*	*I*	*x*	*y*	*z*	*o*
0	$\{a^0\}$	$\{b^0\}$	$\{i^0\}$	$\{x^0\}$	$\{y^0\}$	$\{z^0\}$	$\{o^0\}$
1			$\{a^0\}$	$\{i^1,z^1\}$ $\{a^1,z^1\}$	$\{i^0,b^0,z^0\}$ $\{a^0,b^0,z^0\}$	$\{x^0,y^1\}$ $\{i^1,z^1,b^1\}$ $\{a^1,z^1,b^1\}$ $\{i^1,z^1,y^1\}$ $\{a^1,z^1,y^1\}$	$\{z^0\}$
2				$\{i^1,x^1,y^2\}$ $\{a^1,x^1,y^2\}$			

Sequential Circuit Technology Mapping, Fig. 4 Cut enumeration example

Sequential Circuit Technology Mapping, Fig. 5 Label computation procedure

FindMinLabels(N **)**

foreach node v in N **do** $l(v) \Leftarrow -w_v \cdot \phi$

while (*there are label updates*) **do**

foreach node v in N **do**

$l(v) \Leftarrow \min_{c \in C(v)} \{\max\{l(u) - d \cdot \phi + 1 \mid u^d \in c\}\}$

if v is a primary output and $l(v) > \phi$, **return** failure

return success

iter	*a*	*B*	*I*	*x*	*y*	*z*	*o*
0	$\{a^0\}$:0	$\{b^0\}$:0	$\{i^0\}$:0	$\{x^0\}$:-1	$\{y^0\}$:0	$\{z^0\}$:-1	$\{o^0\}$:-1
1			$\{a^0\}$:1	$\{a^1,z^1\}$:0	$\{a^0,b^0,z^0\}$:1	$\{a^1,z^1,b^1\}$:0	$\{z^0\}$:0

Sequential Circuit Technology Mapping, Fig. 6 Label computation example

Lemma 2 *N has a mapping solution with cycle time* ϕ iff the labeling procedure returns "success."

Mapping Solution Generation

Once the labels for all nodes are computed successfully, a mapping solution can be constructed starting from primary outputs. At each node v, the procedure selects the cut that realizes the label of the node and then moves on to select a cut for u if u^d is in the cut selected for v. On the edge from u to v, d FFs are inserted. For the network in Fig. 1 (1), the mapping solution generated based on the labels found in Fig. 6 is exactly the one in Fig. 2 (2).

To obtain a mapping solution with the target cycle time ϕ, v will be retimed by a value of $\lceil l(v)/\phi \rceil - 1$. For the network in Fig. 1 (1), the final mapping solution after retiming is shown in Fig. 2 (3) which has a cycle time of 1.

Applications

The algorithm can be used to map a technology-independent Boolean network to a network consisting of cells from a target technology library. The concepts and framework are generally enough to be adapted to study other circuit optimizations such as sequential circuit clustering and sequential circuit restructuring [6].

Cross-References

- ▸ Circuit Retiming
- ▸ Circuit Retiming: An Incremental Approach
- ▸ FPGA Technology Mapping
- ▸ Technology Mapping

Recommended Reading

1. Cong J, Ding Y (1994) FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. IEEE Trans Comput Aided Des Integr Circuits Syst 13(1):1–12
2. Cong J, Wu C (1997) FPGA synthesis with retiming and pipelining for clock period minimization of sequential circuits. In: ACM/IEEE design automation conference, Anaheim
3. Cong J, Wu C, Ding Y (1999) Cut ranking and pruning: enabling a general and efficient FPGA mapping solution. In: ACM international symposium on field-programmable gate arrays, Monterey
4. Keutzer K (1987) DAGON: technology binding and local optimization by DAG matching. In: ACM/IEEE design automation conference, Miami Beach
5. Leiserson CE, Saxe JB (1991) Retiming synchronous circuitry. Algorithmica 6:5–35
6. Mishchenko A, Chatterjee S, Brayton R, Pan P (2006) Integrating logic synthesis, technology mapping, and retiming. ERL technical report, EECS Department, UC Berkeley
7. Pan P (1997) Continuous retiming algorithms and applications. In: IEEE international conference on computer design, Austin
8. Pan P, Lin CC (1998) A new retiming-based technology mapping algorithm for LUT-based FPGAs. In: ACM international symposium on field-programmable gate arrays, Monterey
9. Pan P, Liu CL (1996) Optimal clock period FPGA technology mapping for sequential circuits. In: ACM/IEEE design automation conference, Las Vegas
10. Pan P, Liu CL (1998) Optimal clock period FPGA technology mapping for sequential circuits. ACM Trans Des Autom Electron Syst 3(3):437–462

Set Agreement

Michel Raynal
Institut Universitaire de France and IRISA, Université de Rennes, Rennes, France

Keywords

Distributed coordination

Years and Authors of Summarized Original Work

1993; Chaudhuri

Problem Definition

Short History

The k-set agreement problem is a paradigm of coordination problems. Defined in the setting of systems made up of processes prone to failures, it is a simple generalization of the consensus problem (that corresponds to the case $k = 1$). That problem was introduced in 1993 by Chaudhuri [2] to investigate how the number of choices (k) allowed for the processes is related to the maximum number of processes that can crash. (After it has crashed, a process executes no more steps: a crash is a premature halting.)

Definition

Let S be a system made up of n processes where up to t can crash and where each process has an input value (called a *proposed* value). The problem is defined by the three following properties (i.e., any algorithm that solves that problem has to satisfy these properties):

1. Termination. Every nonfaulty process decides a value.
2. Validity. A decided value is a proposed value.
3. Agreement. At most k different values are decided.

The Trivial Case

It is easy to see that this problem can be trivially solved if the upper bound on the number of process failures t is smaller than the allowed number of choices k, also called the *coordination degree*. (The trivial solution consists in having $t + 1$ predetermined processes that send their proposed values to all the processes, and a process deciding the first value it ever receives.) So, $k \leq$ t is implicitly assumed in the following.

Key Results

Key Results in Synchronous Systems

The Synchronous Model

In this computation model, each execution consists of a sequence of rounds. These are identified by the successive integers $1, 2$, etc. For the processes, the current round number appears as a global variable whose global progress entails their own local progress.

During a round, a process first broadcasts a message, then receives messages, and finally executes local computation. The fundamental synchrony property the a synchronous system provides the processes with is the following: a message sent during a round r is received by its destination process during the very same round r. If during a round, a process crashes while sending a message, an arbitrary subset (not known in advance) of the processes receive that message.

Main Results

The k-set agreement problem can always be solved in a synchronous system. The main result is for the minimal number of rounds (R_t) that are needed for the nonfaulty processes to decide in the worst-case scenario (this scenario is when exactly k processes crash in each round). It was shown in [3] that $R_t = \lfloor \frac{t}{k} \rfloor + 1$. A very simple algorithm that meets this lower bound is described in Fig. 1.

Although failures do occur, they are rare in practice. Let f denote the number of processes that crash in a given run, $0 \leq f \leq t$. We are interested in synchronous algorithms that terminate in at most R_t rounds when t processes crash in the current run, but that allow the nonfaulty processes to decide in far fewer rounds when there are few failures. Such algorithms are called *early-deciding* algorithms. It was shown in [4] that, in the presence of f process crashes, any early-deciding k-set agreement algorithm has runs in which no process decides before the round $R_f = \min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$. This lower bound shows an inherent tradeoff linking the coordination degree k, the maximum number of process failures t, the actual number of process failures f, and the best time complexity that can be achieved. Early-deciding k-set agreement algorithms for the synchronous model can be found in [4, 12].

Other Failure Models

In the send omission failure model, a process is faulty if it crashes or forgets to send messages. In the general omission failure model, a process is faulty if it crashes, forgets to send messages, or forgets to receive messages. (A send omission failure models the failure of an output buffer, while a receive omission failure models the failure of an input buffer.) These failure models were introduced in [11].

The notion of *strong* termination for set agreement problems was introduced in [13]. Intuitively, that property requires that as many processes as possible decide. Let a *good* process be a process that neither crashes nor commits receive omission failures. A set agreement algorithm is strongly terminating if it forces all the good processes to decide. (Only the processes that crash during the execution of the algorithm, or that do not receive enough messages, can be prevented from deciding.)

An early-deciding k-set agreement algorithm for the general omission failure model was described in [13]. That algorithm, which requires $t < n/2$, directs a good process to decide and stop in at most $R_f = \min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$ rounds. Moreover, a process that is not a good

Set Agreement, Fig. 1
A simple k-set agreement synchronous algorithm (code for p_i)

```
Function k-set_agreement (v_i)
(1) est_i ← v_i;
(2) when r = 1, 2, ..., ⌊t/k⌋ + 1 do % r: round number %
(3) begin_round
(4)       send (est_i) to all; % including p_i itself %
(5)       est_i ← min({est_j values received during
                                    the current round r});
(6) end_round;
(7) return (est_i)
```

process executes at most $R_f(not\ good)$) $\min(\lceil \frac{f}{k} \rceil + 2, \lfloor \frac{t}{k} \rfloor + 1)$ rounds.

As R_f is a lower bound for the number of rounds in the crash failure model, the previous algorithm shows that R_f is also a lower bound for the nonfaulty processes to decide in the more severe general omission failure model. Proving that $R_f(not\ good)$ is an upper bound for the number of rounds that a nongood process has to execute remains an open problem.

It was shown in [13] that, for a given coordination degree k, $t < \frac{k}{k+1}n$ is an upper bound on the number of process failures when one wants to solve the k-set agreement problem in a synchronous system prone to process general omission failures. A k-set agreement algorithm that meets this bound was described in [13]. That algorithm requires the processes execute $R = t + 2 - k$ rounds to decide. Proving (or disproving) that R is a lower bound when $t < \frac{k}{k+1}n$ is an open problem. Designing an early-deciding k-set agreement algorithm for $t < \frac{k}{k+1}n$ and $k > 1$ is another problem that remains open.

Key Results in Asynchronous Systems

Impossibility

A fundamental result of distributed computing is the impossibility to design a deterministic algorithm that solves the k-set agreement problem in asynchronous systems when $k \leq$ t [1, 7, 15]. Compared with the impossibility of solving asynchronous consensus despite one process crash, that impossibility is based on deep combinatorial arguments. This impossibility has opened new research directions for the connection between distributed computing and topology. This topology approach has allowed the discovery of links relating asynchronous k-set agreement with other distributed computing problems such as the *renaming* problem [5].

Circumventing the Impossibility

Several approaches have been investigated to circumvent the previous impossibility. These approaches are the same as those that have been used to circumvent the impossibility of asynchronous consensus despite process crashes.

One approach consists in replacing the "deterministic algorithm" by a "randomized algorithm." In that case, the termination property becomes "the probability for a correct process to decide tends to 1 when the number of rounds tends to $+\infty$." That approach was investigated in [9].

Another approach that has been proposed is based on failure detectors. Roughly speaking, a failure detector provides each process with a list of processes suspected to have crashed. As an example, the class of failure detectors denoted $\diamond\mathcal{S}_x$ includes all the failure detectors such that, after some finite (but unknown) time, (1) any list contains the crashed processes and (2) there is a set Q of x processes such that Q contains one correct process and that correct process is no longer suspected by the processes of Q (let us observe that correct processes can be suspected intermittently or even forever). Tight bounds for the k-set agreement problem in asynchronous systems equipped with such failure detectors, conjectured in [9], were proved in [6]. More precisely, such

a failure detector class allows the k-set agreement problem to be solved for $k \geq t - x + 2$ [9], and cannot solve it when $k < t - x + 2$ [6].

Another approach that has been investigated is the combination of failure detectors and conditions [8]. A condition is a set of input vectors, and each input vector has one entry per process. The entries of the input vector associated with a run contain the values proposed by the processes in that run. Basically, such an approach guarantees that the nonfaulty processes always decide when the actual input vector belongs to the condition the k-set algorithm has been instantiated with.

Applications

The set agreement problem was introduced to study how the number of failures and the synchronization degree are related in an asynchronous system; hence, it is mainly a theoretical problem. That problem is used as a canonical problem when one is interested in asynchronous computability in the presence of failures. Nevertheless, one can imagine practical problems the solutions of which are based on the set agreement problem (e.g., allocating a small shareable resources – such as broadcast frequencies – in a network).

Cross-References

- ▸ Asynchronous Consensus Impossibility
- ▸ Failure Detectors
- ▸ Renaming
- ▸ Topology Approach in Distributed Computing

Recommended Reading

1. Borowsky E, Gafni E (1993) Generalized FLP impossibility results for t-resilient asynchronous computations. In: Proceedings of the 25th ACM symposium on theory of computation, California, pp 91–100
2. Chaudhuri S (1993) More choices allow more faults: set consensus problems in totally asynchronous systems. Inf Comput 105:132–158
3. Chaudhuri S, Herlihy M, Lynch N, Tuttle M (2000) Tight bounds for k set agreement. J ACM 47(5):912–943
4. Gafni E, Guerraoui R, Pochon B (2005) From a static impossibility to an adaptive lower bound: the complexity of early deciding set agreement. In: Proceedings of the 37th ACM symposium on theory of computing (STOC 2005). ACM, New York, pp 714–722
5. Gafni E, Rajsbaum S, Herlihy M (2006) Subconsensus tasks: renaming is weaker than set agreement. In: Proceedings of the 20th international symposium on distributed computing (DISC'06). LNCS, vol 4167. Springer, Berlin, pp 329–338
6. Herlihy MP, Penso LD (2005) Tight bounds for k set agreement with limited scope accuracy failure detectors. Distrib Comput 18(2):157–166
7. Herlihy MP, Shavit N (1999) The topological structure of asynchronous computability. J ACM 46(6):858–923
8. Mostefaoui A, Rajsbaum S, Raynal M (2005) The combined power of conditions and failure detectors to solve asynchronous set agreement. In: Proceedings of the 24th ACM symposium on principles of distributed computing (PODC'05). ACM, New York, pp 179–188
9. Mostefaoui A, Raynal M (2000) k set agreement with limited scope accuracy failure detectors. In: Proceedings of the 19th ACM symposium on principles of distributed computing. ACM, New York, pp 143–152
10. Mostefaoui A, Raynal M (2001) Randomized set agreement. In: Proceedings of the 13th ACM symposium on parallel algorithms and architectures (SPAA'01), Hersonissos (Crete). ACM, New York, pp 291–297
11. Perry KJ, Toueg S (1986) Distributed agreement in the presence of processor and communication faults. IEEE Trans Softw Eng SE-12(3):477–482
12. Raipin Parvedy P, Raynal M, Travers C (2005) Early-stopping k-set agreement in synchronous systems prone to any number of process crashes. In: Proceedings of the 8th international conference on parallel computing technologies (PaCT'05). LNCS, vol 3606. Springer, Berlin, pp 49–58
13. Raipin Parvedy P, Raynal M, Travers C (2006) Strongly-terminating early-stopping k-set agreement in synchronous systems with general omission failures. In: Proceedings of the 13th colloquium on structural information and communication complexity (SIROCCO'06). LNCS, vol 4056. Springer, Berlin, pp 182–196
14. Raynal M, Travers C (2006) Synchronous set agreement: a concise guided tour (including a new algorithm and a list of open problems). In: Proceedings of the 12th international IEEE pacific rim dependable computing symposium (PRDC'2006). IEEE Computer Society, Los Alamitos, pp 267–274
15. Saks M, Zaharoglou F (2000) Wait-free k-set agreement is impossible: the topology of public knowledge. SIAM J Comput 29(5):1449–1483

Set Cover with Almost Consecutive Ones

Michael Dom
Department of Mathematics and Computer Science, University of Jena, Jena, Germany

Keywords

Hitting set

Years and Authors of Summarized Original Work

2004; Mecke, Wagner

Problem Definition

The SET COVER problem has as input a set R of m items, a set C of n subsets of R and a weight function $w\colon C \to \mathbb{Q}$. The task is to choose a subset $C' \subseteq C$ of minimum weight whose union contains all items of R.

The sets R and C can be represented by an $m \times n$ binary matrix A that consists of a row for every item in R and a column for every subset of R in C, where an entry $a_{i,j}$ is 1 iff the ith item in R is part of the jth subset in C. Therefore, the SET COVER problem can be formulated as follows.

> **Input**: An $m \times n$ binary matrix A and a weight function w on the columns of A.
> **Task:** Select some columns of A with minimum weight such that the submatrix A' of A that is induced by these columns has at least one 1 in every row.

While SET COVER is NP-hard in general [4], it can be solved in polynomial time on instances whose columns can be permuted in such a way that in every row the ones appear consecutively, that is, on instances that have the *consecutive ones property* (*C1P*). (The C1P can be defined symmetrically for columns; this article focuses on rows. SET COVER on instances with the C1P can be solved in polynomial time, e.g., with a linear programming approach, because the corresponding coefficient matrices are totally unimodular (see [9]).

Motivated by problems arising from railway optimization, Mecke and Wagner [7] consider the case of SET COVER instances that have "almost the C1P". Having almost the C1P means that the corresponding matrices are similar to matrices that have been generated by starting with a matrix that has the C1P and replacing randomly a certain percentage of the 1's by 0's [7]. For Ruf and Schöbel [8], in contrast, having almost the C1P means that the average number of blocks of consecutive 1's per row is much smaller than the number of columns of the matrix. This entry will also mention some of their results.

Notation

Given an instance (A, w) of SET COVER, let R denote the row set of A and C its column set. A column c_j *covers* a row r_i, denoted by $r_i \in c_j$, if $a_{i,j} = 1$.

A binary matrix has the *strong C1P* if (without any column permutation) the 1's appear consecutively in every row. A *block of consecutive 1's* is a maximal sequence of consecutive 1's in a row. It is possible to determine in linear time if a matrix has the C1P, and if so, to compute a column permutation that yields the strong C1P [2, 3, 6]. However, note that it is NP-hard to permute the columns of a binary matrix such that the number of blocks of consecutive 1's in the resulting matrix is minimized [1, 4, 5].

A *data reduction rule* transforms in polynomial time a given instance I of an optimization problem into an instance I' of the same problem such that $|I'| < |I|$ and the optimal solution for I' has the same value (e.g., weight) as the optimal solution for I. Given a set of data reduction rules, *to reduce* a problem instance means to repeatedly apply the rules until no rule is applicable; the resulting instance is called *reduced*.

Key Results

Data Reduction Rules

For SET COVER there exist well-known data reduction rules:

Row domination rule: If there are two rows $r_{i_1}, r_{i_2} \in R$ with $\forall c \in C: r_{i_1} \in c \text{implies} r_{i_2} \in c$, then r_{i_2} is *dominated* by r_{i_1}. Remove row r_{i_2} from A.

Column domination rule: If there are two columns $c_{j_1}, c_{j_2} \in C$ with $w(c_{j_1}) \geq w(c_{j_2})$ and $\forall r \in R: r \in c_{j_1}$ implies $r \in c_{j_2}$, then c_{j_1} is *dominated* by c_{j_2}. Remove c_{j_1} from A.

In addition to these two rules, a column $c_{j_1} \in C$ can also be dominated by a subset $C' \subseteq C$ of the columns instead of a single column: If there is a subset $C' \subseteq C$ with $w(c_{j_1}) \geq \sum_{c \in C'} w(c)$ and $\forall r \in R: r \in c_{j_1}$ implies $(\exists c \in C': r \in c)$, then remove c_{j_1} from A. Unfortunately, it is NP-hard to find a dominating subset C' for a given set c_{j_1}. Mecke and Wagner [7], therefore, present a restricted variant of this generalized column domination rule.

For every row $r \in R$, let $c_{\min}(r)$ be a column in C that covers r and has minimum weight under this property. For two columns $c_{j_1}, c_{j_2} \in C$, define $X(c_{j_1}, c_{j_2}) := \{c_{\min}(r) \mid r \in c_{j_1} \wedge r \notin c_{j_2}\}$. The new data reduction rule then reads as follows.

Advanced column domination rule: If there are two columns $c_{j_1}, c_{j_2} \in C$ and a row that is covered by both c_{j_1} and c_{j_2}, and if $w(c_{j_1}) \geq w(c_{j_2}) + \sum_{c \in X(c_{j_1}, c_{j_2})} w(c)$, then c_{j_1} is *dominated* by $\{c_{j_2}\} \cup X(c_{j_1}, c_{j_2})$. Remove c_{j_1} from A.

Theorem 1 ([7]) *A matrix A can be reduced in O(Nn) time with respect to the column domination rule, in O(Nm) time with respect to the row domination rule, and in O(Nmn) time with respect to all three data reduction rules described above, when N is the number of 1's in A.*

In the databases used by Ruf and Schöbel [8], matrices are represented by the column indices of the first and last 1's of its blocks of consecutive 1's. For such matrix representations, a fast data reduction rule is presented [8], which eliminates "unnecessary" columns and which, in the implementations, replaces the column domination rule. The new rule is faster than the column domination rule (a matrix can be reduced in $O(mn)$ time with respect to the new rule), but not as powerful: Reducing a matrix A with the new rule can result in a matrix that has more columns than the matrix resulting from reducing A with the column domination rule.

Algorithms

Mecke and Wagner [7] present an algorithm that solves SET COVER by enumerating all feasible solutions.

Given a row r_i of A, a *partial solution for the rows* $r_1, \ldots, r_i$ is a subset $C' \subseteq C$ of the columns of A such that for each row r_j with $j \in \{1, \ldots, i\}$ there is a column in C' that covers row r_j.

The main idea of the algorithm is to find an optimal solution by iterating over the rows of A and updating in every step a data structure S that keeps *all* partial solutions for the rows considered so far. More exactly, in every iteration step the algorithm considers the first row of A and updates the data structure S accordingly. Thereafter, the first row of A is deleted. The following code shows the algorithm.

```
1 Repeat m times: {
2   for every partial solution C' in S that does not
                                cover the first row of A: {
3     for every column c of A that covers the first row
                                of A: {
4       Add {c} ∪ C' to S; }
5 Delete C' from S; }
6 Delete the first row of A; }
```

This straightforward enumerative algorithm could create a set S of exponential size. Therefore, the data reduction rules presented above are used to delete after each iteration step partial solutions that are not needed any more. To this end, a matrix B is associated with the set S, where every row corresponds to a row of A and every column corresponds to a partial solution in S–an entry $b_{i,j}$ of B is 1 iff the jth partial solution of B contains a column of A that covers the row r_i. The algorithm uses the matrix $C := \left(\begin{array}{c|c} A & B \\ \hline 0 \ldots 0 & 1 \ldots 1 \end{array}\right)$, which is updated together with S in every iteration step. (The last row of C allows to distinguish the columns belonging to A from those belonging

to B.) Line 6 of the code shown above is replaced by the following two lines:

6 Delete the first row of the matrix C;
7 Reduce the matrix C and update S accordingly;}

At the end of the algorithm, S contains exactly one solution, and this solution is optimal. Moreover, if the SET COVER instance is nicely structured, the algorithm has polynomial running time:

Theorem 2 ([7]) *If A has the strong C1P, is reduced, and its rows are sorted in lexicographic order, then the algorithm has a running time of $O(M^{3n})$ where M is the maximum number of 1's per row and per column.*

Theorem 3 ([7]) *If the distance between the first and the last 1 in every column is at most k, then at any time throughout the algorithm the number of columns in the matrix B is $O(2^{kn})$, and the running time is $O(2^{2k}\, kmn^2)$.*

Ruf and Schöbel [8] present a branch and bound algorithm for SET COVER instances that have a small average number of blocks of consecutive 1's per row.

The algorithm considers in each step a row r_i of the current matrix (which has been reduced with data reduction rules before) and branches into bl_i cases, where bl_i is the number of blocks of consecutive 1's in r_i. In each case, one block of consecutive 1's in row r_i is selected, and the 1's of all other blocks in this row are replaced by 0's. Thereafter, a lower and an upper bound on the weight of the solution for each resulting instance is computed. If a lower bound differs by a factor of more than $1 + \epsilon$, for a given constant ε, from the best upper bound achieved so far, the corresponding instance is subjected to further branchings. Finally, the best upper bound that was found is returned.

In each branching step, the bl_i instances that are newly generated are "closer" to have the (strong) C1P than the instance from which they descend. If an instance has the C1P, the lower and upper bound can easily be computed by exactly solving the problem. Otherwise, standard heuristics are used.

Applications

SET COVER instances occur e.g., in railway optimization, where the task is to determine where new railway stations should be built. Each row then corresponds to an existing settlement, and each column corresponds to a location on the existing trackage where a railway station could be build. A column c covers a row r, if the settlement corresponding to r lies within a given radius around the location corresponding to c.

If the railway network consisted of one straight line rail track only, the corresponding SET COVER instance would have the C1P; instances arising from real world data are close to have the C1P [7, 8].

Experimental Results

Mecke and Wagner [7] make experiments on real-world instances as described in the Applications section and on instances that have been generated by starting with a matrix that has the C1P and replacing randomly a certain percentage of the 1's by 0's. The real-world data consists of a railway graph with 8,200 nodes and 8,700 edges, and 30,000 settlements. The generated instances consist of 50–50,000 rows with 10–200 1's per row. Up to 20 % of the 1's are replaced by 0's.

In the real-world instances, the data reduction rules decrease the number of 1's to between 1 % and 25 % of the original number of 1's without and to between 0.2 % and 2.5 % with the advanced column reduction rule. In the case of generated instances that have the C1P, the number of 1's is decreased to about 2 % without and to 0.5 % with the advanced column reduction rule. In instances with 20 % perturbation, the number of 1's is decreased to 67 % without and to 20 % with the advanced column reduction rule.

The enumerative algorithm has a running time that is almost linear for real-world instances and most generated instances. Only in the case of generated instances with 20 % perturbation, the running time is quadratic.

Ruf and Schöbel [8] consider three instance types: real-world instances, instances arising

from Steiner triple systems, and randomly generated instances. The latter have a size of 100×100 and contain either 1–5 blocks of consecutive 1's in each row, each one consisting of between one and nine 1's, or they are generated with a probability of 3 % or 5 % for any entry to be 1.

The data reduction rules used by Ruf and Schöbel turn out to be powerful for the real-world instances (reducing the matrix size from about $1{,}100 \times 3{,}100$ to 100×800 in average), whereas for all other instance types the sizes could not be reduced noticeably.

The branch and bound algorithm could solve almost all real-world instances up to optimality within a time of less than a second up to one hour. In all cases where an optimal solution has been found, the first generated subproblem had already provided a lower bound equal to the weight of the optimal solution.

Cross-References

▸ Greedy Set-Cover Algorithms

Recommended Reading

1. Atkins JE, Middendorf M (1996) On physical mapping and the consecutive ones property for sparse matrices. Discret Appl Math 71(1–3):23–40
2. Booth KS, Lueker GS (1976) Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J Comput Syst Sci 13:335–379
3. Fulkerson DR, Gross OA (1965) Incidence matrices and interval graphs. Pac J Math 15(3):835–855
4. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, New York
5. Goldberg PW, Golumbic MC, Kaplan H, Shamir R (1995) Four strikes against physical mapping of DNA. J Comput Biol 2(1):139–152
6. Hsu WL, McConnell RM (2003) PC trees and circular-ones arrangements. Theor Comput Sci 296(1):99–116
7. Mecke S, Wagner D (2004) Solving geometric covering problems by data reduction. In: Proceedings of the 12th annual European symposium on algorithms (ESA '04). LNCS, vol 3221. Springer, Berlin, pp 760–771
8. Ruf N, Schöbel A (2004) Set covering with almost consecutive ones property. Discret Optim 1(2):215–228
9. Schrijver A (1986) Theory of linear and integer programming. Wiley, Chichester

Shadowless Solutions for Fixed-Parameter Tractability of Directed Graphs

Rajesh Chitnis and Mohammad Taghi Hajiaghayi
Department of Computer Science, University of Maryland, College Park, MD, USA

Keywords

Directed graphs; Fixed-parameter tractability; Important Separators; Shadowless solutions; Transversal problems

Years and Authors of Summarized Original Work

2012; Chitnis, Hajiaghayi, Marx
2012; Chitnis, Cygan, Hajiaghayi, Marx

Problem Definition

The study of the parameterized complexity of problems on directed graphs has been hitherto relatively unexplored. Usually the directed version of the problems require significantly different and more involved ideas than the ones for the undirected version. Furthermore, for directed graphs there are no known algorithmic meta-techniques: for example, there is no known algorithmic analogue of the Graph Minor Theory of Robertson and Seymour for directed graphs. As a result, the fixed-parameter tractability status of the directed versions of several fundamental problems such as Multiway Cut, Multicut, Feedback Vertex Set, etc., was open for a long time. The problem of Feedback Vertex Set best illustrates this gulf between undirected and directed graphs with respect to parameterized complexity. In this problem, we are given a graph and the question is whether there exists a set of size at most k whose deletion makes the graph acyclic. The undirected version was known to be FPT

since 1984 [10]. However, the directed version was a long-standing open problem until it was shown to be FPT in 2008 [1].

The framework of *shadowless solutions* aims to bridge this gap by providing an important first step in designing FPT algorithms for a general class of transversal problems on directed graphs. In undirected graphs, the framework of *shadowless solutions* was introduced in [9] and has since been used in [4, 6, 7]. It was adapted and generalized to directed graphs in [2, 3] for the following general class of problems:

Finding an $\mathcal{F}$-transversal for some T-connected $\mathcal{F}$
Input: A directed graph $G = (V, E)$, a positive integer k, a set $T \subseteq V$, and a set $\mathcal{F} = \{F_1, F_2, \ldots, F_q\}$ of subgraphs such that $\mathcal{F}$ is T-connected, i.e., $\forall\, i \in [q]$ each vertex of F_i can reach some vertex of T by a walk completely contained in $G[F_i]$ and is reachable from some vertex of T by a walk completely contained in $G[F_i]$.
Parameter: k
Question: Is there an $\mathcal{F}$-transversal $W \subseteq V$ with $|W| \leq k$, i.e., a set W such that $F_i \cap W \neq \emptyset$ for every $i \in [q]$?

The collection $\mathcal{F}$ is implicitly defined in a problem-specific way and need not be given explicitly in the input. In fact, it is possible that $\mathcal{F}$ is exponentially large. The *shadow* of a solution X is the set of vertices that are disconnected from T (in either direction) after the removal of X. More formally, the reverse shadow of X is given by $r_T(X) = \{v : X \text{is a } v \to T \text{ separator}\}$. Similarly, the forward shadow of X is given by $f_T(X) = \{v : X \text{is a } T \to v \text{ separator}\}$. The shadow of X is given by the union of its reverse and forward shadows, i.e., $\text{shadow}(X) = r(X) \cup f(X)$. A set X is said to be *shadowless* if its shadow is empty.

The aim is to ensure first that there is a solution whose shadow is empty, as finding such a shadowless solution can be a significantly easier task.

Key Results

For the $\mathcal{F}$-transversal problem defined above, [2] shows how to invoke the technique of *random sampling of important separators* and obtain a set Z which is disjoint from a minimum solution X and covers its shadow.

Theorem 1 (randomized covering of the shadow) *Let $T \subseteq V(G)$. There is an algorithm $\texttt{RandomSet}(G, T, k)$ that runs in $4^k \cdot n^{O(1)}$ time and returns a set $Z \subseteq V(G)$ such that for any set $\mathcal{F}$ of T-connected subgraphs, if there exists an $\mathcal{F}$-transversal of size $\leq k$, then the following holds with probability $2^{-2^{O(k)}}$: there is an $\mathcal{F}$-transversal X of size $\leq k$ such that*

1. *$X \cap Z = \emptyset$ and*
2. *Z covers the shadow of X, i.e., $r(X) \cup f(X) \subseteq Z$.*

The set $\mathcal{F}$ is *not* an input of the algorithm described by Theorem 1: the set Z constructed in the above theorem works for *every* T-connected set $\mathcal{F}$ of subgraphs. Therefore, issues related to the representation of $\mathcal{F}$ do not arise. Theorem 1 can be derandomized using the theory of splitters [11]:

Theorem 2 (deterministic covering of the shadow) *Let $T \subseteq V(G)$. We can construct a set $\{Z_1, Z_2, \ldots, Z_t\}$ with $t = 2^{2^{O(k)}} \cdot \log^2 n$ in time $2^{2^{O(k)}} \cdot n^{O(1)}$ such that for any set $\mathcal{F}$ of T-connected, if there exists an $\mathcal{F}$-transversal of size $\leq k$, then there is an $\mathcal{F}$-transversal X of size $\leq k$ such that for at least one $1 \leq i \leq t$ we have*

1. *$X \cap Z_i = \emptyset$ and*
2. *Z_i covers the shadow of X, i.e., $r(X) \cup f(X) \subseteq Z_i$.*

Consider one such set Z_i for some $1 \leq i \leq 2^{2^{O(k)}} \cdot \log^2 n$. Since this set Z_i is disjoint from a minimum solution X, it can be removed from the graph. However, we need to remember the

structure that the set Z_i imposed on the problem. This structure is problem specific, and the reduced (equivalent) instance is obtained on a supergraph of $G \setminus Z_i$ via the *torso operation*. It can be shown that the original instance G has a solution if and only if the reduced instance has a shadowless solution. Therefore, one can focus on the simpler task of finding a shadowless solution or more precisely, finding any solution under the guarantee that a shadowless solution exists.

Applications

The first FPT algorithms for the Directed Multiway Cut problem [3] and the Directed Subset Feedback Vertex Set problem [2] were obtained via the framework of shadowless solutions.

Directed Multiway Cut

In the Directed Multiway Cut problem, given a directed graph $G = (V, E)$, an integer k, and a set of terminals $T = \{t_1, t_2, \ldots, t_p\}$, the objective is to find whether there exists a set $X \subseteq V(G)$ of size at most k such that $G \setminus X$ has no $t_i \to t_j$ path for any $1 \leq i \neq j \leq p$. Let $\mathcal{F}$ be the set of all paths between pairs of (distinct) terminals. Then it is easy to show that $\mathcal{F}$ is T-connected, and the problem of finding an $\mathcal{F}$-transversal is exactly the same as the Directed Multiway Cut problem. It is shown in [3] that a shadowless solution of Directed Multiway Cut is also a solution of the underlying undirected instance of Multiway Cut, which is known to be FPT [8] parameterized by k. Combining with Theorem 2, this gives an FPT algorithm for the Directed Multiway Cut problem.

Directed Subset Feedback Vertex Set

In the Directed Subset Feedback Vertex Set problem, given a directed graph $G = (V, E)$, an integer k, and a set $S \subseteq V(G)$, the objective is to find whether there exists a set $X \subseteq V(G)$ of size at most k such that $G \setminus X$ has no S-cycles, i.e., cycles containing at least one vertex of S. The special case when $S = V(G)$ is the Directed Feedback Vertex Set problem. Let $\mathcal{F}$ be the set of all S-cycles and T be a solution of size $k + 1$ (which can be obtained via *iterative compression*). Then it is easy to show that $\mathcal{F}$ is T-connected, and the problem of finding an $\mathcal{F}$-transversal is exactly the same as the Directed Subset Feedback Vertex Set problem. It is shown in [2] that a shadowless solution of Directed Subset Feedback Vertex Set can be found in FPT time. Combining with Theorem 2, this gives an FPT algorithm for the Directed Subset Feedback Vertex Set problem. This generalizes the FPT algorithm for Directed Feedback Vertex Set [1].

Open Problems

The two main open problems which fit within the framework of "Finding an $\mathcal{F}$-transversal for some T-connected $\mathcal{F}$" are Directed Multicut and Directed Odd Cycle Transversal. Unfortunately, the structure of shadowless solutions is not yet understood well enough to be able to find them in FPT time.

Directed Multicut

In the Directed Multicut problem, given a directed graph $G = (V, E)$, an integer k, and a set of terminal pairs $T = \{(s_1, t_1), (s_2, t_2), \ldots, (s_p, t_p)\}$, the objective is to find whether there exists a set $X \subseteq V(G)$ of size at most k such that $G \setminus X$ has no $s_i \to t_i$ path for any $1 \leq i \leq p$. Let $\mathcal{F}$ be the union of set of all $s_i \to t_i$ paths for $1 \leq i \leq p$. Then it is easy to show that $\mathcal{F}$ is T-connected, and the problem of finding an $\mathcal{F}$-transversal is exactly the same as the Directed Multicut problem. It is known [9] that Directed Multicut parameterized by k is W[1]-hard. However, for the special case of $p = 2$ terminal pairs, the problem can be reduced to Directed Multiway Cut and is hence FPT parameterized by k [3]. The complexity for $p = 3$ parameterized by k is an important open problem. With respect to the bigger parameter $p + k$, the problem is known [5] to be FPT on directed acyclic graphs. However, this algorithm heavily uses the properties of a topological ordering, and the complexity parameterized by

$p + k$ on general graphs is another important open problem.

Directed Odd Cycle Transversal

In the Directed Odd Cycle Transversal problem, given a directed graph $G = (V, E)$ and an integer k, the objective is to find whether there exists a set $X \subseteq V(G)$ of size at most k such that $G \setminus X$ has no cycle of odd length. Let $\mathcal{F}$ be the set of all odd cycles in G and T be a solution of size $k + 1$ (which can be obtained via *iterative compression* [12]). Then it is easy to show that $\mathcal{F}$ is T-connected, and the problem of finding an $\mathcal{F}$-transversal is exactly the same as the Directed Odd Cycle Transversal problem. The complexity parameterized by k is open. Moreover, it is known that Directed Odd Cycle Transversal problem generalizes the Directed Feedback Vertex Set problem [1] and the Undirected Odd Cycle Transversal problem [12]. Hence, an FPT algorithm for Directed Odd Cycle Transversal would have to generalize the ideas used to obtain FPT algorithms for these two problems.

Cross-References

- Bidimensionality
- Undirected Feedback Vertex Set

Recommended Reading

1. Chen J, Liu Y, Lu S, O'Sullivan B, Razgon I (2008) A fixed-parameter algorithm for the directed feedback vertex set problem. In: STOC, Victoria, pp 177–186
2. Chitnis RH, Cygan M, Hajiaghayi MT, Marx D (2012) Directed subset feedback vertex set is fixed-parameter tractable. In: ICALP (1), Warwick, pp 230–241
3. Chitnis RH, Hajiaghayi M, Marx D (2012) Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. In: SODA, Kyoto, pp 1713–1725
4. Chitnis RH, Egri L, Marx D (2013) List H-coloring a graph by removing few vertices. In: ESA, Sophia Antipolis, pp 313–324
5. Kratsch S, Pilipczuk M, Pilipczuk M, Wahlström M (2012) Fixed-parameter tractability of multicut in directed acyclic graphs. In: ICALP (1), Warwick, pp 581–593
6. Lokshtanov D, Marx D (2011) Clustering with local restrictions. In: ICALP (1), Zurich, pp 785–797
7. Lokshtanov D, Ramanujan MS (2012) Parameterized tractability of multiway cut with parity constraints. In: ICALP (1), Warwick, pp 750–761
8. Marx D (2006) Parameterized graph separation problems. Theor Comput Sci 351(3):394–406
9. Marx D, Razgon I (2011) Fixed-parameter tractability of multicut parameterized by the size of the cutset. In: STOC, San Jose, pp 469–478
10. Mehlhorn K (1984) Data structures and algorithms 2: graph algorithms and NP-completeness. Springer, Berlin/New York
11. Naor M, Schulman LJ, Srinivasan A (1995) Splitters and near-optimal derandomization. In: FOCS, Milwaukee, pp 182–191
12. Reed BA, Smith K, Vetta A (2004) Finding odd cycle transversals. Oper Res Lett 32(4):299–301

Shortest Elapsed Time First Scheduling

Nikhil Bansal
Eindhoven University of Technology, Eindhoven, The Netherlands

Keywords

Feedback queues; MLF algorithm; Response time; Scheduling with unknown job sizes; Sojourn time

Years and Authors of Summarized Original Work

2003; Bansal, Pruhs

Problem Definition

The problem is concerned with scheduling dynamically arriving jobs in the scenario when the processing requirements of jobs are unknown to the scheduler. The lack of knowledge of how long a job will take to execute is a particularly attractive assumption in real systems where such

information might be difficult or impossible to obtain. The goal is to schedule jobs to provide good quality of service to the users. In particular the goal is to design algorithms that have good average performance and are also fair in the sense that no subset of users experiences substantially worse performance than others.

Notations

Let $\mathcal{J} = \{1, 2, \ldots, n\}$ denote the set of jobs in the input instance. Each job j is characterized by its release time r_j and its processing requirement p_j. In the online setting, job j is revealed to the scheduler only at time r_j. A further restriction is the *non-clairvoyant* setting, where only the existence of job j is revealed at r_j, in particular the scheduler does not know p_j until the job meets its processing requirement and leaves the system. Given a schedule, the completion time c_j of a job is the earliest time at which job j receives p_j amount of service. The flow time f_j of j is defined as $c_j - r_j$. The stretch of a job is defined as the ratio of its flow time divided by its size. Stretch is also referred to as normalized flow time or slowdown and is a natural measure of fairness as it measures the waiting time of a job per unit of service received. A schedule is said to be preemptive, if a job can be interrupted arbitrarily, and its execution can be resumed later from the point of interruption without any penalty. It is well known that preemption is necessary to obtain reasonable guarantees for flow time even in the offline setting [6].

Recall that the online shortest remaining processing time (SRPT) algorithm that at any time works on the job with the least remaining processing is optimum for minimizing average flow time. However, a common critique of SRPT is that it may lead to starvation of jobs, where some jobs may be delayed indefinitely. For example, consider the sequence where a job of size 3 arrives at time $t = 0$ and one job of size 1 arrives every unit of time starting $t = 1$ for a long time. Under SRPT, the size 3 job will be delayed until the size 1 jobs stop arriving. On the other hand, if the goal is to minimize the maximum flow time, then it is easily seen that first in first out (FIFO) is the optimum algorithm. However, FIFO can perform very poorly with respect to average flow time (e.g., many small jobs could be stuck behind a very large job that arrived just earlier). A natural way to balance both the average and worst case performance is to consider the ℓ_p norms of flow time and stretch, where the ℓ_p norm of the sequence $x_1, \ldots, x_n$ is defined as $\left(\sum_i x_i^p\right)^{1/p}$.

The shortest elapsed time first (SETF) is a non-clairvoyant algorithm that at any time works on the job that has received the least amount of service thus far. This is a natural way to favor short jobs given the lack of knowledge of job sizes. In fact, SETF is the continuous version of the multilevel feedback (MLF) algorithm. Unfortunately, SETF (or any other deterministic non-clairvoyant algorithm) performs poorly in the framework of competitive analysis, where an algorithm is called c-competitive if for every input instance, its performance is no worse than c times that of the optimum offline (clairvoyant) solution for that instance [7]. However, competitive analysis can be overly pessimistic in its guarantee. A way around this problem was proposed by Kalyanasundaram and Pruhs [5] who allowed the online scheduler a slightly faster processor to make up for its lack of knowledge of future arrivals and job sizes. Formally, an algorithm Alg is said to be s-speed, c-speed competitive where c is the worst case ratio over all instance I, of $\text{Alg}_s(I)/\text{Opt}_1(I)$, where Alg_s is the value of solution produced by Alg when given an s-speed processor, and Opt_1 is the optimum value using a speed 1 processor. Typically the most interesting results are those where c is small and $s = (1+\epsilon)$ for any arbitrary $\epsilon > 0$.

Key Results

In their seminal paper [5], Kalyanasundaram and Pruhs showed the following.

Theorem 1 ([5]) *SETF is a $(1+\epsilon)$-speed, $(1 + 1/\epsilon)$-competitive non-clairvoyant algorithm for minimizing the average flow time on a single machine with preemptions.*

For minimizing the average stretch, Muthukrishnan, Rajaraman, Shaheen, and Gehrke [6] considered the clairvoyant setting and showed that SRPT is 2-competitive for a single machine and 14-competitive for multiple machines. The non-clairvoyant setting was consider by Bansal, Dhamdhere, Konemann, and Sinha [7]. They showed that

Theorem 2 ([1]) *SETF is a* $(1+\epsilon)$*-speed,* $0(\log^2 P)$*-competitive for minimizing average stretch, where* P *is the ratio of the maximum to minimum job size. On the other hand, even with* $O(1)$*-speed, any non-clairvoyant algorithm is at least* $\Omega(\log P)$*-competitive. Interestingly, in terms of* n*, any non-clairvoyant algorithm must be* $\Omega(n)$*-competitive even with* $O(1)$*-speedup. Moreover, SETF is* $O(n)$*-competitive (even without extra speedup). For the special case when all jobs arrive at time 0, SETF is optimum up to constant factors. It is* $O(\log P)$*-competitive (without any extra speedup). Moreover, any non-clairvoyant must be* $\Omega(\log P)$*-competitive even with factor* $O(1)$*-speedup.*

The key idea of the above result was a connection between SETF and SRPT. First, at the expense of $(1+\epsilon)$-speedup, it can be seen that SETF is no worse than MLF where the thresholds are powers of $(1+\epsilon)$. Second, the behavior of MLF on an instance I can be related to the behavior of shortest job first (SJF) algorithm on another instance I' that is obtained from/by dividing each job into logarithmically many jobs with geometrically increasing sizes. Finally, the performance of SJF is related to SRPT using another $(1+\epsilon)$ factor speedup.

Bansal and Pruhs [2] considered the problem of minimizing the ℓ_p norms of flow time and stretch on a single machine. They showed the following.

Theorem 3 ([2]) *In the clairvoyant setting, SRPT and SJF are* $(1+\epsilon)$*-speed,* $O(1/\epsilon)$*-competitive for minimizing the* ℓ_p *norms of both flow time and stretch. On the other hand, for* $1 < p < \infty$*, no online algorithm (possibly clairvoyant) can be* $O(1)$*-competitive for minimizing* ℓ_p *norms of stretch or flow time without speedup. In particular, any randomized online algorithm is at least* $\Omega(n^{(p-1)/3p^2})$*-competitive for* ℓ_p *norms of stretch and is at least* $\Omega(n^{(p-1)/p(3p-1)})$*-competitive for* ℓ_p *norms of flow time.*

The above lower bounds are somewhat surprising, since SRPT and FIFO are optimum for the case $p = 1$ and $p = \infty$ for flow time.

Bansal and Pruhs [2] also consider the non-clairvoyant case.

Theorem 4 ([2]) *In the non-clairvoyant setting, SETF is* $(1+\epsilon)$*-speed,* $O(1/\epsilon^{2+2/p})$*-competitive for minimizing the* ℓ_p *norms of flow time. For minimizing* ℓ_p *norms of stretch, SETF is* $(1+\epsilon)$*-speed,* $O(1/\epsilon^{3+1/p} \cdot \log^{1+1/p} P)$*-competitive*

Finally, Bansal and Pruhs also consider round robin (RR) or processor sharing that at any time splits the processor equally among the unfinished jobs. RR is considered to be an ideal fair strategy since it treats all unfinished jobs equally. However, they show that

Theorem 5 *For any* $p \geq 1$*, there is an* $\epsilon > 0$ *such that even with a* $(1+\epsilon)$ *times faster processor, RR is not* $n^{o(1)}$*-competitive for minimizing the* ℓ_p *norms of flow time. In particular, for* $\epsilon < 1/2p$*, RR is* $(1+\epsilon)$*-speed,* $\Omega(n^{(1-2\epsilon p)/p})$*-competitive. For* ℓ_p *norms of stretch, RR is* $\Omega(n)$*-competitive as is in fact any randomized non-clairvoyant algorithm.*

The results above have been extended in a couple of directions. Bansal and Pruhs [3] extend these results to *weighted* ℓ_p norms of flow time and stretch. Chekuri, Khanna, Kumar, and Goel [4] have extended these results to the multiple machines case. Their algorithms are particularly elegant: Each job is assigned to some machine at random, and all jobs at a particular machine are processed using SRPT or SETF (as applicable).

Applications

SETF and its variants such as MLF are widely used in operating systems [9,10]. Note that SETF is not really practical since each job could be

preempted infinitely often. However, variants of SETF with fewer preemptions are quite popular.

Open Problems

It would be interesting to explore other notions of fairness in the dynamic scheduling setting. In particular, it would be interesting to consider algorithms that are both fair and have a good average performance.

An immediate open problem is whether the gap between $O(\log^2 P)$ and $\Omega(\log P)$ can be closed for minimizing the average stretch in the non-clairvoyant setting.

Cross-References

- ▸ Flow Time Minimization
- ▸ Minimum Flow Time
- ▸ Multilevel Feedback Queues

Recommended Reading

1. Bansal N, Dhamdhere K, Konemann J, Sinha A (2004) Non-clairvoyant scheduling for minimizing mean slowdown. Algorithmica 40(4):305–318
2. Bansal N, Pruhs K (2003) Server scheduling in the Lp norm: a rising tide lifts all boat. In: Symposium on theory of computing (STOC), San Diego, pp 242–250
3. Bansal N, Pruhs K (2004) Server scheduling in the weighted Lp norm. In: LATIN, Buenos Aires, pp 434–443
4. Chekuri C, Goel A, Khanna S, Kumar A (2004) Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In: Symposium on theory of computing (STOC), Chicago, pp 363–372
5. Kalyanasundaram B, Pruhs K (2000) Speed is as powerful as clairvoyance. J ACM 47(4):617–643
6. Kellerer H, Tautenhahn T, Woeginger GJ (1999) Approximability and nonapproximability results for minimizing total flow time on a single machine. SIAM J Comput 28(4):1155–1166
7. Motwani R, Phillips S, Torng E (1994) Non-clairvoyant scheduling. Theor Comput Sci 130(1):17–47
8. Muthukrishnan S, Rajaraman R, Shaheen A, Gehrke J (2004) Online scheduling to minimize average stretch. SIAM J Comput 34(2):433–452
9. Nutt G (1999) Operating system projects using Windows NT. Addison Wesley, Reading
10. Tanenbaum AS (1992) Modern operating systems. Prentice-Hall, Englewood Cliffs

Shortest Paths Approaches for Timetable Information

Riko Jacob
Institute of Computer Science,
Technical University of Munich, Munich, Germany
IT University of Copenhagen, Copenhagen, Denmark

Keywords

Journey planner; Passenger information system; Timetable lookup; Trip planner

Years and Authors of Summarized Original Work

2004; Pyrga, Schulz, Wagner, Zaroliagis

Problem Definition

Consider the route-planning task for passengers of scheduled public transportation. Here, the running example is that of a train system, but the discussion applies equally to bus, light-rail and similar systems. More precisely, the task is to construct a timetable information system that, based upon the detailed schedules of all trains, provides passengers with good itineraries, including the transfer between different trains.

Solutions to this problem consist of a model of the situation (e.g., can queries specify a limit on the number of transfers?), an algorithmic approach, its mathematical analysis (does it always return the best solution? Is it guaranteed to work fast in all settings?), and an evaluation in the real world (Can travelers actually use the produced itineraries? Is an implementation fast enough on current computers and real data?).

Key Results

The problem is discussed in detail in a recent survey article [6].

Modeling

In a simplistic model, it is assumed that a transfer between trains does not take time. A more realistic model specifies a certain minimum transfer time per station. Furthermore, the objective of the optimization problem needs to be defined. Should the itinerary be as fast as possible, or as cheap as possible, or induce the least possible transfers? There are different ways to resolve this as surveyed in [6], all originating in multi-objective optimization, like resource constraints or Pareto-optimal solutions. From a practical point of view, the preferences of a traveler are usually difficult to model mathematically, and one might want to let the user choose the best option among a set of reasonable itineraries himself. For example, one can compute all itineraries that are not inferior to some other itinerary in all considered aspects. As it turns out, in real timetables the number of such itineraries is not too big, such that this approach is computationally feasible and useful for the traveler [5]. Additionally, the fare structure of most railways is fairly complicated [4], mainly because fares usually are not additive, i.e., are not the sum of fares of the parts of a trip.

Algorithmic Models

The current literature establishes two main ideas how to transform the situation into a shortest path problem on a graph. As an example, consider the simplistic modeling where transfer takes no time, and where queries specify starting time and station to ask for an itinerary that achieves the earliest arrival time at the destination.

In the time-expanded model [11], every arrival and departure event of the timetable is a vertex of the directed graph. The arcs of the graph represent consecutive events at one station, and direct train connections. The length of an arc is given by the time difference of its end vertices. Let s be the vertex at the source station whose time is directly after the starting time. Now, a shortest path from s to any vertex of the destination station is an optimal itinerary.

In the time-dependent model [3, 7, 9, 10], the vertices model stations, and the arcs stand for the existence of a direct (non-stop) train connection. Instead of edge length, the arcs are labeled with edge-traversal functions that give the arrival time at the end of the arc in dependence on the time a passenger starts at the beginning of the arc, reflecting the times when trains actually run. To solve this time-dependent shortest path problem, a modification of Dijkstra's algorithm can be used. Further exploiting the structure of this situation, the graph can be represented in a way that allows constant time evaluation of the link traversal functions [3]. To cope with more realistic transfer models, a more complicated graph can be used.

Additionally, many of the speed-up techniques for shortest path computations can be applied to the resulting graph queries.

Applications

The main application are timetable information systems for scheduled transit (buses, trains, etc.). This extends to route planning where trips in such systems are allowed, as for example in the setting of fine-grained traffic simulation to compute fastest itineraries [2].

Open Problems

Improve computation speed, in particular for fully integrated timetables and the multi-criteria case. Extend the problem to the dynamic case, where the current real situation is reflected, i.e., delayed or canceled trains, and otherwise temporarily changed timetables are reflected.

Experimental Results

In the cited literature, experimental results usually are part of the contribution [2, 4, 5, 6, 7, 8, 9, 10, 11]. The time-dependent approach can

be significantly faster than the time-expanded approach. In particular for the simplistic models speed-ups in the range 10–45 are observed [8, 10]. For more detailed models, the performance of the two approaches becomes comparable [6].

Cross-References

- ▸ Implementation Challenge for Shortest Paths
- ▸ Routing in Road Networks with Transit Nodes
- ▸ Single-Source Shortest Paths

Acknowledgments I want to thank Matthias Müller-Hannemann, Dorothea Wagner, and Christos Zaroliagis for helpful comments on an earlier draft of this entry.

Recommended Reading

1. Gerards B, Marchetti-Spaccamela A (eds) (2004) Proceedings of the 3rd workshop on algorithmic methods and models for optimization of railways (ATMOS'03) 2003. Electronic notes in theoretical computer science, vol 92. Elsevier
2. Barrett CL, Bisset K, Jacob R, Konjevod G, Marathe MV (2002) Classical and contemporary shortest path problems in road networks: implementation and experimental analysis of the TRANSIMS router. In: Algorithms – ESA 2002: 10th annual European symposium, Rome, 17–21 Sept 2002. Lecture notes computer science, vol 2461. Springer, Berlin, pp 126–138
3. Brodal GS, Jacob R (2003) Time-dependent networks as models to achieve fast exact time-table queries. In: Proceedings of the 3rd workshop on algorithmic methods and models for optimization of railways (ATMOS'03), [1], pp 3–15
4. Müller-Hannemann M, Schnee M (2006) Paying less for train connections with MOTIS. In: Kroon LG, Möhring RH (eds) Proceedings of the 5th workshop on algorithmic methods and models for optimization of railways (ATMOS'05), Dagstuhl, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl. Dagstuhl Seminar Proceedings, no. 06901
5. Müller-Hannemann M, Schnee M (2007) Finding all attractive train connections by multi-criteria pareto search. In: Geraets F, Kroon LG, Schöbel A, Wagner D, Zaroliagis CD (eds) Algorithmic methods for railway optimization, international Dagstuhl workshop, Dagstuhl Castle, 20–25 June 2004, 4th international workshop, ATMOS 2004, Bergen, 16–17 Sept 2004, revised selected papers. Lecture notes in computer science, vol 4359. Springer, Berlin, pp 246–263
6. Müller-Hannemann M, Schulz F, Wagner D, Zaroliagis CD (2007) Timetable information: models and algorithms. In: Geraets F, Kroon LG, Schöbel A, Wagner D, Zaroliagis CD (eds) Algorithmic methods for railway optimization, international Dagstuhl workshop, Dagstuhl Castle, 20–25 June 2004, 4th International Workshop, ATMOS 2004, Bergen, 16–17 Sept 2004, revised selected papers. Lecture notes in computer science, vol 4359. Springer, pp 67–90
7. Nachtigall K (1995) Time depending shortest-path problems with applications to railway networks. Eur J Oper Res 83:154–166
8. Pyrga E, Schulz F, Wagner D, Zaroliagis C (2004) Experimental comparison of shortest path approaches for timetable information. In: Proceedings 6th workshop on algorithm engineering and experiments (ALENEX). Society for Industrial and Applied Mathematics, pp 88–99
9. Pyrga E, Schulz F, Wagner D, Zaroliagis C (2003) Towards realistic modeling of time-table information through the time-dependent approach. In: Proceedings of the 3rd workshop on algorithmic methods and models for optimization of railways (ATMOS'03), [1], pp 85–103
10. Pyrga E, Schulz F, Wagner D, Zaroliagis C (2007) Efficient models for timetable information in public transportation systems. J Exp Algorithm 12:2.4
11. Schulz F, Wagner D, Weihe K (2000) Dijkstra's algorithm on-line: an empirical case study from public railroad transport. J Exp Algorithm 5:1–23

Shortest Paths in Planar Graphs with Negative Weight Edges

Jittat Fakcharoenphol[1] and Satish Rao[2]
[1]Department of Computer Engineering, Kasetsart University, Bangkok, Thailand
[2]Department of Computer Science, University of California, Berkeley, CA, USA

Keywords

Shortest paths in planar graphs with arbitrary arc weights; Shortest paths in planar graphs with general arc weights

Years and Authors of Summarized Original Work

2001; Fakcharoenphol, Rao

Problem Definition

This problem is to find shortest paths in planar graphs with general edge weights. It is known that shortest paths exist only in graphs that contain no negative weight cycles. Therefore, algorithms that work in this case must deal with the presence of negative cycles, i.e., they must be able to detect negative cycles.

In general graphs, the best known algorithm, the Bellman-Ford algorithm, runs in time $O(mn)$ on graphs with n nodes and m edges, while algorithms on graphs with no negative weight edges run much faster. For example, Dijkstra's algorithm implemented with the Fibonacchi heap runs in time $O(m + n \log n)$, and, in case of integer weights Thorup's algorithm runs in linear time. Goldberg [5] also presented an $O(m\sqrt{n} \log L)$-time algorithm where L denotes the absolute value of the most negative edge weights. Note that his algorithm is weakly polynomial.

Notations

Given a directed graph $G = (V, E)$ and a weight function w: $E \to \mathbb{R}$ on its directed edges, a *distance labeling* for a source node s is a function d: $V \to \mathbb{R}$ such that $d(v)$ is the minimum length over all s-to-v paths, where the *length of path P* is $\sum_{e \in P} w(e)$.

Problem 1 (Single-Source-Shortest-Path)
INPUT: A directed graph $G = (V, E)$, weight function w: $E \to \mathbb{R}$, source node $s \in V$.
OUTPUT: If G does not contain negative length cycles, output a distance labeling d for source node s. Otherwise, report that the graph contains some negative length cycle.

The algorithm by Fakcharoenphol and Rao [4] deals with the case when G is planar. They gave an $O(n \log^3 n)$-time algorithm, improving on an $O(n^{3/2})$-time algorithm by Lipton, Rose, and Tarjan [9] and an $O(n^{4/3} \log nL)$-time algorithm by Henzinger, Klein, Rao, and Subramanian [6].

Their algorithm, as in all previous algorithms, uses a recursive decomposition and constructs a data structure called a dense distance graph, which shall be defined next.

A *decomposition* of a graph is a set of subsets $P_1, P_2, \ldots, P_k$ (not necessarily disjoint) such that the union of all the sets is V and for all $e = (u, v) \in E$, there is a unique P_i that contains e. A node v is a *border node* of a set P_i if $v \in P_i$ and there exists an edge $e = (v, x)$ where $x \notin P_i$. The subgraph induced on a subset P_i is referred to as a *piece* of the decomposition.

The algorithm works with a *recursive decomposition* where at each level, a piece with n nodes and r border nodes is divided into two subpieces such that each subpiece has no more than $2n/3$ nodes and at most $2r/3 + c\sqrt{n}$ border nodes, for some constant c. In this recursive context, a border node of a subpiece is defined to be any border node of the original piece or any new border node introduced by the decomposition of the current piece.

With this recursive decomposition, the *level of a decomposition* can be defined in the natural way, with the entire graph being the only piece in the level 0 decomposition, the pieces of the decomposition of the entire graph being the level 1 pieces in the decomposition, and so on.

For each piece of the decomposition, the all-pair shortest path distances between all its border nodes along paths that lie entirely inside the piece are recursively computed. These all-pair distances form the edge set of a non-planar graph representing shortest paths between border nodes. The dense distance graph of the planar graph is the union of these graphs over all the levels.

Using the dense distance graph, the shortest distance queries between pairs of nodes can be answered.

Problem 2 (Shortest-Path-Distance-Data-Structure)
INPUT: A directed graph $G = (V, E)$, weight function w: $E \to \mathbb{R}$, source node $s \in V$.
OUTPUT: If G does not contain negative length cycles, output a data structure that support distance queries between pairs of nodes. Otherwise, report that the graph contains some negative length cycle.

The algorithm of Fakcharoenphol and Rao relies heavily on planarity, i.e., it exploits properties regarding how shortest paths on each piece intersect. Therefore, unlike previous algorithms that require only that the graph can be recursively decomposed with small numbers of border nodes [10], their algorithm also requires that each piece has a nice embedding.

Given an embedding of the piece, a *hole* is a bounded face where all adjacent nodes are border nodes. Ideally, one would hope that there is a planar embedding of any piece in the recursive decomposition where all the border nodes are on a single face and are circularly ordered, i.e., there is no holes in each piece. Although this is not always true, the algorithm works with any decomposition with a constant number of holes in each piece. This decomposition can be found in $O(n \log n)$ time using the simple cycle separator algorithm by Miller [12].

Key Results

Theorem 1 *Given a recursive decomposition of a planar graph such that each piece of the decomposition contains at most a constant number of holes, there is an algorithm that constructs the dense distance graph is $O(n \log^3 n)$ time.*

Given the procedure that constructs the dense distance graph, the shortest paths from a source s can be computed by first adding s as a border node in every piece of the decomposition, computing the dense distance graph, and then extending the distances into all internal nodes on every piece. This can be done in time $O(n \log^3 n)$.

Theorem 2 *The single-source shortest path problem for an n-node planar graph with negative weight edges can be solved in time $O(n \log^3 n)$.*

The dense distance graph can be used to answer distance queries between pairs of nodes.

Theorem 3 *Given the dense distance graph, the shortest distance between any pair of nodes can be found in $O(\sqrt{n} \log^2 n)$ time.*

It can also be used as a dynamic data structure that answers shortest path queries and allows edge cost updates.

Theorem 4 *For planar graphs with only non-negative weight edges, there is a dynamic data structure that supports distance queries and update operations that change edge weights in amortized $O(n^{2/3} \log^{7/3} n)$ time per operation. For planar graph with negative weight edges, there is a dynamic data structures that supports the same set of operations in amortized $O(n^{4/5} \log^{13/5} n)$ time per operation.*

Note that the dynamic data structure does not support edge insertions and deletions, since these operations might destroy the recursive decomposition.

Applications

The shortest path problem has long been studied and continues to find applications in diverse areas. There are a many problems that reduce to the shortest path problem where negative weight edges are required, for example the minimum-mean length directed circuit. For planar graphs, the problem has wide application even when the underlying graph is a grid. For example, there are recent image segmentation approaches that use negative cycle detection [2, 3]. Some of other applications for planar graphs include separator algorithms [13] and multi-source multi-sink flow algorithms [11].

Open Problems

Klein [8] gives a technique that improves the running time of the construction of the dense distance graph to $O(n \log^2 n)$ when all edge weights are non-negative; this also reduces the amortized running time for the dynamic case down to $O(n^{2/3} \log^{5/3} n)$. Also, for planar graphs with no negative weight edges, Cabello [1] gives a faster algorithm for computing the shortest distances between k pairs of nodes. However, the problem

for improving the bound of $O(n \log^3 n)$ for finding shortest paths in planar graphs with general edge weights remains opened.

It is not known how to handle edge insertions and deletions in the dynamic data structure. A new data structure might be needed instead of the dense distance graph, because the dense distance graph is determined by the decomposition.

Cross-References

- All Pairs Shortest Paths in Sparse Graphs
- All Pairs Shortest Paths via Matrix Multiplication
- Approximation Schemes for Planar Graph Problems
- Decremental All-Pairs Shortest Paths
- Fully Dynamic All Pairs Shortest Paths
- Implementation Challenge for Shortest Paths
- Negative Cycles in Weighted Digraphs
- Planarity Testing
- Shortest Paths Approaches for Timetable Information
- Single-Source Shortest Paths

Recommended Reading

1. Cabello S (2006) Many distances in planar graphs. In: SODA '06: proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithm. ACM, New York, pp 1213–1220
2. Cox IJ, Rao SB, Zhong Y (1996) 'Ratio Regions': a technique for image segmentation. In: Proceedings international conference on pattern recognition, Aug 1996. IEEE, pp 557–564
3. Geiger LCD, Gupta A, Vlontzos J (1995) Dynamic programming for detecting, tracking and matching elastic contours. IEEE Trans Pattern Anal Mach Intell
4. Fakcharoenphol J, Rao S (2006) Planar graphs, negative weight edges, shortest paths, and near linear time. J Comput Syst Sci 72:868–889
5. Goldberg AV (1992) Scaling algorithms for the shortest path problem. SIAM J Comput 21:140–150
6. Henzinger MR, Klein PN, Rao S, Subramanian S (1997) Faster shortest-path algorithms for planar graphs. J Comput Syst Sci 55:3–23
7. Johnson D (1977) Efficient algorithms for shortest paths in sparse networks. J Assoc Comput Mach 24:1–13
8. Klein PN (2005) Multiple-source shortest paths in planar graphs. In: Proceedings of the 16th ACM-SIAM symposium on discrete algorithms, pp 146–155
9. Lipton R, Rose D, Tarjan RE (1979) Generalized nested dissection. SIAM J Numer Anal 16:346–358
10. Lipton RJ, Tarjan RE (1979) A separator theorem for planar graphs. SIAM J Appl Math 36:177–189
11. Miller G, Naor J (1995) Flow in planar graphs with multiple sources and sinks. SIAM J Comput 24:1002–1017
12. Miller GL (1986) Finding small simple cycle separators for 2-connected planar graphs. J Comput Syst Sci 32:265–279
13. Rao SB (1992) Faster algorithms for finding small edge cuts in planar graphs (extended abstract). In: Proceedings of the twenty-fourth annual ACM symposium on the theory of computing, May 1992, pp 229–240
14. Thorup M (2004) Compact oracles for reachability and approximate distances in planar digraphs. J ACM 51:993–1024

Shortest Vector Problem

Daniele Micciancio
Department of Computer Science, University of California, San Diego, La Jolla, CA, USA

Keywords

Closest vector problem; Lattice basis reduction; LLL algorithm; Nearest vector problem; Minimum distance problem

Years and Authors of Summarized Original Work

1982; Lenstra, Lenstra, Lovasz

Problem Definition

A *point lattice* is the set of all integer linear combinations

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i : x_1, \ldots, x_n \in \mathbb{Z} \right\}$$

of n linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$ in m-dimensional Euclidean space. For computational purposes, the lattice vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ are often assumed to have integer (or rational)

entries, so that the lattice can be represented by an integer matrix $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbb{Z}^{m \times n}$ (called *basis*) having the generating vectors as columns. Using matrix notation, lattice points in $\mathcal{L}(\mathbf{B})$ can be conveniently represented as $\mathbf{Bx}$ where $\mathbf{x}$ is an integer vector. The integers m and n are called the *dimension* and *rank* of the lattice respectively. Notice that any lattice admits multiple bases, but they all have the same rank and dimension.

The main computational problems on lattices are the *Shortest Vector Problem*, which asks to find the shortest nonzero vector in a given lattice, and the *Closest Vector Problem*, which asks to find the lattice point closest to a given target. Both problems can be defined with respect to any norm, but the Euclidean norm $\|\mathbf{v}\| = \sqrt{\sum_i v_i^2}$ is the most common. Other norms typically found in computer science applications are the ℓ_1 norm $\|\mathbf{v}\|_1 = \sum_i |v_i|$ and the *max* norm $\|\mathbf{v}\|_\infty = \max_i |v_i|$. This entry focuses on the Euclidean norm.

Since no efficient algorithm is known to solve SVP and CVP exactly in arbitrary high dimension, the problems are usually defined in their approximation version, where the approximation factor $\gamma \geq 1$ can be a function of the dimension or rank of the lattice.

Definition 1 (Shortest Vector Problem, SVP_γ) Given a lattice $\mathcal{L}(\mathbf{B})$, find a nonzero lattice vector $\mathbf{Bx}$ (where $\mathbf{x} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$) such that $\|\mathbf{Bx}\| \leq \gamma \cdot \|\mathbf{By}\|$ for any $\mathbf{y} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$.

Definition 2 (Closest Vector Problem, CVP_γ) Given a lattice $\mathcal{L}(\mathbf{B})$ and a target point $\mathbf{t}$, find a lattice vector $\mathbf{Bx}$ (where $\mathbf{x} \in \mathbb{Z}^n$) such that $\|\mathbf{Bx} - \mathbf{t}\| \leq \gamma \cdot \|\mathbf{By} - \mathbf{t}\|$ for any $\mathbf{y} \in \mathbb{Z}^n$.

Lattices have been investigated by mathematicians for centuries in the equivalent language of quadratic forms, and are the main object of study in the *geometry of numbers*, a field initiated by Minkowski as a bridge between geometry and number theory. For a mathematical introduction to lattices see [3]. The reader is referred to [6, 12] for an introduction to lattices with an emphasis on computational and algorithmic issues.

Key Results

The problem of finding an efficient (polynomial time) solution to SVP_γ for lattices in arbitrary dimension was first solved by the celebrated *lattice reduction* algorithm of Lenstra, Lenstra and Lovász [11], commonly known as the *LLL* algorithm.

Theorem 1 *There is a polynomial time algorithm to solve* SVP_γ *for* $\gamma = (2/\sqrt{3})^n$*, where* n *is the rank of the input lattice.*

The LLL algorithm achieves more than just finding a relatively short lattice vector: it finds a so-called *reduced basis* for the input lattice, i.e., an entire basis of relatively short lattice vectors. Shortly after the discovery of the LLL algorithm, Babai [2] showed that reduced bases can be used to efficiently solve CVP_γ as well within similar approximation factors.

Corollary 1 There is a polynomial time algorithm to solve CVP_γ for $\gamma = O(2/\sqrt{3})^n$, where n is the rank of the input lattice.

The reader is referred to the original papers [2, 11] and [12, chap. 2] for details. Introductory presentations of the LLL algorithm can also be found in many other texts, e.g., [5, chap. 16] and [15, chap. 27]. It is interesting to note that CVP is at least as hard as SVP (see [12, chap 2]) in the sense that any algorithm that solves CVP_γ can be efficiently adapted to solve SVP_γ within the same approximation factor.

Both SVP_γ and CVP_γ are known to be NP-hard in their exact ($\gamma = 1$) or even approximate versions for small values of γ, e.g., constant γ independent of the dimension. (See [13, chaps. 3 and 4] and [4, 10] for the most recent results.) So, no efficient algorithm is likely to exist to solve the problems exactly in arbitrary dimension. For any fixed dimension n, both SVP and CVP can be solved exactly in polynomial time using an algorithm of Kannan [9]. However, the dependency of the running time on the lattice dimension is $n^{O(n)}$. Using randomization, exact SVP can be

solved probabilistically in $2^{O(n)}$ time and space using the *sieving* algorithm of Ajtai, Kumar and Sivakumar [1].

As for approximate solutions, the LLL lattice reduction algorithm has been improved both in terms of running time and approximation guarantee. (See [14] and references therein.) Currently, the best (randomized) polynomial time approximation algorithm achieves approximation factor $\gamma = 2^{O(n \log\log n / \log n)}$.

Applications

Despite the large (exponential in n) approximation factor, the LLL algorithm has found numerous applications and lead to the solution of many algorithmic problems in computer science. The number and variety of applications is too large to give a comprehensive list. Some of the most representative applications in different areas of computer science are mentioned below.

The first motivating applications of lattice basis reduction were the solution of integer programs with a fixed number of variables and the factorization of polynomials with rationals coefficients. (See [11, 8], and [15, chap. 16].) Other classic applications are the solution of random instances of low-density subset-sum problems, breaking (truncated) linear congruential pseudorandom generators, simultaneous Diophantine approximation, and the disproof of Mertens' conjecture. (See [8] and [5, chap. 17].)

More recently, lattice basis reduction has been extensively used to solve many problems in cryptanalysis and coding theory, including breaking several variants of the RSA cryptosystem and the DSA digital signature algorithm, finding small solutions to modular equations, and list decoding of CRT (Chinese Reminder Theorem) codes. The reader is referred to [7, 13] for a survey of recent applications, mostly in the area of cryptanalysis.

One last class of applications of lattice problems is the design of cryptographic functions (e.g., collision resistant hash functions, public key encryption schemes, etc.) based on the apparent intractability of solving SVP_γ within small approximation factors. The reader is referred to [12, chap. 8] and [13] for a survey of such applications, and further pointers to relevant literature. One distinguishing feature of many such lattice based cryptographic functions is that they can be proved to be hard to break *on the average*, based on a *worst-case* intractability assumption about the underlying lattice problem.

Open Problems

The main open problems in the computational study of lattices is to determine the complexity of approximate SVP_γ and CVP_γ for approximation factors $\gamma = n^c$ polynomial in the rank of the lattice. Specifically,

- Are there polynomial time algorithm that solve SVP_γ or CVP_γ for polynomial factors $\gamma = n^c$? (Finding such algorithms even for very large exponent c would be a major breakthrough in computer science.)
- Is there an $\epsilon > 0$ such that approximating SVP_γ or CVP_γ to within $\gamma = n^\epsilon$ is NP-hard? (The strongest known inapproximability results [4] are for factors of the form $n^{O(1/\log\log n)}$ which grow faster than any poly-logarithmic function, but slower than any polynomial.)

There is theoretical evidence that for large polynomials factors $\gamma = n^c$, SVP_γ and CVP_γ are not NP-hard. Specifically, both problems belong to complexity class coAM for approximation factor $\gamma = O(\sqrt{n/\log n})$. (See [12, chap. 9].) So, the problems cannot be NP-hard within such factors unless the polynomial hierarchy PH collapses.

URL to Code

The LLL lattice reduction algorithm is implemented in most library and packages for computational algebra, e.g.,

- GAP (http://www.gap-system.org)
- LiDIA (http://www.cdc.informatik.tu-darmstadt.de/TI/LiDIA/)
- Magma (http://magma.maths.usyd.edu.au/magma/)
- Maple (http://www.maplesoft.com/)
- Mathematica (http://www.wolfram.com/products/mathematica/index.html)
- NTL (http://shoup.net/ntl/).

NTL also includes an implementation of Block Korkine-Zolotarev reduction that has been extensively used for cryptanalysis applications.

Cross-References

- ▸ Cryptographic Hardness of Learning
- ▸ Knapsack
- ▸ Learning Heavy Fourier Coefficients of Boolean Functions
- ▸ Quantum Algorithm for the Discrete Logarithm Problem
- ▸ Quantum Algorithm for Factoring
- ▸ Sphere Packing Problem

Recommended Reading

1. Ajtai M, Kumar R, Sivakumar D (2001) A sieve algorithm for the shortest lattice vector problem. In: Proceedings of the thirty-third annual ACM symposium on theory of computing – STOC 2001, Heraklion, July 2001. ACM, New York, pp 266–275
2. Babai L (1986) On Lovasz' lattice reduction and the nearest lattice point problem. Combinatorica 6(1):1–13, Preliminary version in STACS 1985
3. Cassels JWS (1971) An introduction to the geometry of numbers. Springer, New York
4. Dinur I, Kindler G, Raz R, Safra S (2003) Approximating CVP to within almost-polynomial factors is NP-hard. Combinatorica 23(2):205–243, Preliminary version in FOCS 1998
5. von zur Gathen J, Gerhard J (2003) Modern computer algebra, 2nd edn. Cambridge
6. Grotschel M, Lovász L, Schrijver A (1993) Geometric algorithms and combinatorial optimization. Algorithms and combinatorics, vol 2, 2nd edn. Springer
7. Joux A, Stern J (1998) Lattice reduction: a toolbox for the cryptanalyst. J Cryptol 11(3):161–185
8. Kannan R (1987) Algorithmic geometry of numbers. In: Annual reviews of computer science, vol 2. Annual Review, Palo Alto, pp 231–267
9. Kannan R (1987) Minkowski's convex body theorem and integer programming. Math Oper Res 12(3):415–440
10. Khot S (2005) Hardness of approximating the shortest vector problem in lattices. J ACM 52(5):789–808, Preliminary version in FOCS 2004
11. Lenstra AK, Lenstra HW Jr, Lovász L (1982) Factoring polynomials with rational coefficients. Math Ann 261:513–534
12. Micciancio D, Goldwasser S (2002) Complexity of lattice problems: a cryptographic perspective, vol 671, The Kluwer international series in engineering and computer science. Kluwer Academic, Boston
13. Nguyen P, Stern J (2001) The two faces of lattices in cryptology. In: Silverman J (ed) Cryptography and lattices conference – CaLC 2001, Providence, Mar 2001. Lecture notes in computer science, vol 2146. Springer, Berlin, pp 146–180
14. Schnorr CP (2006) Fast LLL-type lattice reduction. Inf Comput 204(1):1–25
15. Vazirani VV (2001) Approximation algorithms. Springer

Similarity Between Compressed Strings

Jin Wook Kim[1], Amihood Amir[2,5], Gad M. Landau[3], and Kunsoo Park[4]
[1]HM Research, Seoul, Korea
[2]Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel
[3]Department of Computer Science, University of Haifa, Haifa, Israel
[4]School of Computer Science and Engineering, Seoul National University, Seoul, Korea
[5]Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

Keywords

Alignment between compressed strings; Compressed approximate string matching; Similarity between compressed strings

Years and Authors of Summarized Original Work

2005; Kim, Amir, Landau, Park

Similarity Between Compressed Strings, Table 1 Various scoring metrics

Metric	Match	Mismatch	Indel	Indel of k characters
Longest common subsequence	1	0	0	0
Levenshtein distance	0	1	1	k
Weighted edit distance	0	δ	μ	$k\mu$
Affine gap penalty	1	$-\delta$	$-\gamma-\mu$	$-\gamma-k\mu$

Problem Definition

The problem of computing similarity between two strings is concerned with comparing two strings using some scoring metric. There exist various scoring metrics and a popular one is the Levenshtein distance (or edit distance) metric. The standard solution for the Levenshtein distance metric was proposed by Wagner and Fischer [13], which is based on dynamic programming. Other widely used scoring metrics are the longest common subsequence metric, the weighted edit distance metric, and the affine gap penalty metric. The affine gap penalty metric is the most general, and it is a quite complicated metric to deal with. Table 1 shows the differences between the four metrics.

The problem considered in this entry is the similarity between two compressed strings. This problem is concerned with efficiently computing similarity without decompressing two strings. The compressions used for this problem in the literature are run-length encoding and Lempel-Ziv (LZ) compression [14].

Run-Length Encoding

A string S is run-length encoded if it is described as an ordered sequence of pairs (σ, i), often denoted "σ^i", each consisting of an alphabet symbol, σ, and an integer, i. Each pair corresponds to a *run* in S, consisting of i consecutive occurrences of σ. For example, the string *aaabbbbbaccccbb* can be encoded $a^3b^4a^1c^4b^2$ or, equivalently, $(a,3)(b,4)(a,1)(c,4)(b,2)$. Let A and B be two strings with lengths n and m, respectively. Let A' and B' be the run-length encoded strings of A and B, and n' and m' be the lengths of A' and B', respectively.

Problem 1
INPUT: Two run-length encoded strings A' and B', a scoring metric d.
OUTPUT: The similarity between A' and B' using d.

LZ Compression

Let X and Y be two strings with length $O(n)$. Let X' and Y' be the LZ compressed strings of X and Y, respectively. Then the lengths of X' and Y' are $O(hn/\log n)$, where $h \leq 1$ is the entropy of strings X and Y.

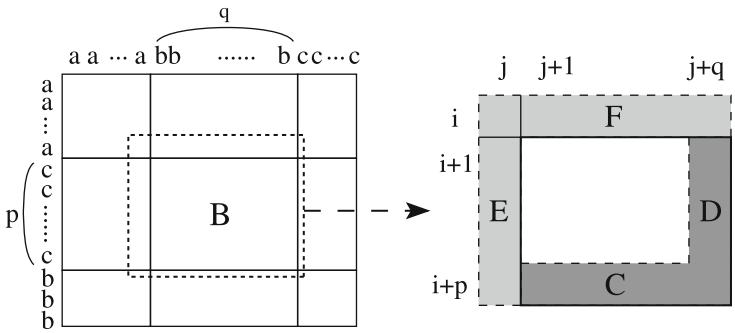
Problem 2
INPUT: Two LZ compressed strings X' and Y', a scoring metric d.
OUTPUT: The similarity between X' and Y' using d.

Block Computation

To compute similarity between compressed strings efficiently, one can use a block computation method. Dynamic programming tables are divided into submatrices, which are called "*blocks*". For run-length encoded strings, a block is a submatrix made up of two runs – one of A and one of B. For LZ compressed strings, a block is a submatrix made up of two phrases – one phrase from each string. See [5] for more details. Then, blocks are computed from left to right and from top to bottom. For each block, only the bottom row and the rightmost column are computed. Figure 1 shows an example of block computation.

Key Results

The problem of computing similarity of two run-length encoded strings, A' and B', has been



Similarity Between Compressed Strings, Fig. 1 Dynamic programming table for strings $a^r c^p b^t$ and $a^s b^q c^u$ is divided into 9 blocks. For one of the blocks, e.g., B, only the bottom row C and the rightmost column D are computed from E and F

studied for various scoring metrics. Bunke and Csirik [4] presented the first solution to Problem 1 using the longest common subsequence metric. The algorithm is based on block computation of the dynamic programming table.

Theorem 1 (Bunke and Csirik [4]) *A longest common subsequence of run-length encoded strings A' and B' can be computed in $O(nm' + n'm)$ time.*

For the Levenshtein distance metric, Arbell, Landau, and Mitchell [2] and Mäkinen, Navarro, and Ukkonen [10] presented $O(nm' + n'm)$ time algorithms, independently. These algorithms are extensions of the algorithm of Bunke and Csirik.

Theorem 2 (Arbell, Landau, and Mitchell [2], Mäkinen, Navarro, and Ukkonen [10]) *The Levenshtein distance between run-length encoded strings A' and B' can be computed in $O(nm' + n'm)$ time.*

For the weighted edit distance metric, Crochemore, Landau, and Ziv-Ukelson [6] and Mäkinen, Navarro, and Ukkonen [11] gave $O(nm' + n'm)$ time algorithms using techniques completely different from each other. The algorithm of Crochemore, Landau, and Ziv-Ukelson [6] is based on the technique which is used in the LZ compressed pattern matching algorithm [6], and the algorithm of Mäkinen, Navarro, and Ukkonen [11] is an extension of the algorithm for the Levenshtein distance metric.

Theorem 3 (Crochemore, Landau, and Ziv-Ukelson [6] Mäkinen, Navarro, and Ukkonen [11]) *The weighted edit distance between run-length encoded strings A' and B' can be computed in $O(nm' + n'm)$ time.*

For the affine gap penalty metric, Kim, Amir, Landau, and Park [8] gave an $O(nm' + n'm)$ time algorithm. To compute similarity in this metric efficiently, the problem is converted into a path problem on a directed acyclic graph and some properties of maximum paths in this graph are used. It is not necessary to build the graph explicitly since they came up with new recurrences using the properties of the graph.

Theorem 4 (Kim, Amir, Landau, and Park [8]) *The similarity between run-length encoded strings A' and B' in the affine gap penalty metric can be computed in $O(nm' + n'm)$ time.*

The above results show that comparison of run-length encoded strings using the longest common subsequence metric is successfully extended to more general scoring metrics.

For the longest common subsequence metric, there exist improved algorithms. Apostolico, Landau, and Skiena [1] gave an $O(n'm' \log(n'm'))$ time algorithm. This algorithm is based on tracing specific optimal paths.

Theorem 5 (Apostolico, Landau, and Skiena [1]) *A longest common subsequence of run-length encoded strings A' and B' can be computed in $O(n'm' \log(n' + m'))$ time.*

Mitchell [12] obtained an $O((d + n' + m') \log(d + n' + m'))$ time algorithm, where d is the number of matches of compressed characters. This algorithm is based on computing geometric

shortest paths using special convex distance functions.

Theorem 6 (Mitchell [12]) *A longest common subsequence of run-length encoded strings A' and B' can be computed in $O((d + n' + m')\log(d + n' + m'))$ time, where d is the number of matches of compressed characters.*

Mäkinen, Navarro, and Ukkonen [11] conjectured an $O(n'm')$ time algorithm on average under the assumption that the lengths of the runs are equally distributed in both strings.

Conjecture 1 (Mäkinen, Navarro, and Ukkonen [11]) A longest common subsequence of run-length encoded strings A' and B' can be computed in $O(n'm')$ time on average.

For Problem 2, Crochemore, Landau, and Ziv-Ukelson [6] presented a solution using the additive gap penalty metric. The additive gap penalty metric consists of 1 for match, $-\delta$ for mismatch, and $-\mu$ for indel, which is almost the same as the weighted edit distance metric.

Theorem 7 (Crochemore, Landau, and Ziv-Ukelson [6]) *The similarity between LZ compressed strings X' and Y' in the additive gap penalty metric can be computed in $O(hn^2/\log n)$ time, where $h \leq 1$ is the entropy of strings X and Y.*

Applications

Run-length encoding serves as a popular image compression technique, since many classes of images (e.g., binary images in facsimile transmission or for use in optical character recognition) typically contain large patches of identically-valued pixels. Approximate matching on images can be a useful tool to handle distortions. Even a one-dimensional compressed approximate matching algorithm would be useful to speed up two-dimensional approximate matching allowing mismatches and even rotations [3, 7, 9].

Open Problems

The worst-case complexity of the problem is not fully understood. For the longest common subsequence metric, there exist some results whose time complexities are better than $O(nm' + n'm)$ to compute the similarity of two run-length encoded strings [1, 11, 12]. It remains open to extend these results to the Levenshtein distance metric, the weighted edit distance metric and the affine gap penalty metric.

In addition, for the longest common subsequence metric, it is an open problem to prove Conjecture 1.

Cross-References

- Approximate String Matching
- Local Alignment (with Affine Gap Weights)
- Multidimensional Compressed Pattern Matching

Recommended Reading

1. Apostolico A, Landau GM, Skiena S (1999) Matching for run length encoded strings. J Complex 15(1):4–16
2. Arbell O, Landau GM, Mitchell J (2002) Edit distance of run-length encoded strings. Inf Process Lett 83(6):307–314
3. Baeza-Yates R, Navaro G (2000) New models and algorithms for multidimensional approximate pattern matching. J Discret Algorithm 1(1):21–49
4. Bunke H, Csirik H (1995) An improved algorithm for computing the edit distance of run length coded strings. Inf Process Lett 54:93–96
5. Crochemore M, Landau GM, Schieber B, Ziv-Ukelson M (2005) Re-use dynamic programming for sequence alignment: an algorithmic toolkit. In: Iliopoulos CS, Lecroq T (eds) String algorithmics. King's College London Publications, London, pp 19–59
6. Crochemore M, Landau GM, Ziv-Ukelson M (2003) A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM J Comput 32(6):1654–1673
7. Fredriksson K, Navarro G, Ukkonen E (2005) Sequential and indexed two-dimensional combinatorial template matching allowing rotations. Theor Comput Sci 347(1–2):239–275

8. Kim JW, Amir A, Landau GM, Park K (2005) Computing similarity of run-length encoded strings with affine gap penalty. In: Proceedings of the 12th symposium on string processing and information retrieval (SPIRE'05), LNCS, vol 3772, pp 440–449
9. Krithivasan K, Sitalakshmi R (1987) Efficient two-dimensional pattern matching in the presence of errors. Inf Sci 43:169–184
10. Mäkinen V, Navarro G, Ukkonen E (2001) Approximate matching of run-length compressed strings. In: Proceedings of the 12th symposium on combinatorial pattern matching (CPM'01). LNCS, vol 2089, pp 31–49
11. Mäkinen V, Navarro G, Ukkonen E (2003) Approximate matching of run-length compressed strings. Algorithmica 35:347–369
12. Mitchell J (1997) A geometric shortest path problem, with application to computing a longest common subsequence in run-length encoded strings. Technical report, Department of Applied Mathematics, SUNY Stony Brook
13. Wagner RA, Fischer MJ (1974) The string-to-string correction problem. J ACM 21(1):168–173
14. Ziv J, Lempel A (1978) Compression of individual sequences via variable rate coding. IEEE Trans Inf Theory 24(5):530–536

Simple Algorithms for Spanners in Weighted Graphs

Surender Baswana[1] and Sandeep Sen[2]
[1]Department of Computer Science and Engineering, Indian Institute of Technology (IIT), Kanpur, Kanpur, India
[2]Indian Institute of Technology (IIT) Delhi, Hauz Khas, New Delhi, India

Keywords

Graph algorithms; Randomized algorithms; Shortest path; Spanner

Years and Authors of Summarized Original Work

2003; Baswana, Sen

Problem Definition

A spanner is a *sparse* subgraph of a given undirected graph that preserves approximate distance between each pair of vertices. More precisely, a t-spanner of a graph $G = (V, E)$ is a subgraph (V, E_S), $E_S \subseteq E$ such that, for any pair of vertices, their distance in the subgraph is at most t times their distance in the original graph, where t is called the *stretch factor*. The spanners were defined formally by Peleg and Schäffer [15] though the associated notion was used implicitly by Awerbuch [3] in the context of network synchronizers.

Computing t-spanner of smallest size for a given graph is a well-motivated combinatorial problem with many applications. However, computing t-spanner of smallest size for a graph is NP-hard. In fact, for $t > 2$, it is NP-hard [11] even to approximate the smallest size of t-spanner of a graph with ratio $O(2^{(1-\mu)\ln n})$ for any $\mu > 0$. Having realized this fact, researchers have pursued another direction which is quite interesting and useful. Let S_G^t be the size of the sparsest t-spanner of a graph G, and let $\mathcal{S}_n^t$ be the maximum value of S_G^t over all possible graphs on n vertices. Does there exist a polynomial time algorithm which computes, for any weighted graph and parameter t, its t-spanner of size $O(\mathcal{S}_n^t)$? Such an algorithm would be the best one can hope for given the hardness of the original t-spanner problem. Naturally, the question arises as to how large can $\mathcal{S}_n^t$ be ? A 43-year-old girth lower bound conjecture by Erdös [13] implies that there are graphs on n vertices whose $2k-$ as well as $(2k-1)$-spanner will require $\Omega(n^{1+1/k})$ edges. This conjecture has been proved for $k = 1, 2, 3$, and 5. Note that a $(2k-1)$-spanner is also a $2k$-spanner, and the lower bound on the size is the same for both $2k$-spanner and $(2k-1)$-spanner. So the objective is to design an algorithm that, for any weighted graph on n vertices, computes a $(2k-1)$-spanner of $O(n^{1+1/k})$ size. Needless to say, one would like to design the fastest algorithm for this problem, and the most ambitious aim would be to achieve the linear time complexity.

Key Results

The key results of this entry are two very simple algorithms which compute a $(2k-1)$-spanner of a given weighted graph $G = (V, E)$. Let n and m

denote, respectively, the number of vertices and edges of G. The first algorithm, due to Althöfer et al. [2], is based on a greedy strategy and runs in $O(mn^{1+1/k})$ time. The second algorithm [6] is based on a very local approach and runs in an expected $O(m)$ time. To start with, consider the following simple observation. Suppose there is a subset $E_S \subset E$ that ensures the following proposition for every edge $(x, y) \in E \setminus E_S$.

> $\mathcal{P}_t(x, y)$: the vertices x and y are connected in the subgraph (V, E_S) by a path consisting of at most t edges, and the weight of each edge on this path is not more than that of the edge (x, y).

It follows easily that the subgraph (V, E_S) will be a t-spanner of G. The two algorithms for computing $(2k-1)$-spanner eventually compute such set E_S based on two completely different approaches.

Algorithm I

This algorithm selects edges for spanner in a greedy fashion and is similar to Kruskal's algorithm for computing a minimum spanning tree. The edges of the graph are processed in the increasing order of their weights. To begin with, the spanner $E_S = \emptyset$; and the algorithm adds edges to it gradually. The decision as to whether an edge, say (u, v), has to be added (or not) to E_S is made as follows:

> If the distance between u and v in the subgraph induced by the current spanner edges E_S is more than $t \cdot weight(u, v)$, then add the edge (u, v) to E_S; otherwise, discard the edge.

It follows that $\mathcal{P}_t(x, y)$ would hold for each edge of E missing in E_S, and so at the end, the subgraph (V, E_S) will be a t-spanner. A well-known result in elementary graph theory states that a graph with more than $n^{1+1/k}$ edges must have a cycle of length at most $2k$. It follows from the above algorithm that the length of any cycle in the subgraph (V, E_S) has to be at least $t + 1$. Hence, for $t = 2k - 1$, the number of edges in the subgraph (V, E_S) will be less than $n^{1+1/k}$. Thus, the algorithm I described above computes a $(2k-1)$-spanner of size $O(n^{1+1/k})$, which is indeed optimal based on the lower bound mentioned earlier.

A simple $O(mn^{1+1/k})$ implementation of algorithm I follows based on Dijkstra's algorithm. Cohen [10] and later Thorup and Zwick [19] designed algorithms for $(2k-1)$-spanner with an improved running time of $O(kmn^{1+1/k})$. These algorithms relied on several calls to Dijkstra's single-source Shortest path algorithm for distance computation and therefore were far from achieving linear time. On the other hand, since a spanner must approximate all-pairs distances in a graph, it appears difficult to compute a spanner by avoiding explicit distance information. Somewhat surprisingly, algorithm II, described in the following section, avoids any sort of distance computation and achieves expected linear time.

Algorithm II

This algorithm employs a novel clustering based on a very local approach and establishes the following result for the spanner problem:

> Given a weighted graph $G = (V, E)$ and an integer $k > 1$, a spanner of $(2k-1)$ stretch and $O(kn^{1+1/k})$ size can be computed in expected $O(km)$ time.

The algorithm executes in $O(k)$ rounds, and in each round it essentially explores adjacency list of each vertex to prune dispensable edges. As a testimony of its simplicity, we will present the entire algorithm for 3-spanner and its analysis in the following section. The algorithm can be easily adapted in other computational models (parallel, external memory, distributed) with nearly optimal performance (see [6] for more details).

Computing a 3-Spanner in Linear Time

To meet the size constraint of a 3-spanner, a vertex, on an average, contributes $\sqrt{n}$ edges to the spanner. So the vertices with degree $O(\sqrt{n})$ are easy to handle since all their edges can be selected in the spanner. For vertices with higher degree, a clustering (groupings) scheme is employed to tackle this problem which has its basis in *dominating sets*.

To begin with, there is a set of edges E' initialized to E and empty spanner E_S. The algorithm processes the edges E', moves some of them to the spanner E_S, and discards the remaining ones. It does so in the following two phases:

1. *Forming the clusters*
   A sample $\mathcal{R} \subset V$ is chosen by picking each vertex independently with probability $\frac{1}{\sqrt{n}}$. The clusters will be formed around these sampled vertices. Initially, the clusters are $\{\{u\}|u \in \mathcal{R}\}$. Each $u \in \mathcal{R}$ is called the *center* of its cluster. Each unsampled vertex $v \in V - \mathcal{R}$ is processed as follows:
   (a) If v is not adjacent to any sampled vertex, then every edge incident on v is moved to E_S.
   (b) If v is adjacent to one or more sampled vertices, let $\mathcal{N}(v, \mathcal{R})$ be the sampled neighbor that is nearest (Ties can be broken arbitrarily. However, it helps conceptually to assume that all weights are distinct) to v. The edge $(v, \mathcal{N}(v, \mathcal{R}))$ along with every edge that is incident on v with weight less than this edge is moved to E_S. The vertex v is added to the cluster centered at $\mathcal{N}(v, \mathcal{R})$.
   As a last step of the first phase, all those edges (u, v) from E' where u and v are not sampled and belong to the same cluster are discarded.
   Let V' be the set of vertices corresponding to the endpoints of the edges E' left after the first phase. It follows that each vertex from V' is either a sampled vertex or adjacent to some sampled vertex, and step 1(b) has partitioned V' into disjoint clusters each centered around some sampled vertex. Also note that, as a consequence of the last step, each edge of the set E' is an intercluster edge. The graph (V', E'), and the corresponding clustering of V', is passed onto the following (second) phase.
2. *Joining vertices with their neighboring clusters*
   Each vertex v of graph (V', E') is processed as follows. Let $E'(v, c)$ be the edges from the set E' incident on v from a cluster c. For each cluster c neighboring to v, the least-weight edge from $E'(v, c)$ is moved to E_S, and the remaining edges are discarded.

The number of edges added to the spanner E_S during the algorithm described above can be bounded as follows. Note that the sample set $\mathcal{R}$ is formed by picking each vertex randomly independently with probability $\frac{1}{\sqrt{n}}$. It thus follows from elementary probability that for each vertex $v \in V$, the expected number of incident edges with weight less than that of $(v, \mathcal{N}(v, \mathcal{R}))$ is at most $\sqrt{n}$. Thus, the expected number of edges contributed to the spanner by each vertex in the first phase of the algorithm is at most $\sqrt{n}$. The number of edges added to the spanner in the second phase is $O(n|\mathcal{R}|)$. Since the expected size of the sample $\mathcal{R}$ is $\sqrt{n}$, therefore, the expected number of edges added to the spanner in the second phase is at most $n^{3/2}$. Hence, the expected size of the spanner E_S at the end of the algorithm described above is at most $2n^{3/2}$. The algorithm is repeated if the size of the spanner exceeds $3n^{3/2}$. It follows using Markov's inequality that the expected number of such repetitions will be $O(1)$.

We now establish that E_S is a 3-spanner. Note that for every edge $(u, v) \notin E_S$, the vertices u, v belong to some cluster in the first phase. There are two cases now.

Case 1 : *(u and v belong to the same cluster)*

Let u and v belong to the cluster centered at $x \in \mathcal{R}$. It follows from the first phase of the algorithm that there is a 2-edge path $u - x - v$ in the spanner with each edge not heavier than the edge (u, v). (This provides a justification for discarding all intracluster edges at the end of the first phase.)

Case 2 : *(u and v belong to different clusters)*

Clearly, the edge (u, v) was removed from E' during phase 2, and suppose it was removed while processing the vertex u. Let v belong to the cluster centered at $x \in \mathcal{R}$.

In the beginning of the second phase, let $(u, v') \in E'$ be the least-weight edge among all the edges incident on u from the vertices of the cluster centered at x. So it

must be that $weight(u, v') \leq weight(u, v)$. The processing of vertex u during the second phase of our algorithm ensures that the edge (u, v') gets added to E_S. Hence, there is a path $\Pi_{uv} = u - v' - x - v$ between u and v in the spanner E_S, and its weight can be bounded as $weight(\Pi_{uv}) = weight(u, v') + weight(v', x) + weight(x, v)$. Since (v', x) and (v, x) were chosen in the first phase, it follows that $weight(v', x) \leq weight(u, v')$ and $weight(x, v) \leq weight(u, v)$. It follows that the spanner (V, E_S) has stretch 3. Moreover, both phases of the algorithm can be executed in $O(m)$ time using elementary data structures and bucket sorting.

The algorithm for computing a $(2k - 1)$-spanner executes k iterations where each iteration is similar to the first phase of the 3-spanner algorithm. For details and formal proofs, the reader may refer to [6].

Other Related Works

The notion of a spanner has been generalized in the past by many researchers.

Additive Spanners

A t-spanner as defined above approximates pairwise distances with multiplicative error and can be called a multiplicative spanner. In an analogous manner, one can define spanners that approximate pairwise distances with additive error. Such a spanner is called an additive spanner, and the corresponding error is called *surplus*. Aingworth et al. [1] presented the first additive spanner of size $O(n^{3/2} \log n)$ with surplus 2. Baswana et al. [7] presented a construction of $O(n^{4/3})$-size additive spanner with surplus 6. Recently, Chechik [9] presented a construction of $O(n^{7/5})$-size additive spanner with surplus 4. It is a major open problem if there exists any sparser additive spanner.

(α, β)-Spanner

Elkin and Peleg [12] introduced the notion of (α, β)-spanner for unweighted graphs, which can be viewed as a hybrid of multiplicative and additive spanners. An (α, β)-spanner is a subgraph such that the distance between any pair of vertices $u, v \in V$ in this subgraph is bounded by $\alpha\delta(u, v) + \beta$, where $\delta(u, v)$ is the distance between u and v in the original graph. Elkin and Peleg showed that an $(1 + \epsilon, \beta)$-spanner of size $O(\beta n^{1+\delta})$, for arbitrarily small $\epsilon, \delta > 0$, can be computed at the expense of sufficiently large surplus β. Recently, Thorup and Zwick [20] introduced a spanner where the additive error is sublinear in terms of the *distance* being approximated.

Other interesting variants of spanner include *distance preserver* proposed by Bollobás et al. [8] and *lightweight* spanner proposed by Awerbuch et al. [4]. A subgraph is said to be a d-preserver if it preserves exact distances for each pair of vertices which are separated by distance at least d. A lightweight spanner tries to minimize the number of edges as well as the total edge weight. A *lightness* parameter is defined for a subgraph as the ratio of total weight of all its edges and the weight of the minimum spanning tree of the graph. Awerbuch et al. [4] showed that for any weighted graph and integer $k > 1$, there exists a polynomially constructible $O(k)$-spanner with $O(k\rho n^{1+1/k})$ edges and $O(k\rho n^{1/k})$ lightness, where $\rho = \log(diameter)$.

In addition to the above work on the generalization of spanners, a lot of work has also been done on computing spanners for special classes of graphs, e.g., chordal graphs, unweighted graphs, and Euclidean graphs. For chordal graphs, Peleg and Schäffer [15] designed an algorithm that computes a 2-spanner of size $O(n^{3/2})$ and a 3-spanner of size $O(n \log n)$. For unweighted graphs, Halperin and Zwick [14] gave an $O(m)$ time algorithm for this problem. Salowe [18] presented an algorithm for computing a $(1 + \epsilon)$-spanner of a d-dimensional complete Euclidean graph in $O(n \log n + \frac{n}{\epsilon^d})$ time. However, none of the algorithms for these special classes of graphs seem to extend to general weighted undirected graphs.

Applications

Spanners are quite useful in various applications in the area of distributed systems and

communication networks. In these applications, spanners appear as the underlying graph structure. In order to build compact routing tables [17], many existing routing schemes use the edges of a sparse spanner for routing messages. In distributed systems, spanners play an important role in designing *synchronizers*. Awerbuch [3] and Peleg and Ullman [16] showed that the quality of a spanner (in terms of stretch factor and the number of spanner edges) is very closely related to the time and communication complexity of any synchronizer for the network. The spanners have also been used implicitly in a number of algorithms for computing all-pairs approximate shortest paths [5, 10, 19, 21]. For a number of other applications, please refer to the papers [2, 3, 15, 17].

Open Problems

The running time as well as the size of the $(2k-1)$-spanner computed by the algorithm described above are away from their respective worst-case lower bounds by a factor of k. For any constant value of k, both these parameters are optimal. However, for the extreme value of k, that is, for $k = \log n$, there is deviation by a factor of $\log n$. Is it possible to get rid of this multiplicative factor of k from the running time of the algorithm and/or the size of the $(2k-1)$-spanner computed? It seems that a more careful analysis coupled with advanced probabilistic tools might be useful in this direction.

Recommended Reading

1. Aingworth D, Chekuri C, Indyk P, Motwani R (1999) Fast estimation of diameter and shortest paths(without matrix multiplication). SIAM J Comput 28:1167–1181
2. Althöfer I, Das G, Dobkin DP, Joseph D, Soares J (1993) On sparse spanners of weighted graphs. Discret Comput Geom 9:81–100
3. Awerbuch B (1985) Complexity of network synchronization. J Assoc Comput Mach 32(4):804–823
4. Awerbuch B, Baratz A, Peleg D (1992) Efficient broadcast and light weight spanners. Tech. Report CS92-22, Weizmann Institute of Science
5. Awerbuch B, Berger B, Cowen L, Peleg D (1998) Near-linear time construction of sparse neighborhod covers. SIAM J Comput 28:263–277
6. Baswana S, Sen S (2007) A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. Random Struct Algorithms 30(4):532–563
7. Baswana S, Kavitha T, Mehlhorn K, Pettie S (2010) Additive spanners and (alpha, beta)-spanners. ACM Trans Algorithms 7(1):5
8. Bollobás B, Coppersmith D, Elkin M (2003) Sparse distance preserves and additive spanners. In Proceedings of the 14th annual ACM-SIAM symposium on discrete algorithms (SODA), Baltimore, pp 414–423
9. Chechik S (2013) New additive spanners. In Proceedings of the twenty-fourth annual ACM-SIAM symposium on discrete algorithms, SODA 2013, New Orleans, 6–8 Jan 2013, pp 498–512
10. Cohen E (1998) Fast algorithms for constructing t-spanners and paths with stretch t. SIAM J Comput 28:210–236
11. Elkin M, Peleg D (2000) The hardness of approximating spanner problems. In STACS 2000, 17th annual symposium on theoretical aspects of computer science, Lille, Feb 2000, proceedings, pp 370–381
12. Elkin M, Peleg D (2004) $(1+\epsilon, \beta)$-spanner construction for general graphs. SIAM J Comput 33:608–631
13. Erdős P (1964) Extremal problems in graph theory. In Theory of graphs and its applications (Proc. Sympos. Smolenice,1963), pp 29–36, Publ. House Czechoslovak Acad. Sci., Prague
14. Halperin S, Zwick U (1996) Linear time deterministic algorithm for computing spanners for unweighted graphs. Unpublished manuscript
15. Peleg D, Schaffer AA (1989) Graph spanners. J Graph Theory 13:99–116
16. Peleg D, Ullman JD (1989) An optimal synchronizer for the hypercube. SIAM J Comput 18:740–747
17. Peleg D, Upfal E (1989) A trade-off between space amd efficiency for routing tables. J Assoc Comput Mach 36(3):510–530
18. Salowe JD (1991) Construction of multidimensional spanner graphs, with application to minimum spanning trees. In ACM symposium on computational geometry, North Conway, pp 256–261
19. Thorup M, Zwick U (2005) Approximate distance oracles. J Assoc Comput Mach 52:1–24
20. Thorup M, Zwick U (2006) Spanners and emulators with sublinear distance errors. In Proceedings of 17th annual ACM-SIAM symposium on discrete algorithms, Miami, 22-26 Jan 2006, pp 802–809

21. Wulff-Nilsen C (2012) Approximate distance oracles with improved preprocessing time. In Proceedings of the twenty-third annual ACM-SIAM symposium on discrete algorithms, SODA 2012, Kyoto, 17–19 Jan 2012, pp 202–208

Simpler Approximation for Stable Marriage

Zoltán Király
Department of Computer Science, Eötvös Loránd University, Budapest, Hungary
Egerváry Research Group (MTA-ELTE), Eötvös Loránd University, Budapest, Hungary

Keywords

Approximation; Local algorithm; SMTI; Stable marriage

Years and Authors of Summarized Original Work

2013; Király

Introduction

We have a two-sided market, one side is a set U of men, the other side is a set V of women. The first part of the input also contains the mutually acceptable man-woman pairs E. This makes up a bipartite graph $G(U \cup V, E)$. The second part of the input contains the preference lists of each person, that is a weak order (may contain ties) on his/her acceptable pairs.

A *matching* is a set of mutually disjoint acceptable man-woman pairs. Given a matching M, a man m and a woman w form a *blocking* pair, if they are an acceptable pair but are not partners in M, and they both prefer each other to their partner, or have no partner in M. That is either w is unmatched in M or w prefers m to her M-partner, *and* either m is unmatched in M or m prefers w to his M-partner. A matching M is *stable* if there are no blocking pairs.

We consider a two-sided market under incomplete preference lists with ties (SMTI), where the goal is to find a maximum size stable matching (MAX-SMTI).

Problem Definition

Problem 1 (MAX-SMTI)

INPUT: Set U of men, and set V of women and each person's preference list.
OUTPUT: A stable matching of maximum size.

Input format A list of an agent a consists of pairs (a_1, p_1), $(a_2, p_2), \ldots, (a_d, p_d)$, where a_i are the acceptable persons from the other gender and $1 \leq p_i \leq \max(|U|, |V|)$ are integers with ordering $p_1 \geq p_2 \geq \cdots \geq p_d$. Agent a strictly prefers a_i to a_j if $p_i > p_j$ and is indifferent between a_i and a_j if $p_i = p_j$. Moreover women needs a black-box procedure, which on input a_i outputs in constant time p_i (we assume that this procedure is also a part of the input). The size of the input is the number of agents plus the total length of the lists.

Definition of approximation ratios A goodness measure of an approximation algorithm A for a maximization problem is defined as follows: the *approximation ratio* of A is $\max\{\text{opt}(I)/A(I)\}$ over all instances I, where $\text{opt}(I)$ and $A(I)$ are the size of the optimal and the algorithm's solution on instance I, respectively.

Short history It was shown in [4] that finding the optimal solution is NP-hard; moreover, it is APX-hard [3]. The original Deferred Acceptance Algorithm of Gale and Shapley gives a 2-approximation; the first approximation algorithm with a strictly better ratio was presented in [5], where the approximation ratio was 15/8. This was improved in [6] to a 5/3-approximation and later in [9] to a 3/2-approximation; this latter algorithm had nonlinear running time. Recently in [10] and

in [7], linear time 3/2-approximation algorithms were given.

Key Results

A simple variation of the famous Deferred Acceptance Algorithm of Gale and Shapley is presented; which also runs in linear time and gives a 3/2-approximation for the problem MAX-SMTI. This algorithm is local; no central agent or knowledge about the global input is needed.

Algorithm

Preliminary Definitions and Concepts for the Algorithm

During the algorithm, the agents may have different statuses, and some Boolean properties described below, and also varying actual preferences.

A status of a man can be either a **lad** or a **bachelor** or an **old bachelor**. A man can be **active** or inactive. A man is active, if he is not an old bachelor and he is not **engaged** (i.e., he has actually no partner). A man can also be **uncertain**, described later. Initially every man is an active lad.

A status of a woman can be either **maiden** or **engaged**. An engaged woman is **flighty**, if her fiancé is *uncertain*. Initially every woman is maiden.

The actual preferences a man m is described as follows. If women w_1 and w_2 are indifferent on m's list, and w_1 is maiden but w_2 is engaged, then m **prefers maiden** w_1 to **engaged** w_2. An engaged lad is **uncertain** if his list contains a woman he prefers to his actual fiancée (this can happen, if there were two maidens with the same highest priority on m's list, and m became engaged to one of them).

The actual preferences a woman w is described as follows. If there are two men, m_1 and m_2 with the same priority in w's list, and m_1 is a lad, but m_2 is a bachelor, then w **prefers bachelor** m_2 to **lad** m_1. If w is flighty, then she prefers a man who is not uncertain, to a man who is uncertain (regardless of her original preferences).

The Algorithm

> While there exists an active man m, he proposes to his favorite woman w. If w accepts his proposal, they become engaged. If w rejects him, m deletes w from his list and remain active.
> When a woman w gets a new proposal from man m, she accepts this proposal if she (actually) prefers m to her current fiancé. Otherwise she rejects m.
> If w accepted m, then she rejects her previous fiancé, if there was one (breaks off her engagement), and becomes engaged to m.
> If m was engaged to a woman w and later w rejects him, then m becomes active again and deletes w from his list, except if m is uncertain, in this case m keeps w on the list.
> If the list of m becomes empty for the first time, he turns into a bachelor, his original list is recovered, and he reactivates himself. If the list of m becomes empty for the second time, he will turn into an old bachelor and will remain inactive forever.

After the algorithm finishes, the engaged pairs get married and form matching M.

Theorem 1 ([7]) *The algorithm always gives a stable matching M and it is $3/2$-approximating, i.e., the stable matching given has size at least $2/3$ of the maximum size stable matching.*

Running Time, Locality

This algorithm runs in linear time using the assumptions on the input format. Though it is clear that along every edge at most three proposals happen, the technical details must be worked out; see [7] for details.

Local algorithm Each agent (a man or woman) always makes a greedy decision based only on local information (his/her preference list, and provided by some communication with his/her acceptable partners). A local algorithm is linear if

every agent communicates with each acceptable partner only a constant time during the algorithm.

The algorithm presented is a linear time local algorithm (using the appropriate data structures); see [7] for details.

Cross-References

- ▸ Hospitals/Residents Problem
- ▸ Maximum Cardinality Stable Matchings
- ▸ Stable Marriage
- ▸ Stable Marriage with One-Sided Ties
- ▸ Stable Marriage with Ties and Incomplete Lists

Recommended Reading

1. Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69:9–15
2. Gusfield D, Irving RW (1989) The stable marriage problem: structure and algorithms. MIT, Boston
3. Halldórsson MM, Irving RW, Iwama K, Manlove DF, Miyazaki S, Morita Y, Scott S (2003) Approximability results for stable marriage problems with ties. Theor Comput Sci 306:431–447
4. Iwama K, Manlove DF, Miyazaki S, Morita Y (1999) Stable marriage with incomplete lists and ties. In: Proceedings of the 26th international colloquium on automata, languages and programming (ICALP 1999), Prague. LNCS, vol 1644, pp 443–452
5. Iwama K, Miyazaki S, Yamauchi N (2007) A 1.875-approximation algorithm for the stable marriage problem. In: SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms, pp 288–297
6. Király Z (2009(online), 2011) Better and simpler approximation algorithms for the stable marriage problem. Algorithmica 60(1):3–20
7. Király Z (2013) Linear time local approximation algorithm for maximum stable marriage. Algorithms 6(3):471–484
8. Manlove D (2013) Algorithmics of matching under preferences. World Scientific Publishing, Singapore
9. McDermid EJ (2009) A $\frac{3}{2}$-approximation algorithm for general stable marriage. In: Proceedings of the 36th international colloquium automata, languages and programming (ICALP 2009), Rhodes, pp 689–700
10. Paluch K (2014) Faster and simpler approximation of stable matchings. Algorithms 7(2):189–202

Single and Multiple Buffer Processing

Sergey I. Nikolenko[1] and Kirill Kogan[2]
[1]Laboratory of Mathematical Logic, Steklov Institute of Mathematics, St. Petersburg, Russia
[2]IMDEA Networks, Madrid, Spain

Keywords

Admission control; Buffer management policies; Online algorithms

Years and Authors of Summarized Original Work

2004; Kesselman, Mansour
2012; Keslassy, Kogan, Scalosub, Segal
2012; Kesselman, Kogan, Segal
2012, 2013; Kogan, López-Ortiz, Nikolenko, Sirotkin
2014; Eugster, Kogan, Nikolenko, Sirotkin

Problem Definition

Buffer management policies are online algorithms that control a limited buffer of packets with homogeneous or heterogeneous characteristics, deciding whether to accept new packets when they arrive, which packets to process and transmit, and possibly whether to push out packets already residing in the buffer. Although settings differ, the problem is always to achieve the best possible competitive ratio, i.e., find a policy with good worst-case guarantees in comparison with an optimal offline clairvoyant algorithm. The policies themselves are often simple, simplicity being an important advantage for implementation in switches; the hard problem is to find proofs of lower and especially upper bounds for their competitive

The work of Sergey Nikolenko was partially supported by the Government of the Russian Federation (grant 14.Z50.31.0030).

ratios. Thus, this problem is more theoretical in nature, although the resulting throughput guarantees are important tools in the design of network elements. Comprehensive surveys of this field have been given in the past by Goldwasser [9] and Epstein and van Stee [7].

General Model Description

We assume discrete slotted time. A packet is *fully processed* if the processing unit has scheduled the packet for processing for at least its required number of cycles. Each packet may have the following characteristics: (i) *required processing*, i.e., how many processing cycles the packet has to go through before it can be transmitted; (ii) *value*, i.e., how much the packet contributes to the objective function when it is transmitted; (iii) *output port*, i.e., where the packet is headed (in settings with multiple output ports, it is usually assumed that processing occurs independently at each port, so it becomes advantageous to have more busy output ports at a time); and (iv) *size*, i.e., how many slots (bytes) a packet occupies in the buffer. The objective of a buffer management policy is to maximize the total value of transmitted packets. Different settings may assume that some characteristics are uniform.

Competitive Analysis

Competitive analysis provides a uniform throughput guarantee for online algorithms across all traffic patterns. An online algorithm ALG is said to be α-*competitive* with respect to some objective function f (for some $\alpha \geq 1$ which is called the *competitive ratio*) if for any arrival sequence σ the objective function value on the result of ALG is at least $1/\alpha$ times the objective function value on the solution obtained by an offline clairvoyant algorithm, denoted OPT.

Problem 1 (Competitive Ratio) For a given switch architecture, packet characteristics, and an online algorithm ALG in a given setting, prove lower and upper bounds on its competitive ratio with respect to weighted throughput (total value of packets transmitted by an algorithm).

Key Results

Policies and lower and upper bounds on their competitive ratios are outlined according to problem settings; the latter differ in which packet characteristics they assume to be uniform and which are allowed to vary, and additional restrictions may be imposed on admission, processing and/or transmission order, and admissible packet characteristics.

Uniform Processing, Uniform Value, Shared Memory Switch

Since all packets are identical, the problem for a single queue with one output port is trivial. We consider an $M \times N$ shared memory switch that can hold B packets, with a separate processor on each output port. All packets require a single processing cycle and have equal value; the goal is to maximize the number of transmitted packets. Each packet is labeled with an output port where it has to be processed and transmitted.

Non-Push-Out Policies

Kesselman and Mansour [14] show an adversarial logarithmic lower bound: no non-push-out policy can achieve competitive ratio better than $d/2$ for $d = \log_d N$. On the positive side, they present the Harmonic policy that allocates approximately $1/i$ of the buffer to the ith largest queue and, for its variant, the Parametric Harmonic policy, show an upper bound of $c \log_c N + 1$.

Push-Out Policies

The best known policy is Longest Queue Drop (LQD): accept packets greedily if the buffer is not full; if it is, accept the new packet and then drop a packet from the longest queue (destined to the output port with the most packets assigned to it). Aiello et al. [1,10] show that the competitive ratio of LQD is between $\sqrt{2}$ and 2; they also provide nonconstant lower bounds for other popular policies and a general adversarial lower bound of $\frac{4}{3}$ on the competitive ratio of any online algorithm.

Uniform Processing, Uniform Value, Multiple Separated Queues

In an $N \times 1$ switch where each of N input queues has a separate independent buffer of size B, a policy must select which input queue to take a packet from and set admission policies for input queues. For uniform values, the problem was closed by Azar and Litichevskey [3] with a deterministic policy with competitive ratio converging to $\frac{e}{e-1} \approx 1.582$ for arbitrary B; a matching lower bound was shown by Azar and Richter [4].

Uniform Processing, Variable Values, Single Queue

Here, there is only one output port (a single queue), and each packet is fully processed in one cycle; however, packets have different values, making it desirable to drop packets with smaller value and process packets of larger value. It is easy to show that the Priority Queue (PQ) policy that sorts packets with respect to values and pushes out smaller values for larger ones is optimal. Research has concentrated on models with additional constraints: non-push-out policies that are not allowed to push admitted packets out and the FIFO model where packets have to be transmitted in order of arrival. Another important special case considers two possible values: 1 and $V > 1$.

Non-Push-Out Policies

Aiello et al. [2] consider five online policies for the two-valued case, considering the specific cases of $V = 1$, $V = 2$, and $V = \infty$. Andelman, Mansour, and Zhu provide a deterministic policy (Ratio Partition) that achieves optimal $\left(2 - \frac{1}{V}\right)$-competitiveness [26]. In the case of arbitrary values between 1 and $V > 1$, they show that the optimal competitive ratio is $\ln V$, proving tightly matching bounds of $1 + \ln V$ and $2 + \ln V + O(\ln^2 V/B)$ [2,26].

Push-Out Policies

In the FIFO model, there has been a line of adversarial lower bounds culminating in the lower bound of 1.419 shown by Kesselman, Mansour, and van Stee [18] that applies to all algorithms, with a stronger bound of 1.434 for $B = 2$ [2,26]. As for upper bounds, in this simple model the FIFO greedy push-out policy (accept every packet to end of queue, then push out the packet with smallest value if buffer has overflown) has been shown by Kesselman et al. to be 2-competitive [17]; in the two-valued case, they provide an adversarial lower bound of 1.282, and a long line of improvements for the upper bound has led to the optimal Account Strategy policy of Englert and Westermann [6]. They show an adversarial lower bound of $r = \frac{1}{2}(\sqrt{13} - 1) \approx 1.303$ for any $B \geq 2$ and $r_\infty = \sqrt{2} - \frac{1}{2}(\sqrt{5 + 4\sqrt{2}} - 3) \approx 1.282$ for $B \to \infty$ and show that Account Strategy achieves competitive ratio r for arbitrary B and r_∞ for $B \to \infty$. Thus, in the push-out two-valued case, the gap between lower and upper bounds has been closed completely.

Uniform Processing, Variable Values, Multiple Separated Queues

Kawahara et al. [11] consider an $N \times 1$ switch with N separated queues, each of which has a distinct buffer of size B and has a value α_j associated with it, $1 = \alpha_1 \leq \ldots \leq \alpha_N = \alpha$. A policy selects one of N queues, maximizing total transmitted value; [11] provides matching lower and upper bounds for the PQ policy as $1 + \frac{\sum_{j=1}^{n'} \alpha_j}{\sum_{j=1}^{n'+1} \alpha_j}$, where $n' = \arg\max_n \frac{\sum_{j=1}^{n} \alpha_j}{\sum_{j=1}^{n+1} \alpha_j}$, and an adversarial lower bound $1 + \frac{\alpha^3+\alpha^2+\alpha}{\alpha^4+4\alpha^3+3\alpha^2+4\alpha+1}$ for any online algorithm. Azar and Richter [4] show that any r-competitive policy for a FIFO queue with variable values yields a $2r$-competitive policy for multiple queues. Kobayashi et al. [21] show that an r-competitive policy for unit values and multiple queues yields a $\min\left\{Vr, \frac{Vr(2-r)+r^2-2r+2}{V(2-r)+r-1}\right\}$-competitive policy for the two-valued case.

Uniform Processing, Variable Values, Shared Memory Switch

Several output queues, each with a processor, share a buffer of size B, and each unit-sized packet is labeled with an output port

and an intrinsic value from 1 to V. Eugster, Kogan, Nikolenko, and Sirotkin [8] show a $(\sqrt[3]{V} - o(\sqrt[3]{V}))$ lower bound for the LQD (Longest Queue Drop) policy, an $\frac{1}{2}(\min\{V, B\} - 1)$ lower bound for the MVD (Minimal Value Drop) policy, and a $\frac{4}{3}$ lower bound for the MRD (Maximal Ratio Drop) policy.

Uniform Processing, CIOQ Switches

In CIOQ (Combined Input–Output Queued) switches, one maintains at each input a separate queue for each output (also called Virtual Output Queuing, VOQ). To get delay guarantees of an input queuing (IQ) switch closer to those of an output queuing switch (OQ), one usually assumes increased *speedup* S: the switching fabric runs S times faster than each of the input or the output ports. Hence, an OQ switch has a speedup of N (where N is the number of input/output ports), whereas an IQ switch has a speedup of 1; for $1 < S < N$, packets need to be buffered at the inputs before switching as well as at the outputs after switching. This architecture is called a CIOQ switch.

Uniform Values

Consider an $N \times N$ CIOQ switch with speedup S. Packets of equal size arrive at input ports, each labeled with the output port where it has to leave the switch. Each packet is placed in the input queue corresponding to its output port; when it crosses the switch fabric, it is placed in the output queue and resides there until it is sent on the output link. For unit-valued packets, Kesselman and Rosén [15] proposed a non-push-out policy which is 3-competitive for any S and 2-competitive for $S = 1$. Kesselman, Kogan, and Segal [13] show an upper bound of 4 on the competitiveness of a simple greedy policy.

Variable Values

For up to m packet values in $[1, V]$, Kesselman and Rosén [15] show two push-out policies to be $4S$- and $8\min\{m, 2\log V\}$-competitive. Azar and Richter [5] propose a push-out policy β-PG with parameter β; Kesselman et al. [20] show that the competitive ratio of β-PG is at most 7.5 for $\beta = 3$ and at most 7.47 for $\beta = 2.8$. Kesselman and Rosén [16] consider CIOQ switches with PQ buffers (transmit the highest value packet) and show that this policy is 6-competitive for any S.

Uniform Processing, Crossbar Switches

In the buffered crossbar switch architecture, a small buffer is placed on each crosspoint in addition to input and output queues, which greatly simplifies the scheduling process. For packets with unit length and value, Kesselman et al. [20] introduce a greedy switch policy with competitive ratio between $\frac{3}{2}$ and 4 and show a general lower bound of $\frac{3}{2}$ for unit-size buffers. For variable values and PQ buffers, they propose a push-out greedy switch policy with preemption factor β with competitive ratio between $(2\beta-1)/(\beta-1)$ (3.87 for $\beta = 1.53$) and $(\beta + 2)^2 + 2/(\beta - 1)$ (16.24 for $\beta = 1.53$). For variable values and FIFO buffers, they propose a β-push-out greedy switching policy with competitive ratio $6 + 4\beta + \beta^2 + 3/(\beta - 1)$ (19.95 for $\beta = 1.67$) [19].

Uniform Values, Variable Processing, Single Queue

In this setting, each packet contributes one unit to the objective function, but different packets have different processing requirements, i.e., they spend a different number of time slots at the processor. We denote maximal possible required processing by k.

Non-Push-Out Policies

For a single queue and packets with heterogeneous processing, non-push-out policies have not been considered in any detail. Kogan, López-Ortiz, Nikolenko, and Sirotkin [23] have shown that any greedy non-push-out policy is at least $\frac{1}{2}(k + 1)$-competitive. It remains an open problem to find non-push-out policies with sublinear competitive ratios or show that none exists.

Push-Out Policies

Keslassy et al. [12] showed that again, for a single queue, PQ (Priority Queue) that sorts packets with respect to required processing (smallest first) is optimal; research has concentrated on the FIFO case, where packets have to be transmitted in order of arrival. Kogan et al. [24] introduced

lazy policies that process packets down to a single cycle but then delay their transmission until the entire queue consists of such packets; then all packets are transmitted out in as many time slots as there are packets in the queue. In [24], LPO (Lazy Push-Out) was proven to be at most $(\max\{1, \ln k\} + 2 + o(1))$-competitive; [24] also provides a lower bound of $\lfloor \log_B k \rfloor + 1 - O(1/B)$ for both PO (push-out FIFO) and LPO; for large k this bound matches the upper bound up to a factor of $\log B$. Proving a matching upper bound for the PO policy remains an important open problem. In the two-valued case, when packets may have required processing only 1 or k, LPO has a lower bound of $2 - \frac{1}{k}$ and a matching upper bound of $2 + \frac{1}{B}$ [24]. Kogan, López-Ortiz, Nikolenko, and Sirotkin [23] introduce *semi-FIFO* policies, separating processing order from transmission order so that transmission can conform to FIFO constraints while processing order remains arbitrary. Lazy policies thus become a special case of semi-FIFO policies. The authors show a general upper bound of $\frac{1}{B} \log_{\frac{B}{B-1}} k + 3$ on the competitive ratio of any lazy policy and a matching lower bound of $\frac{1}{B} \log_{\frac{B}{B-1}} k + 1$ for several processing orders. In the two-valued case, when processing is only 1 or k, this upper bound improves to $2 + \frac{1}{B}$, so any lazy policy has constant competitiveness. LPQ (Lazy Priority Queue) also falls in the semi-FIFO class; its competitiveness is between $\left(2 - \frac{1}{B}\left\lceil \frac{B}{k} \right\rceil\right)$ and 2 even for arbitrary processing requirements. Kogan et al. [22] consider a generalization with packets of varying size, considering several natural policies and showing an upper bound of $4L$ for one of PO policies, where L is the maximal packet size.

Copying Cost

An important generalization of the heterogeneous processing model was introduced by Keslassy et al. [12]. They attach a penalty α called copying cost to admitting a packet in the queue; thus, the objective function is now $T - \alpha A$, where T is the number of transmitted packets and A is the number of accepted ones, and it becomes less advantageous to push packets out. To deal with copying cost, the authors propose to use β-push-out policies that push a packet out only if its required processing is at least $\beta > 1$ times less than the required processing of a packet which is being pushed out. Keslassy et al. [12] consider the PQ_β policy (Priority Queue with β-push-out) and show that it is at most $\frac{1}{1-\alpha \log_\beta k}\left(1 + \log_{\frac{\beta}{\beta-1}} \frac{k}{2} + 2\log_\beta k\right)(1-\alpha)$-competitive. Kogan, López-Ortiz, Nikolenko, and Sirotkin [23] show that for any processing order, a β-push-out lazy policy LA_β has competitive ratio at most $\left(3 + \frac{1}{B}\log_{\frac{\beta B}{\beta B - 1}} k\right)\frac{1-\alpha}{1-\alpha\log_\beta k}$. They show a lower bound $\frac{1-\alpha}{1-\alpha\log_\beta k}$ on the competitive ratio of any β-push-out policy, which matches the additional factor in the upper bound. In the two-valued case, the upper bound becomes $\left(2 + \frac{1}{B}\right)\frac{1-\alpha}{1-2\alpha}$, and the authors also show a matching lower bound of $\frac{(2B-2)(1-\alpha)}{(B-1)(1-2\alpha)+(1-\alpha)}$.

Uniform Values, Variable Processing, Multiple Separated Queues

Consider k separate queues of size B each; packets with required processing i fall into the ith queue, and the processor chooses which queue to process on a given time slot. Push-out is irrelevant since queues are independent and packets in a queue are identical. Kogan, López-Ortiz, Nikolenko, and Sirotkin [25] show linear lower bounds for several seemingly attractive policies: $\frac{1}{2}\min\{k, B\}$ for LQF (Longest Queue First), k for SQF (Shortest Queue First), $\frac{3k(k+2)}{4k+16}$ for PRR (Packet Round Robin), and an almost linear lower bound of $\frac{k}{H(k)}$, where $H(k) = \sum_{i=1}^{k} \frac{1}{i} \approx \ln k + \gamma$, for CRR (Cycle Round Robin). They introduce a policy called MQF (Minimal Queue First) that processes packets from a nonempty queue with minimal processing requirement. They show that MQF is at least $\left(1 + \frac{k-1}{2k}\right)$-competitive and prove a constant upper bound of 2. For the two-valued case with two queues, 1 and k, Kogan et al. [25] show exactly matching lower and upper bounds for MQF of $1 + \left(1 + \left\lfloor \frac{aB-1}{b} \right\rfloor\right) / \left(B + \left\lceil \frac{1}{a}\left(b\left\lfloor \frac{aB-1}{b} \right\rfloor + 1\right)\right\rceil\right)$.

Uniform Values, Variable Processing, Shared Memory Switch

In this setting, multiple queues with shared memory are implemented in the same way as for

uniform processing and heterogeneous values: there are N output ports, each output port manages a single output queue Q_i, and each output queue collects packets with the same processing requirement (so packets in a given queue are identical).

Non-Push-Out Policies

Eugster, Kogan, Nikolenko, and Sirotkin [8] consider non-push-out policies and show that NHST (Non-Push-Out Harmonic with Static Threshold: $|Q_i|$ is bounded by $\frac{B}{r_i Z}$) is $(kZ + o(kZ))$-competitive, NEST (Non-Push-Out with Equal Static Threshold: $|Q_i|$ is bounded by B/n) is $(N + o(N))$-competitive, NHDT (Non-Push-Out with Harmonic Dynamic Threshold: accept into Q_i if $\sum_{s=1}^{m} |Q_{j_s}| < \frac{B}{H_k}\left(1 + \frac{1}{2} + \ldots + \frac{1}{m}\right)$, where $j_1 \ldots j_m = i$ are queues for which $|Q_j| \geq |Q_i|$) is $(\frac{1}{2}\sqrt{k \ln k} - o(\sqrt{k \ln k}))$-competitive; finding better non-push-out policies is an open problem.

Push-Out Policies

The work [8] also shows lower bounds on the competitive ratio of well-known policies: $(\sqrt{k} - o(\sqrt{k}))$ for LQD (Longest Queue Drop), $(\ln k + \gamma)$ for BQD (Biggest Packet Drop), and $\left(\frac{4}{3} - \frac{6}{B}\right)$ for LWD (Largest Work Drop). The main result of [8] is that LWD is at most 2-competitive.

Open Problems

1. Close the gap between competitive ratios $\frac{4}{3}$ (lower bound for any policy) and 2 (upper bound for LQD) in the uniform processing, uniform value case.
2. Do there exist policies with constant competitive ratio in the uniform processing, variable values, shared memory multiple output queues setting?
3. Do there exist non-push-out policies with sublinear competitive ratio in the case of a single queue with packets with variable processing and uniform values?
4. Prove an upper bound on the competitiveness of PO (push-out) policy in the single-queue FIFO model with heterogeneous required processing and uniform values.
5. Do there exist non-push-out policies with logarithmic competitive ratio in the case of multiple output ports with shared memory that contain packets with variable processing and uniform values?
6. Design efficient policies for CIOQ and crossbar switches with packets with heterogeneous processing and uniform values; prove bounds on their competitive ratios.
7. Design efficient policies and prove bounds on their competitive ratios for the case of packets with both variable values and heterogeneous processing requirements in all of the above settings.

Cross-References

- Packet Switching in Multi-queue Switches
- Packet Switching in Single Buffer

Recommended Reading

1. Aiello W, Kesselman A, Mansour Y (2008) Competitive buffer management for shared-memory switches. ACM Trans Algorithms 5(1):1–16
2. Andelman N, Mansour Y, Zhu A (2003) Competitive queueing policies for QoS switches. In: Proceedings of the 4th annual ACM-SIAM symposium on discrete algorithms, Baltimore, pp 761–770
3. Azar Y, Litichevskey A (2006) Maximizing throughput in multi-queue switches. Algorithmica 45(1):69–90
4. Azar Y, Richter Y (2005) Management of multi-queue switches in QoS networks. Algorithmica 43(1-2):81–96
5. Azar Y, Richter Y (2006) An improved algorithm for CIOQ switches. ACM Trans Algorithms 2(2):282–295
6. Englert M, Westermann M (2009) Lower and upper bounds on FIFO buffer management in QoS switches. Algorithmica 53(4):523–548
7. Epstein L, van Stee R (2004) Buffer management problems. SIGACT News 35(3):58–66
8. Eugster P, Kogan K, Nikolenko SI, Sirotkin AV (2014) Shared-memory buffer management for heterogeneous packet processing. In: Proceedings of the 34th international conference on distributed computing systems, Madrid
9. Goldwasser M (2010) A survey of buffer management policies for packet switches. SIGACT News 41(1):100–128

10. Hahne EL, Kesselman A, Mansour Y (2001) Competitive buffer management for shared-memory switches. In: 13th ACM symposium on parallel algorithms and architectures, Crete Island, pp 53–58
11. Kawahara J, Kobayashi K, Maeda T (2012) Tight analysis of priority queuing policy for egress traffic. CoRR abs/1207.5959
12. Keslassy I, Kogan K, Scalosub G, Segal M (2012) Providing performance guarantees in multipass network processors. IEEE/ACM Trans Netw 20(6):1895–1909
13. Kesselman A, Kogan K, Segal M (2008) Best effort and priority queuing policies for buffered crossbar switches. In: Structural information and communication complexity, 15th international colloquium (SIROCCO 2008), Villars-sur-Ollon, 170–184 http://dx.doi.org/10.1007/978-3-540-69355-0_15
14. Kesselman A, Mansour Y (2004) Harmonic buffer management policy for shared memory switches. Theor Comput Sci 324(2-3):161–182
15. Kesselman A, Rosén A (2006) Scheduling policies for CIOQ switches. J Algorithms 60(1):60–83
16. Kesselman A, Rosén A (2008) Controlling CIOQ switches with priority queuing and in multistage interconnection networks. J Interconnect Netw 9(1/2):53–72
17. Kesselman A, Lotker Z, Mansour Y, Patt-Shamir B, Schieber B, Sviridenko M (2004) Buffer overflow management in QoS switches. SIAM J Comput 33(3):563–583
18. Kesselman A, Mansour Y, van Stee R (2005) Improved competitive guarantees for QoS buffering. Algorithmica 43(1-2):63–80
19. Kesselman A, Kogan K, Segal M (2010) Packet mode and QoS algorithms for buffered crossbar switches with FIFO queuing. Distrib Comput 23(3):163–175
20. Kesselman A, Kogan K, Segal M (2012) Improved competitive performance bounds for CIOQ switches. Algorithmica 63(1–2):411–424
21. Kobayashi K, Miyazaki S, Okabe Y (2009) Competitive buffer management for multi-queue switches in QoS networks using packet buffering algorithms. In: Proceedings of the 21st ACM symposium on parallelism in algorithms and architectures (SPAA), Portland, OR, USA, pp 328–336
22. Kogan K, López-Ortiz A, Nikolenko S, Scalosub G, Segal M (2014) Balancing Work and Size with Bounded Buffers. In: Proceedings of the 6th international conference on communication systems and networks (COMSNETS 2014), Bangalore, pp 1–8
23. Kogan K, López-Ortiz A, Nikolenko SI, Sirotkin AV (2012) A taxonomy of semi-FIFO policies. In: Proceedings of the 31st IEEE international performance computing and communications conference (IPCCC2012), Austin, pp 295–304
24. Kogan K, López-Ortiz A, Nikolenko SI, Sirotkin AV, Tugaryov D (2012) FIFO queueing policies for packets with heterogeneous processing. In: Proceedings of the 1st Mediterranean conference on algorithms (MedAlg 2012), Ein Gedi. Lecture notes in computer science, vol 7659, pp 248–260
25. Kogan K, López-Ortiz A, Nikolenko SI, Sirotkin A (2013) Multi-queued network processors for packets with heterogeneous processing requirements. In: Proceedings of the 5th international conference on communication systems and networks (COMSNETS 2013), Bangalore, pp 1–10
26. Zhu A (2004) Analysis of queueing policies in QoS switches. J Algorithms 53(2):137–168

Single-Source Fully Dynamic Reachability

Camil Demetrescu[1,2] and Giuseppe F. Italiano[1,2]
[1]Department of Computer and Systems Science, University of Rome, Rome, Italy
[2]Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords

Single-source fully dynamic transitive closure

Years and Authors of Summarized Original Work

2005; Demetrescu, Italiano

Problem Definition

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. An algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions and *partially dynamic* if it can handle either edge insertions or edge deletions, but not both.

Given a graph with n vertices and m edges, the *transitive closure* (or *reachability*) problem

consists of building an $n \times n$ Boolean matrix M such that $M[x, y] = 1$ if and only if there is a directed path from vertex x to vertex y in the graph. The fully dynamic version of this problem can be defifined as follows:

Definition 1 (Fully dynamic reachability problem) The *fully dynamic reachability problem* consists of maintaining a directed graph under an intermixed sequence of the following operations:

- `insert`(u,v): insert edge (u,v) into the graph.
- `delete`(u,v): delete edge (u,v) from the graph.
- `reachable`(x,y): return *true* if there is a directed path from vertex x to vertex y, and *false* otherwise.

This entry addresses the *single-source* version of the fully-dynamic reachability problem, where one is only interested in queries with a fixed source vertex s. The problem is defined as follows:

Definition 2 (Single-source fully dynamic reachability problem) The *fully dynamic single-source reachability problem* consists of maintaining a directed graph under an intermixed sequence of the following operations:

- `insert`(u,v): insert edge (u,v) into the graph.
- `delete`(u,v): delete edge (u,v) from the graph.
- `reachable`(y): return *true* if there is a directed path from the source vertex s to vertex y, and *false* otherwise.

Approaches

A simple-minded solution to the problem of Definition would be to keep explicit reachability information from the source to all other vertices and update it by running any graph traversal algorithm from the source s after each insert or delete. This takes $O(m + n)$ time per operation, and then reachability queries can be answered in constant time.

Another simple-minded solution would be to answer queries by running a point-to-point reachability computation, without the need to keep explicit reachability information up to date after each insertion or deletion. This can be done in $O(m + n)$ time using any graph traversal algorithm. With this approach, queries are answered in $O(m + n)$ time and updates require constant time. Notice that the time required by the slowest operation is $O(m+n)$ for both approaches, which can be as high as $O(n^2)$ in the case of dense graphs.

The first improvement upon these two basic solutions is due to Demetrescu and Italiano, who showed how to support update operations in $O(n^{1.575})$ time and reachability queries in $O(1)$ time [1] in a directed acyclic graph. The result is based on a simple reduction of the single-source problem of Definition to the all-pairs problem of Definition. Using a result by Sankowski [2], the bounds above can be extended to the case of general directed graphs.

Key Results

This Section presents a simple reduction presented in [1] that allows it to keep explicit single-source reachability information up to date in subquadratic time per operation in a directed graph subject to an intermixed sequence of edge insertions and edge deletions. The bounds reported in this entry were originally presented for the case of directed acyclic graphs, but can be extended to general directed graphs using the following theorem from [2]:

Theorem 1 *Given a general directed graph with n vertices, there is a data structure for the fully dynamic reachability problem that supports each insertion/deletion in $O(n^{1.575})$ time and each reachability query in $O(n^{0.575})$ time. The algorithm is randomized with one-sided error.*

The idea described in [1] is to maintain reachability information from the source vertex s to all other vertices explicitly by keeping a Boolean array R of size n such that $R[y] = 1$ if and

only if there is a directed path from s to y. An instance D of the data structure for fully dynamic reachability of Theorem is also maintained. After each insertion or deletion, it is possible to update D in $O(n^{1.575})$ time and then rebuild R in $O(n \cdot n^{0.575}) = O(n^{1.575})$ time by letting $R[y] \leftarrow D.\texttt{reachable}(s,y)$ for each vertex y. This yields the following bounds for the single-source fully dynamic reachability problem:

Theorem 2 *Given a general directed graph with n vertices, there is a data structure for the single-source fully dynamic reachability problem that supports each insertion/deletion in $O(n^{1.575})$ time and each reachability query in $O(1)$ time.*

Applications

The graph reachability problem is particularly relevant to the field of databases for supporting transitivity queries on dynamic graphs of relations [3]. The problem also arises in many other areas such as compilers, interactive verification systems, garbage collection, and industrial robotics.

Open Problems

An important open problem is whether one can extend the result described in this entry to maintain fully dynamic single-source shortest paths in subquadratic time per operation.

Cross-References

▸ Trade-Offs for Dynamic Graph Problems

Recommended Reading

1. Demetrescu C, Italiano G (2005) Trade-offs for fully dynamic reachability on dags: breaking through the $O(n^2)$ barrier. J Assoc Comput Mach 52: 147–156
2. Sankowski P (2004) Dynamic transitive closure via dynamic matrix inverse. In: FOCS '04: proceedings of the 45th annual IEEE symposium on foundations of computer science (FOCS'04). IEEE Computer Society, Washington, DC, pp 509–517
3. Yannakakis M (1990) Graph-theoretic methods in database theory. In: Proceedings of the 9-th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, Nashville, pp 230–242

Single-Source Shortest Paths

Seth Pettie
Electrical Engineering and Computer Science (EECS) Department, University of Michigan, Ann Arbor, MI, USA

Keywords

Shortest route; Quickest route

Years and Authors of Summarized Original Work

1999; Thorup

Problem Definition

The *single-source* shortest path problem (SSSP) is, given a graph $G = (V, E, l)$ and a *source* vertex $s \in V$, to find the shortest path from s to every $v \in V$. The difficulty of the problem depends on whether the graph is directed or undirected and the assumptions placed on the length function ℓ. In the most general situation, $l : E \to \mathbb{R}$ assigns arbitrary (positive and negative) real lengths. The algorithms of Bellman-Ford and Edmonds [1, 4] may be applied in this situation and have running times of roughly $O(mn)$, (Edmonds's algorithm works for undirected graphs and presumes that there are no negative length simple cycles.) where $m = |E|$ and $n = |V|$ are the number of edges and vertices. If ℓ assigns only *nonnegative* real edge lengths, then the

algorithms of Dijkstra and Pettie-Ramachandran [4,13] may be applied on directed and undirected graphs, respectively. These algorithms include a *sorting bottleneck* and, in the worst case, take $\Omega(m + n \log n)$ time. (The [13] algorithm actually runs in $O(m + n \log \log n)$ time if the ratio of any two edge lengths is polynomial in n).

A common assumption is that ℓ assigns *integer* edge lengths in the range $\{0, \ldots, 2^w - 1\}$ or $\{-2^{w-1}, \ldots, 2^{w-1} - 1\}$ and that the machine is a w-bit *word RAM*; that is, each edge length fits in one register. For general integer edge lengths, the best SSSP algorithms improve on Bellman-Ford and Edmonds by a factor of roughly $\sqrt{n}$ [6]. For nonnegative integer edge lengths, the best SSSP algorithms are faster than Dijkstra and Pettie-Ramachandran by up to a logarithmic factor. They are frequently based on integer priority queues [9].

Key Results

Thorup's primary result [16] is an optimal linear time SSSP algorithm for undirected graphs with integer edge lengths. This is the first and only linear time shortest path algorithm that does not make serious assumptions on the class of input graphs.

Theorem 1 *There is a SSSP algorithm for integer-weighted undirected graphs that runs in $O(m)$ time.*

Thorup avoids the sorting bottleneck inherent in Dijkstra's algorithm by precomputing (in linear time) a *component hierarchy*. The algorithm of [16] operates in a manner similar to Dijkstra's algorithm [4] but uses the component hierarchy to identify groups of vertices that can be visited in any order. In later work, Thorup [17] extended this approach to work when the edge lengths are floating-point numbers. (There is some flexibility in the definition of *shortest path* since floating-point addition is neither commutative nor associative).

Thorup's hierarchy-based approach has since been extended to directed and/or real-weighted graphs and to solve the *all pairs* shortest path (APSP) problem [11–13]. The generalizations to related SSSP problems are summarized below. See [11,12] for hierarchy-based APSP algorithms.

Theorem 2 (Hagerup [8], 2000) *A component hierarchy for a directed graph $G = (V, E, l)$, where $l : E \to \{0, \ldots, 2^w - 1\}$, can be constructed in $O(m \log w)$ time. Thereafter, SSSP from any source can be computed in $O(m + n \log \log n)$ time.*

Theorem 3 (Pettie and Ramachandran [13], 2005) *A component hierarchy for an undirected graph $G = (V, E, l)$, where $l : E \to \mathbb{R}^+$, can be constructed in $O(m\alpha(m, n) + \min\{n \log \log r, n \log n\})$ time, where r is the ratio of the maximum-to-minimum edge length. Thereafter, SSSP from any source can be computed in $O(m \log \alpha(m, n))$ time.*

The algorithms of Hagerup [8] and Pettie-Ramachandran [13] take the same basic approach as Thorup's algorithm: use some kind of component hierarchy to identify groups of vertices that can safely be visited in any order. However, the assumption of directed graphs [8] and real edge lengths [13] renders Thorup's hierarchy inapplicable or inefficient. Hagerup's component hierarchy is based on a directed analogue of the minimum spanning tree. The Pettie-Ramachandran algorithm enforces a certain degree of balance in its component hierarchy and, when computing SSSP, uses a specialized priority queue to take advantage of this balance.

Applications

Shortest path algorithms are frequently used as a subroutine in other optimization problems, such as flow and matching problems [1] and facility location [18]. A widely used commercial application of shortest path algorithms is finding efficient routes on road networks, e.g., as provided by Google Maps, MapQuest, or Yahoo Maps.

Open Problems

Thorup's SSSP algorithm [16] runs in linear time and is therefore optimal. The main open problem is to find a linear time SSSP algorithm that works on *real*-weighted *directed* graphs. For real-weighted undirected graphs, the best running time is given in Theorem 3. For integer-weighted directed graphs, the fastest algorithms are based on Dijkstra's algorithm (not Theorem 2) and run in $O(m\sqrt{\log\log n})$ time (randomized) and deterministically in $O(m + n\log\log n)$ time.

Problem 1 Is there an $O(m)$ time SSSP algorithm for integer-weighted directed graphs?

Problem 2 Is there an $O(m) + o(n\log n)$ time SSSP algorithm for real-weighted graphs, either directed or undirected?

The complexity of SSSP on graphs with positive and negative edge lengths is also open.

Experimental Results

Asano and Imai [2] and Pettie et al. [14] evaluated the performance of the hierarchy-based SSSP algorithms [13, 16]. There have been a number of studies of SSSP algorithms on integer-weighted directed graphs; see [7] for the latest and references to many others. The trend in recent years is to find practical preprocessing schemes that allow for very quick point-to-point shortest path queries. See [3, 10, 15] for recent work in this area.

Data Sets

See [5] for a number of US and European road networks.

URL to Code

See [5].

Cross-References

▸ All Pairs Shortest Paths via Matrix Multiplication

Recommended Reading

1. Ahuja RK, Magnati TL, Orlin JB (1993) Network flows: theory, algorithms, and applications. Prentice Hall, Englewood Cliffs
2. Asano Y, Imai H (2000) Practical efficiency of the linear-time algorithm for the single source shortest path problem. J Oper Res Soc Jpn 43(4):431–447
3. Bast H, Funke S, Matijevic D, Sanders P, Schultes D (2007) In transit to constant shortest-path queries in road networks. In: Proceedings 9th workshop on algorithm engineering and experiments (ALENEX), New Orleans
4. Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms. MIT, Cambridge
5. Demetrescu C, Goldberg AV, Johnson D (2006) 9th DIMACS implementation challenge-shortest paths. http://www.dis.uniroma1.it/~challenge9/
6. Goldberg AV (1995) Scaling algorithms for the shortest paths problem. SIAM J Comput 24(3):494–504
7. Goldberg AV (2001) Shortest path algorithms: engineering aspects. In: Proceedings of the 12th international symposium on algorithms and computation (ISAAC), Christchurch. LNCS, vol 2223. Springer, Berlin, pp 502–513
8. Hagerup T (2000) Improved shortest paths on the word RAM. In: Proceedings of the 27th international colloquium on automata, languages, and programming (ICALP), Geneva. LNCS, vol 1853. Springer, Berlin, pp 61–72
9. Han Y, Thorup M (2002) Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In: Proceedings of the 43rd symposium on foundations of computer science (FOCS), Vancouver, pp 135–144
10. Knopp S, Sanders P, Schultes D, Schulz F, Wagner D (2007) Computing many-to-many shortest paths using highway hierarchies. In: Proceedings of the 9th workshop on algorithm engineering and experiments (ALENEX), New Orleans
11. Pettie S (2002) On the comparison-addition complexity of all-pairs shortest paths. In: Proceedings of the 13th international symposium on algorithms and computation (ISAAC), Vancouver, pp 32–43
12. Pettie S (2004) A new approach to all-pairs shortest paths on real-weighted graphs. Theor Comput Sci 312(1):47–74
13. Pettie S, Ramachandran V (2005) A shortest path algorithm for real-weighted undirected graphs. SIAM J Comput 34(6):1398–1431
14. Pettie S, Ramachandran V, Sridhar S (2002) Experimental evaluation of a new shortest path algorithm. In: Proceedings of the 4th workshop on algorithm

engineering and experiments (ALENEX), San Francisco, pp 126–142
15. Sanders P, Schultes D (2006) Engineering highway hierarchies. In: Proceedings of the 14th European symposium on algorithms (ESA), Zurich, pp 804–816
16. Thorup M (1999) Undirected single-source shortest paths with positive integer weights in linear time. J ACM 46(3):362–394
17. Thorup M (2000) Floats, integers, and single source shortest paths. J Algorithms 35:189–201
18. Thorup M (2003) Quick and good facility location. In: Proceedings of the 14th annual ACM-SIAM symposium on discrete algorithms (SODA), Baltimore, pp 178–185

Ski Rental Problem

Mark S. Manasse
Microsoft Research, Mountain View, CA, USA

Keywords

Metrical task systems; Oblivious adversaries; Worst-case approximation

Years and Authors of Summarized Original Work

1990; Karlin, Manasse, McGeogh, Owicki

Problem Definition

The ski rental problem was developed as a pedagogical tool for understanding the basic concepts in some early results in online algorithms. (In the interest of full disclosure, the earliest presentations of these results described the problem as the wedding-tuxedo-rental problem. Objections were presented that this was a gender-biased name for the problem, since while groomsmen can rent their wedding apparel, bridesmaids usually cannot. A further complication, owing to the difficulty of instantaneously producing fitted garments or ski equipment outlined below, suggests that some complications could have been avoided by focusing on the dilemma of choosing between daily lift passes or season passes, although this leads to the pricing complexities of purchasing season passes well in advance of the season, as opposed to the higher cost of purchasing them at the mountain during the ski season. A similar problem could be derived from the question as to whether to purchase the daily newspaper at a newsstand or to take a subscription, after adding the challenge that one's peers will treat one contemptuously if one has not read the news on days on which they have.) The ski rental problem considers the plight of one consumer who, in order to socialize with peers, is forced to engage in a variety of athletic activities, such as skiing, bicycling, windsurfing, rollerblading, sky diving, scuba-diving, tennis, soccer, and ultimate Frisbee, each of which has a set of associated apparatus, clothing, or protective gear.

In all of these, it is possible either to purchase the accoutrements needed or to rent them. For the purpose of this problem, it is assumed that one-time rental is less expensive than purchasing. It is also assumed that purchased items are durable, and suitable for reuse for future activities of the same type without further expense, until the items wear out (which occurs at the same rate for all users), are outgrown, become unfashionable, or are disposed of to make room for other purchased items. The social consumer must make the decision to rent or buy for each event, although it is assumed that the consumer is sufficiently parsimonious as to abjure rental if already in possession of serviceable purchased equipment. Whether purchases are as easy to arrange as rentals, or whether some advance planning is required (e.g., to mount bindings on a ski) is a further detail considered in this problem. It is assumed that the social consumer has no particular independent interest in these activities, and engages in these activities only to socialize with peers who choose to engage in these activities disregarding the consumer's desires.

These putative peers are more interested in demonstrating the superiority of their financial acumen to that of the social consumer in question than they are in any particular activity. To that end, the social consumer is taunted mercilessly based on the ratio of his/her total expenses on

rentals and purchases to theirs. Consequently, the peers endeavor to invite the social consumer to engage in events while they are costly to him/her, and once the activities are free to the social consumer, if continued activity would be costly to them, cease. But, to present an illusion of fairness, skis, both rented and purchased, have the same cost for the peers as they do for the social consumer in question. The ski rental problem takes a very restricted setting. It assumes that purchased ski equipment never needs replacement, and that there are no costs to a ski trip other than the skis (thus, no cost for the gasoline, for the lift and/or speeding tickets, for the hot chocolates during skiing, or for the après-ski liqueurs and meals). It is assumed that the social consumer experiences no physical disabilities preventing him/her from skiing and has no impending restrictions to his/her participation in ski trips (obviously, a near-term-fatal illness or an anticipated conviction leading to confinement for life in a penitentiary would eliminate any potential interest in purchasing alpine equipment – when the ratio of purchase to rental exceeds the maximum need for equipment, one should always rent). It is assumed that the social consumer's peers have disavowed any interest in activities other than skiing, and that the closet, basement, attic, garage, or storage locker included in the social consumer's rent or mortgage (or necessitated by other storage needs) has sufficient capacity to hold purchased ski equipment without entailing the disposal of any potentially useful items. Bringing these complexities into consideration brings one closer to the hardware-based problems which initially inspired this work.

The impact of invitations issued with sufficient time allowed for purchasing skis, as well as those without, will be considered.

Given all of that, what ratio of expenses can the social consumer hope to attain? What ratio can the social consumer not expect to beat? These are the basic questions of competitive analysis.

The impact of keeping secrets from one's peers is further considered. Rather than a fixed strategy for when to purchase skis, the social consumer may introduce an element of chance into the process. If the peers are able to observe his/her ski equipment and notice when it changes from rented skis to purchased skis, and change their schedule for alpine recreation in light of this observation, randomness provides no advantages. If, on the other hand, the social consumer announces to the peers, in advance of the first trip, how he/she will decide when the time is right for purchasing skis, including any use of probabilistic techniques, and they then decide on the schedule for ski trips for the coming winter, a deterministic decision procedure generally produces a larger competitive ratio than does a randomized procedure.

Key Results

Given an unbounded sequence of skiing trips, one should eventually purchase skis if the cost of renting skis, r, is positive. In particular, let the cost of purchasing skis be some number $p \geq r$. If one never intends to make a purchase, one's cost for the season will be rn, where n is the number of ski trips in which one participates. If n exceeds p/r, one's cost will exceed the price of purchasing skis; as n continues to increase, the ratio of one's costs to those of one's peers increases to nr/p, which grows unboundedly with n, since your peers, knowing that n exceeds p/r, will have purchased skis prior to the first trip.

On the other hand, if one rushes out to purchase skis upon being told that the ski season is approaching, one's peers will decide that this season looks inopportune, and that skiing is passé, leaving their costs at zero, and one's costs at p, leaving an infinite ratio between one's costs and theirs; if one chooses to defer the purchase until after one's first ski trip, this produces the less unfavorable ratio p/r or $1 + p/r$, depending on whether the invitation left one time to purchase skis before the first trip or not.

Suppose one chooses, instead, to defer one's purchase until after one has made k rentals, but before ski trip $k+1$. One's costs are then bounded by $kr + p$. After k ski trips, the cost to one's peers will be the lesser of kr and p (as one's peers will have decided whether to rent or buy for

the season upon knowing one's plans, which in this case amounts to knowing k), for a ratio equal to the larger of $1 + kr/p$ and $1 + p/kr$. Were they to choose to terminate the activity earlier (so $n < k$), the ratio would be only the greater of kr/p and 1, which is guaranteed to be less than the sum of the two – one's peers would be shirking their opportunity to make one's behavior look foolish were they to allow one to stop skiing prior to one's purchase of a pair of skis!

It is certain, since kr/p and p/kr are reciprocals, that one of them is at least equal to 1, ensuring that one will be compelled to spend at least twice as much as one's peers.

The analysis above applies to the case where ski trips are announced without enough warning to leave one time to buy skis. Purchases in that case are not instantaneous; in contrast, if one is able to purchase skis on demand, the cost to one's peers changes to the lesser of $(k+1)r$ and p. The overall results are not much different; the ratio choices become the larger of $1 + kr/p$ and $1 + (p - r)/((k + 1)r)$.

When probabilistic algorithms are considered with oblivious frenemies (those who know the way in which random choices will affect one's purchasing decisions, but who do not take time to notice that one's skis are no longer marked with the name and phone number of a rental agency), one can appear more thrifty.

A randomized algorithm can be viewed as a distribution over deterministic algorithms. No good algorithm can purchase skis prior to the first invitation, lest it exhibit infinite regrettability (some positive cost compared to zero). A good algorithm must purchase skis by the time one's peers will have; otherwise, one's cost ratio continues to increase with the number of ski trips. Moreover, the ratio should be the same after every ski trip; if not, then there is an earliest ratio not equal to the largest, and probabilities can be adjusted to change this earliest ratio to be closer to the largest while decreasing all larger ratios.

Consider, for example, the case of $p = 2r$, with purchases allowed at the time of an invitation. The best deterministic ratio in this case is 1.5. It is only necessary to choose a probability q, the probability of purchasing at the time of the first invitation. The cost after one trip is then $(1-q)r+2qr = r(1+q)$, for a ratio of $1+q$, and after two trips the cost is $q(2r) + (1 - q)(3r) = 3 - q)r$, producing a ratio of $(3 - q)/2$. Setting these to be equal yields $q = 1/3$, for a ratio of 4/3.

If insufficient time is allowed for purchases before skiing, the best deterministic ratio is 2. Purchasing after the first ski trip with probability q (and after the second with probability $1 - q$) leads to expected costs of $(1-q)r+3qr = r(1+2q)$ after the first trip, and $(1-q)(2+2)r+3qr = r(4 + q)$, leading to a ratio of $2 - q/2$. Setting $1 + 2q = 2 - q/2$ yields $q = 2/5$, for a ratio of 9/5.

More careful analysis, for which readers are referred to the references and the remainder of this volume, shows that the best achievable ratio approaches $\epsilon/(\epsilon - 1) \approx 1.58197$ as p/r increases, approaching the limit from below if sufficient warning time is offered, and from above otherwise.

Applications

The primary initial results were directed towards problems of computer architecture; in particular, design questions for capacity conflicts in caches, and shared memory design in the presence of a shared communication channel. The motivation for these analyses was to find designs which would perform reasonably well on as-yet-unknown workloads, including those to be designed by competitors who may have chosen alternative designs which favor certain cases. While it is probably unrealistic to assume that precisely the least-desirable workloads will occur in ordinary practice, it is not unreasonable to assume that extremal workloads favoring either end of a decision will occur.

History and Further Reading

This technique of finding algorithms with bounded worst-case performance ratios is common in analyzing approximation algorithms.

The initial proof techniques used for such analyses (the method of amortized analysis) were first presented by Sleator and Tarjan.

The reader is advised to consult the remainder of this volume for further extensions and applications of the principles of competitive online algorithms.

Cross-References

- ▸ Algorithm DC-TREE for k-Servers on Trees
- ▸ Metrical Task Systems
- ▸ Online List Update
- ▸ Online Paging and Caching
- ▸ Work-Function Algorithm for k-Servers

Recommended Reading

1. Karlin AR, Manasse MS, Rudolph L, Sleator DD (1988) Competitive snoopy caching. Algorithmica 3:77–119 (Conference version: FOCS 1986, pp 244–254)
2. Karlin AR, Manasse MS, McGeoch LA, Owicki SS (1994) Competitive randomized algorithms for nonuniform problems. Algorithmica 11(6):542–571 (Conference version: SODA 1990, pp 301–309)
3. Reingold N, Westbrook J, Sleator DD (1994) Randomized competitive algorithms for the list update problem. Algorithmica 11(1):15–32 (Conference version included author Irani S: SODA 1991, pp 251–260)

Slicing Floorplan Orientation

Evangeline F.Y. Young
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China

Keywords

Shape curve computation

Years and Authors of Summarized Original Work

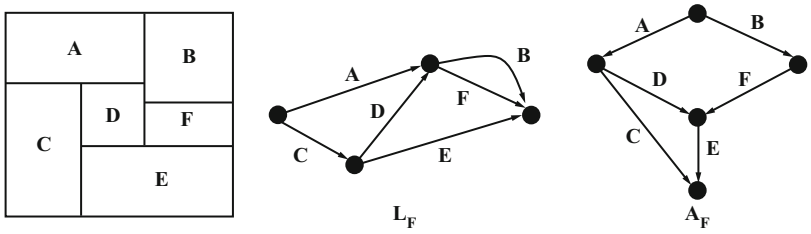
1983; Stockmeyer

Problem Definition

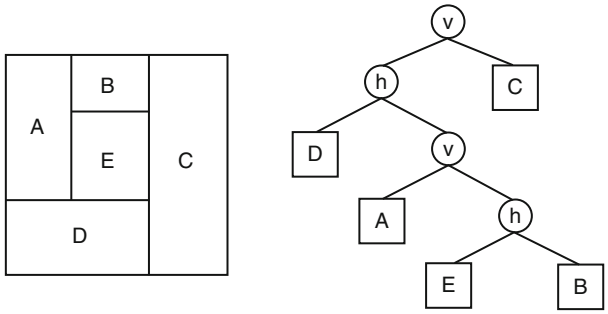
This problem is about finding the optimal orientations of the cells in a slicing floorplan to minimize the total area. In a floorplan, cells represent basic pieces of the circuit which are regarded as indivisible. After performing an initial placement, for example, by repeated application of a min-cut partitioning algorithm, the relative positions between the cells on a chip are fixed. Various optimizations can then be done on this initial layout to optimize different cost measures such as chip area, interconnect length, routability, etc. One such optimization, as mentioned in Lauther [3], Otten [4], and Zibert and Saal [13], is to determine the best orientation of each cell to minimize the total chip area. This work by Stockmeyer [8] gives a polynomial time algorithm to solve the problem optimally in a special type of floorplans called *slicing floorplans* and shows that this orientation optimization problem in general non-slicing floorplans is NP-complete.

Slicing Floorplan

A floorplan consists of an enclosing rectangle subdivided by horizontal and vertical line segments into a set of non-overlapping *basic rectangles*. Two different line segments can meet but not cross. A floorplan F is characterized by a pair of planar acyclic directed graphs A_F and L_F defined as follows. Each graph has one source and one sink. The graph A_F captures the "above" relationships and has a vertex for each horizontal line segment, including the top and the bottom of the enclosing rectangle. For each basic rectangle R, there is an edge e_r directed from segment σ to segment σ' if and only if σ (or part of σ) is the top of R and σ' (or part of σ') is the bottom of R. There is a one-to-one correspondence between the basic rectangles and the edges in A_F. The graph L_F is defined similarly for the



Slicing Floorplan Orientation, Fig. 1 A floorplan F and its A_F and L_F representing the above and left relationships



Slicing Floorplan Orientation, Fig. 2 A slicing floorplan F and its slicing tree representation

"left" relationships of the vertical segments. An example is shown in Fig. 1. Two floorplans F and G are equivalent if and only if $A_F = A_G$ and $L_F = L_G$. A floorplan F is slicing if and only if both its A_F and L_F are series parallel.

Slicing Tree

A slicing floorplan can also be described naturally by a rooted binary tree called *slicing tree*. In a slicing tree, each internal node is labeled by either an h or a v, indicating a horizontal or a vertical slice respectively. Each leaf corresponds to a basic rectangle. An example is shown in Fig. 2. There can be several slicing trees describing the same slicing floorplan, but this redundancy can be removed by requiring the label of an internal node to differ from that of its right child [12]. For the algorithm presented in this work, a tree of smallest depth should be chosen, and this depth minimization process can be done in $O(n \log n)$ time using the algorithm by Golumbic [2].

Orientation Optimization

In optimization of a floorplan layout, some freedom in moving the line segments and in choosing the dimensions of the rectangles are allowed. In the input, each basic rectangle R has two positive integers a_R and b_R, representing the dimensions of the cell that will be fit into R. Each cell has two possible orientations resulting in either the side of length a_R or b_R being horizontal. Given a floorplan F and an orientation p, each edge e in A_F and L_F is given a label $l(e)$ representing the height or the width of the cell corresponding to e depending on its orientation. Define an (F, ρ)-*placement* to be a labeling l of the vertices in A_F and L_F such that (i) the sources are labeled by zero and (ii) if e is an edge from vertex σ to $\sigma', l(\sigma') \geq l(\sigma) + l(e)$. Intuitively, if σ is a horizontal segment, $l(\sigma)$ is the distance of σ from the top of the enclosing rectangle, and the inequality constraint ensures that the basic rectangle corresponding to e is tall enough for the cell contained in it and similarly for the vertical segments. Now, $h_F(\rho)$ (resp. $w_F(\rho)$) is defined to be the minimum label of the sink in $A_F(\rho)$ (resp. $L_F(\rho)$) over all (F, ρ)-placements, where $A_F(\rho)$ (resp. $L_F(\rho)$) is obtained from A_F (resp. L_F) by labeling the edges and vertices as described above. Intuitively, $h_F(\rho)$ and $w_F(\rho)$ give the minimum height and width of a floorplan F given an orientation ρ of all the cells such that each cell fits well into its associated basic rectangle.

The orientation optimization problem can be defined formally as follows:

Problem 1 (Orientation Optimization Problem for Slicing Floorplan)
INPUT: A slicing floorplan F of n cells described by a slicing tree T, the widths and heights of the cells a_i and b_i for $i = 1...n$, and a cost function $\psi(h, w)$.
OUTPUT: An orientation ρ of all the cells that minimizes the objective function $\psi(h_F(\rho), w_F(\rho))$ over all orientations ρ.

For this problem, Lauther [3] has suggested a greedy heuristic. Zibert and Saal [13] use integer programming methods to do rotation optimization and several other optimization simultaneously for general floorplans. In the following sections, an efficient algorithm will be given to solve the problem optimally in $O(nd)$ time where n is the number of cells and d is the depth of the given slicing tree.

Key Results

In the following algorithm, $F(u)$ denotes the floorplan described by the subtree rooted at u in the given slicing tree T, and let $L(u)$ be the set of leaves in that subtree. For each node u of T, the algorithm constructs recursively a list of pairs:

$$\{(h_1, w_1), (h_2, w_2), \ldots, (h_m, w_m)\}$$

where (1) $m \leq |L(u)| + 1$, (2) $h_i > h_{i+1}$ and $w_i < w_{i+1}$ for $i = 1...m - 1$, (3) there is an orientation ρ of the cells in $L(u)$ such that $(h_i, w_i) = (h_F(u)(\rho), w_F(u)(\rho))$ for each $i = 1...m$, and (4) for each orientation ρ of the cells in $L(u)$, there is a pair (h_i, w_i) in the list such that $h_i \leq h_{F(u)}(\rho)$ and $w_i \leq w_{F(u)}(\rho)$.

$L(u)$ is thus a non-redundant list of all possible dimensions of the floorplan described by the subtree rooted at u. Since the cost function ψ is non-decreasing, it can be minimized over all orientations by finding the minimum $\psi(h_i, w_i)$ over all the pairs (h_i, w_i) in the list constructed at the root of T. At the beginning, a list is constructed at each leaf node of T representing the possible dimensions of the cell. If a leaf cell has dimensions a and b with $a > b$, the list is $\{(a, b), (b, a)\}$. If $a = b$, there will just be one pair (a, b) in the list. (If the cell has a fixed orientation, there will also be just one pair as defined by the fixed orientation.) Notice that the condition (1) above is satisfied in these leaf node lists. The algorithm then works its way up the tree and constructs the list at each node recursively. In general, assume that u is an internal node with children v and v' and u represents a vertical slice. Let $\{(h_1, w_1) \ldots (h_k, w_k)\}$ and $\{(h'_1, w'_1) \ldots (h'_m, w'_m)\}$ be the lists at v and v' respectively where $k \leq |L(v)| + 1$ and $m \leq |L(v')| + 1$. A pair (h_i, wi) from v can be put together by a vertical slice with a pair (h'_j, w'_j) from v' to give a pair:

$$\text{join}((h_i, w_i), (h'_j, w'_j)) = (\max(h_i, h'_j), w_i + w'_j)$$

in the list of u (see Fig. 3). The key fact is that most of the km pairs are sub-optimal and do not need to be considered. For example, if $h_i > h'_j$, there is no need to join (h_i, w_i) with (h'_z, w'_z) for any $z > j$ since

$$\max(h_i, h'_z) = \max(h_i, h'_j) = h_i,$$
$$w_i + w'_z > w_i + w'_j$$

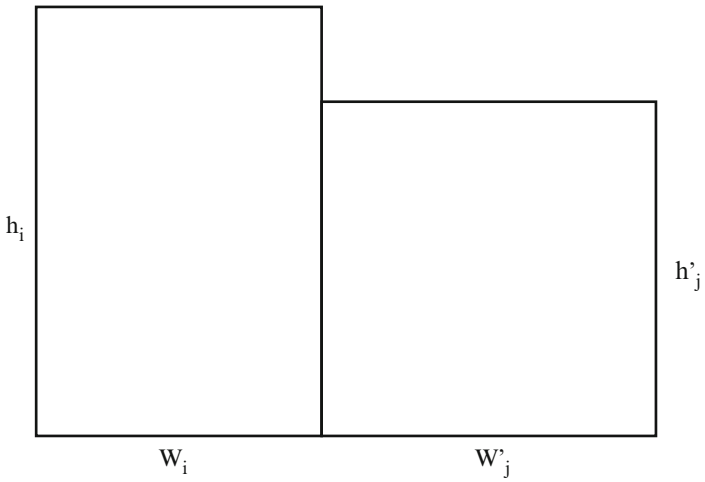
Similarly, if node u represents a horizontal slice, the join operation will be

$$\text{join}((h_i, w_i), (h'_j, w'_j)) = (h_i + h'_j, \max(w_i, w'_j))$$

The algorithm also keeps two pointers for each element in the lists in order to construct back the optimal orientation at the end. The algorithm is summarized by the following pseudocode:

Pseudocode *Stockmeyer()*

1. Initialize the list at each leaf node.
2. Traverse the tree in postorder. At each internal node u with children v and v', construct a list at node u as follows:
3. Let $\{(h_1, w_1) \ldots (h_k, w_k)\}$ and $\{(h'_1, w'_1) \ldots (h'_m, w'_m)\}$ be the lists at v and v' respectively.
4. Initialize i and j to one.



Slicing Floorplan Orientation, Fig. 3 An illustration of the merging step

5. If $i > k$ or $j > m$, the whole list at u is constructed.
6. Add join$((h_i, w_i), (h'_j, w'_j))$ to the list with pointers pointing to (h_i, w_i) and (h'_j, w'_j) in $L(v)$ and $L(v')$ respectively.
7. If $h_i > h'_j$, increment i by 1.
8. If $h_i > h'_j$, increment j by 1.
9. If $h_i > h'_j$, increment both i and j by 1.
10. Go to step 5
11. Compute $\psi(h_i, w_i)$ for each pair (h_i, w_i) in the list L_r at the root r of T.
12. Return the minimum $\psi(h_i, w_i)$ for all (h_i, w_i) in L_r and construct back the optimal orientation by following the pointers.

Correctness

The algorithm is correct since at each node u, a list is constructed that records all the possible non-redundant dimensions of the floorplan described by the subtree rooted at u. This can be proved easily by induction starting from the leaf nodes and working up the tree recursively. Since the cost function ψ is non-decreasing, it can be minimized over all orientations of the cells by finding the minimum $\psi(h_i, w_i)$ over all the pairs (h_i, w_i) in the list L_r constructed at the root r of T.

Runtime

At each internal node u with children v and v'. If the lengths of the lists at v and v' are k and m respectively, the time spent at u to combine the two lists is $O(k + m)$. Each possible dimension of a cell will thus invoke one unit of execution time at each node on its path up to the root in the postorder traversal. The total runtime is thus $O(d \times N)$ where N is the total number of realizations of all the n cells, which is equal to $2n$ in the orientation optimization problem. Therefore, the runtime of this algorithm is $O(nd)$.

Theorem 1 *Let $\psi(h, w)$ be non-decreasing in both arguments, i.e., if $h \leq h'$ and $w \leq w'$, $\psi(h, w) \leq \psi(h', w')$, and computable in constant time. For a slicing floorplan F described by a binary slicing tree T, the problem of minimizing $\psi(h_F(\rho), w_F(\rho))$ over all orientations ρ can be solved in time $O(nd)$ time, where n is the number of leaves of T (equivalently, the number of cells of F) and d is the depth of T.*

Applications

Floorplan design is an important step in the physical design of VLSI circuits. Stockmeyer's optimal orientation algorithm [8] has been generalized to solve the area minimization problem in slicing floorplans [7], in hierarchical non-slicing floorplans of order five [6,9], and in general floorplans [5]. The floorplan area minimization problem is similar except that each *soft cell* now has a number of possible realizations, instead of just two different orientations. The same technique

can be applied immediately to solve optimally the area minimization problem for slicing floorplans in $O(nd)$ time where n is the total number of realizations of all the cells in a given floorplan F and d is the depth of the slicing tree of F. Shi [7] has further improved this result to $O(n \log n)$ time. This is done by storing the list of non-redundant pairs at each node in a balanced binary search tree structure called *realization tree* and using a new merging algorithm to combine two such trees to create a new one. It is also proved in [7] that this $O(n \log n)$ time complexity is the lower bound for this area minimization problem in slicing floorplans.

For hierarchical non-slicing floorplans, Pan et al. [6] prove that the problem is NP-complete. Branch-and-bound algorithms are developed by Wang and Wong [9], and pseudopolynomial time algorithms are developed by Wang and Wong [10] and Pan et al. [6]. For general floorplans, Stockmeyer [8] has shown that the problem is strongly NP-complete. It is therefore unlikely to have any pseudopolynomial time algorithm. Wimer et al. [11] and Chong and Sahni [1] propose branch-and-bound algorithms. Pan et al. [5] develop algorithms for general floorplans that are approximately slicing.

Recommended Reading

1. Chong K, Sahni S (1993) Optimal realizations of floorplans. IEEE Trans Comput Aided Des 12(6):793–901
2. Golumbic MC (1976) Combinatorial merging. IEEE Trans Comput C-25:1164–1167
3. Lauther U (1980) A Min-cut placement algorithm for general cell assemblies based on a graph representation. J Digit Syst 4:21–34
4. Otten RHJM (1982) Automatic floorplan design. In: Proceedings of the 19th design automation conference, Las Vegas, pp 261–267
5. Pan P, Liu CL (1995) Area minimization for floorplans. IEEE Trans Comput Aided Des 14(1):123–132
6. Pan P, Shi W, Liu CL (1996) Area minimization for hierarchical floorplans. Algorithmica 15(6):550–571
7. Shi W (1996) A fast algorithm for area minimization of slicing floorplan. IEEE Trans Comput Aided Des 15(12):1525–1532
8. Stockmeyer L (1983) Optimal orientations of cells in slicing floorplan designs. Infect Control 59:91–101
9. Wang TC, Wong DF (1992) Optimal floorplan area optimization. IEEE Trans Comput Aided Des 11(8):992–1002
10. Wang TC, Wong DF (1993) A note on the complexity of Stockmeyer's floorplan optimization technique. In: Algorithmic aspects of VLSI layout. Lecture notes series on computing, vol 2. World Scientific, Singapore, pp 309–320
11. Wimer S, Koren I, Cederbaum I (1989) Optimal aspect ratios of building blocks in VLSI. IEEE Trans Comput Aided Des 8(2):139–145
12. Wong DF, Liu CL (1986) A new algorithm for floorplan design. In: Proceedings of the 23rd ACM/IEEE design automation conference, Las Vegas, pp 101–107
13. Zibert K, Saal R (1974) On computer aided hybrid circuit layout. In: Proceedings of the IEEE international symposium on circuits and systems, San Francisco, pp 314–318

Sliding Window Algorithms

Vladimir Braverman
Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

Keywords

Data streams; Histograms; Randomized algorithms; Sampling; Sketching

Years and Authors of Summarized Original Work

2007; Braverman, Ostrovsky

Problem Definition

In the last decade, the theoretical study of the sliding window model was developed to advance applications with very large input and time-sensitive output. In some practical situations, input might be seen as an ordered sequence, and it is useful to restrict computations to recent portions of the input. Examples include the analysis of recent tweets and time series of the stock market.

To address the aforementioned practical situations, Datar et al. [20] introduced the sliding window model that assumes that the input is a stream (i.e., the ordered sequence) of data elements and divides the data elements into two categories: *active* elements and *expired* elements. Typically, a recent portion (i.e., a suffix) of the stream defines the window of active elements, and the reminder (i.e., a complimenting prefix) of the stream defines the set of expired elements. When a new data element arrives, the set of active elements expands to include the new element, but the set might also shrink by discarding some portion of oldest active elements. This process of additions and expirations reminds one of the movements of an interval (or a window) along a line and explains the name of the model. The number of active elements N is often called a *size of the sliding window*. There are two popular variants of the sliding window model. The variant of a *sequence-based* window fixes the number of active elements N, and every insertion (or arrival) of a new element corresponds to a deletion (or expiration) of the oldest active element (after the size of the stream becomes larger than N). For example, a sequence-based window on a stream of IP packets is a set of last N packets. The variant of a *timestamp-based* window associates each element with a nondecreasing timestamp, and the window contains all elements with timestamps larger than a certain value. Thus, there is no obvious dependence between the number of elements that arrive and expire. In the previous example, the timestamp-based window might be defined as a set of all packets that arrived within the last t seconds.

Formal Definition

We denote the stream D by a sequence of elements $\{p_i\}_{i=1}^{m}$ where $p_i \in [n]$. It is important to note that m is incremented for each new arrival. A *bucket* $B(x, y) = \{p_i, i \in [x, y]\}$ is the set of all stream elements between p_x and p_y, inclusively. A sequence-based window is defined $W = B(m - N + 1, m)$ where N is a predefined parameter. Consider a nondecreasing *timestamp* function $T : [m] \to R$ and let t be a parameter. Given T and t, a timestamp-based window is defined as $W = B(l(t), m)$ where $l(t) = \min\{i : T(i) \geq T(m) - t\}$. Consider function f that is defined on buckets. An algorithm maintains a $(1 \pm \epsilon)$-approximation of f on W if, at any moment, the algorithm outputs X s.t. $|f(W) - X| \leq \epsilon f(W)$. Similarly, a randomized algorithm maintains a $(1 \pm \epsilon, \delta)$-approximation if $P(|f(W) - X| > \epsilon f(W)) \leq \delta$. It is often the case that f can be computed precisely if the entire window is available, but sublinear-space approximations, i.e., computation when the size of the available memory is $o(N + n)$, might be challenging. For example, Datar et al. [20] show linear space is required to maintain a $(1 \pm \epsilon, \delta)$-approximation of a sum of active elements if $p_i \in \{1, 0, -1\}$. A typical question in the sliding window model is the following: given function f, what are the upper and lower bounds on the space complexity of maintaining $(1 \pm \epsilon, \delta)$-approximation of f.

History

In their pioneering papers, Datar et al. [20, 21] and Babcock et al. [3] gave the first formal definition of the sliding window model. The model arose in the context of relational databases as a special case of time-sensitive queries in temporal databases [3]. Below we give a short survey of a subset of known results. A survey of Datar and Motwani [1] provides additional details. Datar et al. [20] gave the first algorithms for estimating the count and sum of positive integers, average, L_p for $p \in [1, 2]$, and a wide class of *weakly additive functions*. Gibbons and Tirthapura [24] provided further improvements to count and sum and gave the first methods for distributed computations. Lee and Ting [29] provided an optimal solution for a relaxed version of the counting problem, where the correct answer is provided only if it is comparable with the window's size. Braverman and Ostrovsky [6, 7] extended the results in [20] to a wider class of *smooth functions*. Chi et al. [15] considered a problem of frequent itemsets. Arasu and Manku [2], Lee and Ting [30], and Golab et al. [26] considered the problem of finding frequent elements, frequency counts, and quantiles. Babcock, Datar, Motwani, and O'Callaghan [5] pro-

vided first algorithms for variance and k-medians problems. Feigenbaum, Kannan and Zhang [22] presented an efficient solution for the diameter of a data set in multidimensional space. Later, Chan and Sadjad [23] presented optimal solutions for this and other geometric problems. Babcock, Datar and Motwani [4] presented algorithms for uniform random sampling from sliding windows.

Recently, Crouch et al. [17] presented the first approximation algorithms for important graph problems such as combinatorial sparsifiers and spanners, graph matching, and minimum spanning tree. Among other results, the methods in [17] allow non-smooth statistics using a modified smooth histogram to be computed. McGregor provided a detailed survey of these and other graph algorithms [32]. Datar and Muthukrishnan [19] solved problems of rarity and similarity. Braverman et al. [11] gave improved algorithms for rarity, similarity, and L_2-heavy hitters. Cormode and Yi developed several first algorithms for sliding windows in distributed streams [16]. Babcock et al. [4] gave the first method of sampling an element with constant expected space complexity. Braverman et al. [9, 10] gave a solution with a space complexity that is a constant in the worst case. Tatbul and Zdonik [35] considered the problem of load shedding for aggregation queries. Golab and Özsu [25] gave the first algorithm for approximating multi-joins. Recently, Braverman et al. [13] extended the zero-one law for increasing frequency-based functions [8] to sliding windows.

Key Results

Smooth Histogram

Extending the results in [20], Braverman and Ostrovsky [6, 7] introduced a notion of a *smooth function* and presented techniques for approximating smooth functions over sliding windows. Denote by $B \subseteq_r A$ the event when bucket B is a suffix of A; i.e., if $A = \{p_{n_1}, \ldots, p_{n_2}\}$ (for some $n_1 < n_2$), then $B = \{p_{n_3}, \ldots, p_{n_2}\}$, where $n_1 \leq n_3 \leq n_2$. Denote by $A \cup C$ the union of adjacent buckets A and C.

Definition 1 Function f is (α, β)-*smooth* if it preserves the following properties:

1. $f(A) \geq 0$.
2. $f(A) \geq f(B)$ for $B \subseteq_r A$.
3. $f(A) \leq poly(|A|)$.
4. For any $0 < \epsilon < 1$, there exist $\alpha = \alpha(\epsilon, f)$ and $\beta = \beta(\epsilon, f)$ such that
   - $0 < \beta \leq \alpha < 1$.
   - If $B \subseteq_r A$ and $(1-\beta) f(A) \leq f(B)$, then $(1-\alpha) f(A \cup C) \leq f(B \cup C)$ for any adjacent C.

In other words, a nonnegative, nondecreasing, and polynomially bounded function f is (α, β)-smooth if the following is true. If $f(B)$ is a $(1 \pm \beta)$-approximation of $f(A)$, then $f(B \cup C)$ is $(1 \pm \alpha)$-approximation of $f(A \cup C)$ for any $B \subseteq_r A$ and C. The main technical result of [7] is a new data structure called "smooth histogram" that allows algorithms for insertion-only streams to be extended to sliding windows with space complexity increased by a polylogarithmic factor. If there exists an algorithm that computes f precisely using g space and h time per element, then a smooth histogram can be used to maintain a $(1 \pm \alpha)$-approximation of f over sliding windows, using $O\left(\frac{1}{\beta} \log n (g + \log n)\right)$ bits and $O\left(\frac{1}{\beta} h \log n\right)$ time. Further, $(1 \pm \rho)$-approximation of f on D results in $(1 \pm (\alpha + \rho))$-approximation of f over sliding windows. Examples of smooth functions include sum, count, min, diameter, weakly additive functions, L_p norms, frequency moments, length of longest subsequence, and geometric mean.

Let f be (α, β)-smooth for which there exists an algorithm Λ that calculates f on D using g space and h operation per element. To maintain f on sliding windows, we construct a data structure that we call *smooth histogram*. It consists of a set of indexes $x_1 < x_2 < \cdots < x_s = N$ and instances of Λ for each bucket $B(x_i, N)$. Informally, the smooth histogram ensures the following properties of the sequence. The first two elements of the sequence always

"sandwich" the window, i.e., $x_1 \leq N - n < x_2$. This requirement and the monotonicity of f give us useful bounds for the sliding window W: $f(x_2, N) \leq f(W) \leq f(x_1, N)$. Also, f should slowly but constantly decrease with i, i.e., $f(x_{i+2}, N) < (1 - \beta) f(x_i, N)$. This gradual decrease, together with the fact that f is polynomially bounded, ensures that the sequence is short, i.e., $s = O\left(\frac{1}{\beta} \log n\right)$. Finally, the values of f on successive buckets were close in the past, i.e., $f(x_{i+1}, N') \geq (1 - \beta) f(x_i, N')$ for some $N' \leq N$. This represents our key idea and exploits the properties of smoothness. Indeed, $f(x_2, N') \geq (1-\beta) f(x_1, N')$ for some $N' \leq N$; thus, by the (α, β)-smoothness of f, we have $f(x_2, N) \geq (1 - \alpha) f(x_1, N) \geq (1 - \alpha) f(W)$. We refer a reader to [7] for further technical details.

Applications

There are several applications of the theoretical methods for the sliding window model, for example, [15, 18, 31, 33, 36].

Open Problems

We list several interesting open problems. It would be important to understand the difference between the sliding window model and other streaming models such as the insertion-only model, the turnstile, and decay models. This is perhaps one of the most important unresolved open problems; see, e.g., Sohler [34]. In particular, it would be nice to understand the exact space complexity of the frequency moments that are well understood in the other streaming models [12, 27, 28]. Also, it would be interesting to extend the coreset methods [14] to sliding windows, obtain polylogarithmic solutions for clustering, and improve the first clustering algorithm in [5]. Also, it would be nice to further develop graph methods [17]. Improving the approximation ratio of the maximum matching and obtaining the $O(n^{1+1/t})$ space bound for $(2t - 1)$-spanners are important open problems.

Acknowledgments This material is based upon work supported in part by the National Science Foundation under Grant No. 1447639.

Recommended Reading

1. Aggarwal C (2007) Data streams: models and algorithms. Advances in database systems. http://www.springer.com/west/home/default?SGWID=4-40356-22-107949228-0
2. Arasu A, Manku GS (2004) Approximate counts and quantiles over sliding windows. In: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS'04). ACM, New York, pp 286–296. doi:10.1145/1055558.1055598
3. Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS'02). ACM, New York, pp 1–16. doi:10.1145/543613.543615
4. Babcock B, Datar M, Motwani R (2002) Sampling from a moving window over streaming data. In: Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms (SODA'02). Society for Industrial and Applied Mathematics, Philadelphia, pp 633–634. http://dl.acm.org/citation.cfm?id=545381.545465
5. Babcock B, Datar M, Motwani R, O'Callaghan L (2003) Maintaining variance and k-medians over data stream windows. In: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS'03). ACM, New York, pp 234–243. doi:10.1145/773153.773176
6. Braverman V, Ostrovsky R (2007) Smooth histograms for sliding windows. In: Proceedings of the 48th annual IEEE symposium on foundations of computer science (FOCS'07). IEEE Computer Society, Washington, DC, pp 283–293. doi:10.1109/FOCS.2007.63
7. Braverman V, Ostrovsky R (2010) Effective computations on sliding windows. SIAM J Comput 39(6):2113–2131. doi:10.1137/090749281
8. Braverman V, Ostrovsky R (2010) Zero-one frequency laws. In: Proceedings of the 42nd ACM symposium on theory of computing (STOC'10). ACM, New York, pp 281–290. doi:10.1145/1806689.1806729
9. Braverman V, Ostrovsky R, Zaniolo C (2009) Optimal sampling from sliding windows. In: Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-

SIGART symposium on principles of database systems (PODS'09). ACM, New York, pp 147–156. doi:10.1145/1559795.1559818
10. Braverman V, Ostrovsky R, Zaniolo C (2012) Optimal sampling from sliding windows. J Comput Syst Sci 78(1):260–272. doi:10.1016/j.jcss.2011.04.004
11. Braverman V, Gelles R, Ostrovsky R (2013) How to catch l 2-heavy-hitters on sliding windows. In: Du DZ, Zhang G (eds) Computing and combinatorics. Lecture notes in computer science, vol 7936. Springer, Berlin/Heidelberg, pp 638–650. doi:10.1007/978-3-642-38768-5_56
12. Braverman V, Katzman J, Seidell C, Vorsanger G (2014) An optimal algorithm for large frequency moments using $O(n^{1-2/k})$ bits. In: Proceedings of the 18th international workshop on randomization and computation (RANDOM'2014)
13. Braverman V, Ostrovsky R, Roytman A (2014) Universal Streaming. ArXiv e-prints 1408.2604
14. Chen K (2009) On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. SIAM J Comput 39(3):923–947. doi:http://dx.doi.org/10.1137/070699007
15. Chi Y, Wang H, Yu PS, Muntz RR (2004) Moment: Maintaining closed frequent itemsets over a stream sliding window. In: ICDM, pp 59–66
16. Cormode G, Yi K (2012) Tracking distributed aggregates over time-based sliding windows. In: Proceedings of the 24th international conference on scientific and statistical database management (SSDBM'12). Springer, Berlin/Heidelberg, pp 416–430. doi:10.1007/978-3-642-31235-9_28
17. Crouch MS, McGregor A, Stubbs D (2013) Dynamic graphs in the sliding-window model. In: ESA, pp 337–348
18. Dang XH, Lee VC, Ng WK, Ong KL (2009) Incremental and adaptive clustering stream data over sliding window. In: Proceedings of the 20th international conference on database and expert systems applications (DEXA'09). Springer, Berlin/Heidelberg, pp 660–674. doi:10.1007/978-3-642-03573-9_55
19. Datar M, Muthukrishnan MS (2002) Estimating rarity and similarity over data stream windows. In: Proceedings of the 10th annual European symposium on algorithms (ESA'02). Springer, London, pp 323–334. http://dl.acm.org/citation.cfm?id=647912.740833
20. Datar M, Gionis A, Indyk P, Motwani R (2002) Maintaining stream statistics over sliding windows. SIAM J Comput 31(6):1794–1813
21. Datar M, Gionis A, Indyk P, Motwani R (2002) Maintaining stream statistics over sliding windows: (extended abstract). In: Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms (SODA'02). Society for Industrial and Applied Mathematics, Philadelphia, pp 635–644. http://dl.acm.org/citation.cfm?id=545381.545466
22. Feigenbaum J, Kannan S, Zhang J (2004) Computing diameter in the streaming and sliding-window models. Algorithmica 41(1):25–41
23. Feigenbaum J, Kannan S, Zhang J (2005) Computing diameter in the streaming and sliding-window models. Algorithmica 41:25–41
24. Gibbons PB, Tirthapura S (2002) Distributed streams algorithms for sliding windows. In: Proceedings of the fourteenth annual ACM symposium on parallel algorithms and architectures (SPAA'02). ACM, New York, pp 63–72. doi:10.1145/564870.564880
25. Golab L, Özsu MT (2003) Processing sliding window multi-joins in continuous queries over data streams. In: Proceedings of the 29th international conference on Very large data bases (VLDB'03), vol 29. VLDB Endowment, pp 500–511. http://dl.acm.org/citation.cfm?id=1315451.1315495
26. Golab L, DeHaan D, Demaine ED, Lopez-Ortiz A, Munro JI (2003) Identifying frequent items in sliding windows over on-line packet streams. In: Proceedings of the 3rd ACM SIGCOMM conference on internet measurement (IMC'03). ACM, New York, pp 173–178. doi:10.1145/948205.948227
27. Kane DM, Nelson J, Woodruff DP (2010) On the exact space complexity of sketching and streaming small norms. In: Proceedings of the 21st annual ACM-SIAM symposium on discrete algorithms (SODA'10)
28. Kane DM, Nelson J, Woodruff DP (2010) An optimal algorithm for the distinct elements problem. In: Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS'10). ACM, New York, pp 41–52. doi:10.1145/1807085.1807094
29. Lee LK, Ting HF (2006) Maintaining significant stream statistics over sliding windows. In: SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm. ACM, New York, pp 724–732. doi:http://doi.acm.org/10.1145/1109557.1109636
30. Lee LK, Ting HF (2006) A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS'06). ACM, New York, pp 290–297. doi:http://doi.acm.org/10.1145/1142351.1142393
31. Li J, Maier D, Tufte K, Papadimos V, Tucker PA (2005) No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. SIGMOD Rec 34(1):39–44. doi:10.1145/1058150.1058158
32. McGregor A (2014) Graph stream algorithms: A survey. SIGMOD Rec 43(1):9–20. doi:10.1145/2627692.2627694
33. Ren J, Ma R, Ren J (2009) Density-based data streams clustering over sliding windows. In: Proceedings of the 6th international conference on Fuzzy systems and knowledge discovery (FSKD'09), vol 5. IEEE Press, Piscataway, pp 248–252. http://dl.acm.org/citation.cfm?id=1801874.1801929
34. Sohler C (2006) List of open problems in sublinear algorithms: problem **20**. http://sublinear.info/20

35. Tatbul N, Zdonik S (2006) Window-aware load shedding for aggregation queries over data streams. In: Proceedings of the 32nd international conference on very large data bases (VLDB'06). VLDB Endowment, pp 799–810. http://dl.acm.org/citation.cfm?id=1182635.1164196
36. Zhang L, Li Z, Yu M, Wang Y, Jiang Y (2005) Random sampling algorithms for sliding windows over data streams. In: Proceedings of the 11th joint international computer conference, pp 572–575

Smooth Surface and Volume Meshing

Tamal Krishna Dey
Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA

Keywords

Delaunay mesh; Delaunay refinement; Surface mesh; Topology; Volume mesh

Years and Authors of Summarized Original Work

2001; Cheng, Dey, Edelsbrunner, Sullivan
2003; Boissonnat, Oudot
2004; Cheng, Dey, Ramos
2005; Oudot, Rienau, Yvinec
2012; Cheng, Dey, Shewchuk

Problem Definition

Given a smooth surface $S \subset \mathbb{R}^3$, we are required to compute a set of points $P \subset S$ and connect them with edges and triangles so that the resulted triangulation T is *geometrically* close and is *topologically* equivalent to S.

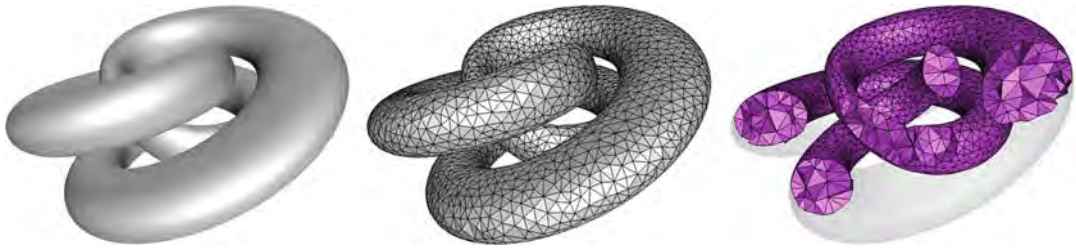
The output triangulation T is a simplicial 2-complex whose vertices are the points in P. Its underlying space, which is the pointwise union of the simplices (vertices, edges, triangles), is denoted with $|T|$. Geometric proximity is often characterized by Hausdorff distance between S and the underlying space $|T|$ of T. It is also desired that the triangle normals in T closely approximate the surface normals at its vertices. Topological equivalence is characterized by the existence of a *homeomorphism* between S and $|T|$. In some cases, the topological guarantee can be given in terms of *isotopy* which is stronger than homeomorphism. It is important to notice that, unlike polyhedral surfaces, a smooth surface cannot be represented *exactly* and hence needs to be approximated with a finite triangulation. This approximation requires that the mesh generation algorithms guarantee topological fidelity in addition to the geometric proximity.

In volume mesh generation, the space bounded by a smooth surface S is required to be tessellated with tetrahedra which form a simplicial 3-complex T. Similar to the surface case, it is required that the underlying space $|T|$ is geometrically close and topologically equivalent to the space bounded by S. It turns out that if the underlying space of the boundary 2-complex of T is geometrically close and has an isotopy to S, then so is $|T|$.

In both surface and volume meshes, it is desirable that the triangles and tetrahedra have good aspect ratio. This is often achieved by bounding the circumradius to shortest edge length ratios for triangles. Unfortunately, for tetrahedra, a bounded radius-edge ratio does not necessarily imply a bounded aspect ratio though most poor quality tetrahedra except slivers [4] are eliminated by bounded radius-edge ratio. Figure 1 shows an example of a surface and a volume mesh.

Key Results

Theoretically sound algorithms for surface meshing use the technique of Delaunay refinement originally proposed by Chew [8]. For a point set $P \subset \mathbb{R}^3$, let Vor P and Del P denote the Voronoi diagram and Delaunay triangulation of P, respectively. A typical Delaunay refinement algorithm iteratively samples the space to be meshed with a *locally furthest point* strategy that inserts points where a Voronoi face of appropriate dimension intersects the space. The decision of



Smooth Surface and Volume Meshing, Fig. 1 A knotted torus, its surface mesh, and its volume mesh

which points to be inserted is guided by certain desirable properties of the output such as topological equivalence, simplex radius-edge ratios, geometric proximity, and so on.

In both surface and volume meshing, the features of the surface S play an important role because regions of small features need to be sampled relatively densely to capture the geometry and topology of S. The definition of *local feature size* and ε-sample given by Amenta, Bern, and Eppstein [2] captures this idea.

Let S be a smooth, closed surface, that is, S is compact, C^2-smooth, and has no boundary. The *medial axis* $M(S)$ of S is defined as the closure of the set of points $x \in \mathbb{R}^3$ so that the distance $d(x, S)$ is realized by two or more points in S. The *local feature size* is defined as

$$f(x) = d(x, M).$$

A set of points $P \subset S$ is called an ε-sample of S if every point $x \in S$ has a sample point in P within $\varepsilon f(x)$ distance.

It turns out that if P is an ε-sample of S for a sufficiently small value of ε, a subcomplex of the Delaunay triangulation of this sample captures the topology of S. We define this subcomplex in generality and then specialize it to S.

Let V_ξ denote the dual Voronoi face of a Delaunay simplex ξ in Del P. The restricted Voronoi face of V_ξ with respect to $\mathbb{X} \subset \mathbb{R}^3$ is the intersection $V_\xi|_{\mathbb{X}} = V_\xi \cap \mathbb{X}$. The *restricted Voronoi diagram* and *restricted Delaunay triangulation* of P with respect to $\mathbb{X}$ are

$$\begin{aligned}\text{Vor } P|_{\mathbb{X}} &= \{V_\xi|_{\mathbb{X}} \mid V_\xi|_{\mathbb{X}} \neq \emptyset\} \text{ and Del } P|_{\mathbb{X}} \\ &= \{\xi \mid V_\xi|_{\mathbb{X}} \neq \emptyset\} \text{ respectively.}\end{aligned}$$

In words, Del $P|_{\mathbb{X}}$ consists of those Delaunay simplices in Del P whose dual Voronoi face intersects $\mathbb{X}$. We call these simplices *restricted*.

Now consider a sample P on the surface S. The restricted Delaunay triangulation of P with respect to S is Del $P|_S$. It is known that if P is an ε-sample of S for $\varepsilon \leq 0.09$, then Del $P|_S$ has its underlying space homeomorphic to S [1, 9]. To use this result one requires computing an ε-sample of S. A computation of local feature size or its approximation is necessary to determine if a sample is an ε-sample for a predetermined ε. Even if one is allowed to assume the availability of the local feature size at any given point, it is not immediately obvious how to place points on S so that they become ε-sample for a given $\varepsilon > 0$.

Surface Meshing

The following theorem about the fidelity of the restricted Delaunay triangulation of a dense sample on a smooth closed surface is the basis of provable surface meshing algorithms. It has been proved in various versions in [1, 5, 7, 9].

Theorem 1 *Let P be an ε-sample of a smooth, compact, boundary-less surface $S \subset \mathbb{R}^3$. The restricted Delaunay complex $T =$ Del $P|_S$ satisfies the following properties for $\varepsilon \leq 0.09$:*

1. *The underlying space $|T|$ is homeomorphic to S (actually, there is an ambient isotopy taking $|T|$ to S).*
2. *Every point in $|T|$ has a point $x \in S$ so that $d(p, x) \leq O(\varepsilon) f(x)$. Similarly, every point x in S has a point p in $|T|$ so that $d(p, x) \leq O(\varepsilon) f(x)$.*

3. *Each triangle $t \in T$ has a normal making an angle $O(\varepsilon)$ with the normal to the surface S at any of its vertices.*

Cheng, Dey, Edelsbrunner, and Sullivan [5] applied Chew's furthest point placement strategy [8] to maintain a dynamic surface mesh of a special type of surface called *skin surface* for which they computed the local feature size explicitly. The above theorem then allowed them to argue the geometric and topological fidelity of the output. Boissonnat and Oudot [3] used similar point placement strategy assuming that the local feature sizes are available, but they suggested how to initialize the meshing procedure for general surfaces. For a restricted triangle $t \in \text{Del}\, P|_S$, the dual Voronoi edge intersects S possibly at multiple points. Each ball centering such an intersection point and circumscribing vertices of t is called a *surface Delaunay ball* of t. Boissonnat and Oudot observed that if every surface Delaunay ball of each restricted triangle has small radius, say at most 0.05 times the local feature size at the center, then P a 0.09-sample of S. It follows that $\text{Del}\, P|_S$ at this point satisfies the properties stated in Theorem 1. The deduction of this conclusion also requires that every component of S has at least one Voronoi edge intersecting it which Boissonnat and Oudot ensure with *persistent triangles*.

When local feature sizes are not known, we cannot use the method of Boissonnat and Oudot [3]. Instead, we fall back upon a different strategy to drive the Delaunay refinement. A result of Edelsbrunner and Shah [10] says that if Voronoi faces intersect S in a closed topological ball of appropriate dimension, then the underlying space of the restricted Delaunay triangulation becomes homeomorphic to S. In fact, this is the basis of the proof of Theorem 1. Therefore, a Delaunay refinement driven by the violation of the topological ball conditions provides a viable strategy for meshing with topological guarantees. This strategy is followed by Cheng, Dey, Ramos, and Ray [6].

The algorithm of Cheng et al. avoids computing local feature sizes or their approximation; however, it needs to compute critical points of certain functions on the surface, which may not be easily computable. In a recent book on Delaunay mesh generation [7], Cheng, Dey, and Shewchuk have suggested a strategy that is more practical which leverages on both algorithms of Boissonnat and Oudot [3] and Cheng et al. [6]. It operates with an input parameter $\lambda > 0$. As long as the surface Delaunay balls of the restricted triangles are not all smaller than a ball of radius λ, the algorithm refines. It also refines if the restricted triangles around each vertex do not form a topological disk. The algorithm can be shown to terminate and has the following guarantees.

Theorem 2 ([7]) *There is a Delaunay refinement algorithm that runs with a parameter $\lambda > 0$ on an input smooth, compact, boundary-less surface S with the following guarantees:*

1. *The output mesh is a Delaunay subcomplex and is a 2-manifold for all values of λ.*
2. *If λ is sufficiently small, then the output mesh has similar guarantees with respect to the input surface S as in Theorem 1 (replace ε with λ).*

It should be noted that in any of the above algorithms, one may introduce the condition that the output triangles have radius-edge ratio of at most 1 without loosing any of the geometric or topological guarantees. Even a graded mesh can be guaranteed by supplying an appropriate grading function as input. For details see [7].

Volume Meshing

Let $\mathcal{O}$ denote the volume enclosed by a smooth surface S. Consider the surface mesh of S produced by one of the algorithms mentioned above. The volume enclosed by this surface mesh is already triangulated with Delaunay tetrahedra. We can further refine them for quality using the radius-edge ratio condition. The circumcenters of skinny tetrahedra can be added as long as they do not disturb the surface triangulation. One easy approach is to skip adding those circumcenters who encroach the surface Delaunay balls meaning that they lie inside these balls. This ensures that all

surface triangles remain intact. The trade-off of this easy fix is that the tetrahedra near the boundary may not have bounded radius-edge ratios. To ensure the quality for all tetrahedra, additional effort is required to maintain the surface. Oudot, Rineau, and Yvinec [11] proposed an algorithm for guaranteed quality volume meshing.

The algorithm first runs the algorithm of [3] to obtain a surface triangulation with a vertex set P on the surface. It uses two parameters ε and ρ where ε controls the level of refinement and ρ controls the aspect ratios of the tetrahedra and triangles. It ensures that all restricted triangles on the surface have vertices from S. It refines surface triangles as in surface meshing algorithm. Then, it refines the tetrahedra. Refinement of surface triangles is given priority over the tetrahedra. Oudot et al. [11] prove that their algorithm terminates and has the following geometric and topological guarantees.

Theorem 3 ([11]) *Given a volume $\mathcal{O}$ bounded by a smooth surface S, for $\varepsilon \leq 0.05$ and $\rho > 1$, there is an algorithm that produces $T = \text{Del}\, P|_{\mathcal{O}}$ where each tetrahedron in T has radius-edge ratio at most ρ and $|T|$ is homeomorphic (isotopic) to $\mathcal{O}$ and the boundary of T is $\text{Del}\, P|_S$. Furthermore, the isotopy moves a point $x \in S$ by at most $O(\varepsilon^2) f(x)$ distance.*

An improved version of the algorithm and its analysis in presented in the book [7].

URLs to Code and Data Sets

CGAL(http://cgal.org), a library of geometric algorithms, contains software for surface and volume mesh generation. The DelPSC software that implements the surface and volume meshing algorithms as described in [7] is also available from http://web.cse.ohio-state.edu/~tamaldey/delpsc.html.

Cross-References

- Manifold Reconstruction
- Meshing Piecewise Smooth Complexes
- Surface Reconstruction

Recommended Reading

1. Amenta N, Bern M (1999) Surface reconstruction by Voronoi filtering. Discret Comput Geom 22:481–504
2. Amenta N, Bern M, Eppstein D (1998) The crust and the beta-skeleton: combinatorial curve reconstruction. Graph Models Image Process 60(2:2):125–135
3. Boissonnat J-D, Oudot S (2005) Provably good surface sampling and meshing of surfaces. Graph Models 67:405–451. Conference version 2003
4. Cheng S-W, Dey TK, Edelsbrunner H, Teng SH (2000) Sliver exudation. J ACM 47:883–904
5. Cheng H-L, Dey TK, Edelsbrunner H, Sullivan J (2001) Dynamic skin triangulation. Discret Comput Geom 25:525–568
6. Cheng S-W, Dey TK, Ramos EA, Ray T (2007) Sampling and meshing a surface with guaranteed topology and geometry. SIAM J Comput 37:1199–1227. Conference version 2004
7. Cheng S-W, Dey TK, Shewchuk JR (2012) Delaunay mesh generation. CRC Press, Boca Raton
8. Chew LP (1993) Guaranteed-quality mesh generation for curved surfaces. In: Proceedings of the 9th annual symposium on computational geometry, San Diego, pp 274–280
9. Dey TK (2006) Curve and surface reconstruction: algorithms with mathematical analysis. Cambridge University Press, New York
10. Edelsbrunner H, Shah N (1997) Triangulating topological spaces. Int J Comput Geom Appl 7:365–378
11. Oudot S, Rineau L, Yvinec M (2005) Meshing volumes bounded by smooth surfaces. In: Proceedings of the 14th international meshing roundtable, pp 203–219

Smoothed Analysis

Heiko Röglin
Department of Computer Science, University of Bonn, Bonn, Germany

Keywords

Computational complexity; Linear programming; Probabilistic analysis

Years and Authors of Summarized Original Work

2001; Spielman, Teng
2004; Beier, Vöcking

Problem Definition

Smoothed analysis has originally been introduced by Spielman and Teng [22] in 2001 to explain why the simplex method is usually fast in practice despite its exponential worst-case running time. Since then it has been applied to a wide range of algorithms and optimization problem. In smoothed analysis, inputs are generated in two steps: first, an adversary chooses an arbitrary instance, and then this instance is slightly perturbed at random. The smoothed performance of an algorithm is defined to be the worst expected performance the adversary can achieve. This model can be viewed as a less pessimistic worst-case analysis, in which the randomness rules out pathological worst-case instances that are rarely observed in practice but dominate the worst-case analysis. If the smoothed running time of an algorithm is low (i.e., the algorithm is efficient in expectation on any perturbed instance) and inputs are subject to a small amount of random noise, then it is unlikely to encounter an instance on which the algorithm performs poorly. In practice, random noise can stem, for example, from measurement errors, numerical imprecision, or rounding errors. It can also model arbitrary influences, which we cannot quantify exactly, but for which there is also no reason to believe that they are adversarial. After its invention smoothed analysis has been applied in a variety of different contexts, e.g., linear programming [8, 19, 21, 23], multi-objective optimization [5, 10, 17, 18], online and approximation algorithms [4, 7, 20], searching and sorting [3, 12, 15, 16], game theory [9, 11], and local search [1, 2, 13, 14].

Key Results

Simplex Method

Spielman and Teng [22] considered linear programs of the form

$$\begin{aligned} &\text{maximize} \quad c^T x \\ &\text{subject to} \quad (\overline{A} + G)x \leq (\overline{b} + h), \end{aligned}$$

where $\overline{A} \in \mathbb{R}^{n \times d}$ and $\overline{b} \in \mathbb{R}^n$ are chosen arbitrarily by an adversary and the entries of the matrix $G \in \mathbb{R}^{n \times d}$ and the vector $h \in \mathbb{R}^n$ are independent Gaussian random variables that represent the perturbation. These Gaussian random variables have mean 0 and standard deviation $\sigma \cdot (\max_i \|(\overline{b}_i, \overline{a}_i)\|)$, where the vector $(\overline{b}_i, \overline{a}_i) \in \mathbb{R}^{d+1}$ consists of the i-th component of $\overline{b}$ and the i-th row of $\overline{A}$ and $\|\cdot\|$ denotes the Euclidean norm. Without loss of generality, we can scale the linear program specified by the adversary and assume that $\max_i \|(\overline{b}_i, \overline{a}_i)\| = 1$. Then the perturbation consists of adding an independent Gaussian random variable with standard deviation σ to each entry of $\overline{A}$ and $\overline{b}$. The smaller σ is chosen, the more concentrated are the random variables, and hence, the better worst-case instances can be approximated by the adversary. Intuitively, σ can be seen as a measure specifying how close the analysis is to a worst-case analysis.

Spielman and Teng analyzed the smoothed running time of the simplex algorithm using the *shadow vertex pivot rule*. This pivot rule has a simple and intuitive geometric description which makes probabilistic analyses feasible. Let x_0 denote the given initial vertex of the polytope $\mathcal{P}$ of feasible solutions. Since x_0 is a vertex of the polytope, there exists an objective function $u^T x$ which is maximized by x_0 subject to the constraint $x \in \mathcal{P}$. In the first step, the shadow vertex pivot rule computes an objective function $u^T x$ with this property. If x_0 is not an optimal solution of the linear program, then the vectors c and u are linearly independent and span a plane. The shadow vertex method projects the polytope $\mathcal{P}$ onto this plane. The *shadow*, that is, the projection of $\mathcal{P}$ onto this plane is a possibly open polygon. One can show that both x_0 and the optimal solution x^* are projected onto vertices of the polygon and that each path between the projections of x_0 and x^* in the polygon corresponds to a path between x_0 and x^* in the polytope. Hence, one only needs to follow the edges of the polygon starting from the projection of x_0 to (the projection of) x^*.

The number of steps performed by the simplex method with shadow vertex pivot rule is upper bounded by the number of vertices of the two-dimensional projection of the polytope. Hence, bounding the expected number of vertices on the

polygon is the crucial step for bounding the expected running time of the simplex method with shadow vertex pivot rule. Spielman and Teng first consider the case that the polytope $\mathcal{P}$ is projected onto a fixed plane specified by two fixed vectors c and u. They show that the expected number of vertices of the polygon is polynomially bounded in d, n, and $1/\sigma$. Though this result is the main ingredient of the analysis, alone it does not yield a polynomial bound on the smoothed running time of the simplex method. We have, for example, not yet described how the initial solution x_0 is found. It is also problematic that the vector u is not independent of the constraints because it is determined by x_0 which in turn is determined by a subset of the constraints. Spielman and Teng showed in a very involved analysis the following theorem.

Theorem 1 *The smoothed running time of the shadow vertex simplex method is bounded polynomially in d, n, and $1/\sigma$.*

Later, this analysis was substantially improved and simplified by Vershynin [23], who proved that the smoothed running time is even polynomially bounded in d, $\log n$, and $1/\sigma$.

Binary Optimization Problems

Beier and Vöcking [6] studied the question which *linear binary optimization problems* have *polynomial smoothed complexity*. Intuitively these are the problems that can be solved efficiently on perturbed inputs. An instance I of such an optimization problem Π consists of a set of feasible solutions $\mathcal{S} \subseteq \{0,1\}^n$ and a linear objective function $f : \{0,1\}^n \to \mathbb{R}$ of the form maximize (or minimize) $f(x) = c^T x$ for some $c \in \mathbb{R}^n$. Many well-known optimization problems can be formulated this way, e.g., the problem of finding a *Minimum Spanning Tree*, the *Knapsack Problem*, and the *Traveling Salesman Problem*.

It is assumed that an adversary is allowed to choose the coefficients of the objective function from the interval $[-1, 1]$. In the second step, these coefficients are perturbed by adding independent Gaussian random variables with mean 0 and standard deviation σ to them. Naturally one might define that a problem Π has *polynomial smoothed complexity* if there exists an algorithm A for Π whose expected running time $\mathbf{E}[T_A(I)]$ is bounded polynomially in the input size $|I|$ and $1/\sigma$. This definition, however, is not sufficiently robust as it depends on the machine model. An algorithm with expected polynomial running time on one machine model might have expected exponential running time on another machine model even if the former can be simulated by the latter in polynomial time. In contrast, the definition from [6] yields a notion of polynomial smoothed complexity that does not vary among classes of machines admitting polynomial time simulations among each other. It states that a problem Π has polynomial smoothed complexity if there exists an algorithm A for Π and some $\alpha > 0$ such that $\mathbf{E}[T_A(I)^\alpha]$ is bounded polynomially in the input size $|I|$ and $1/\sigma$.

Beier and Vöcking proved the following theorem that characterizes the class of linear binary optimization problems with polynomial smoothed complexity.

Theorem 2 *A linear binary optimization problem Π has polynomial smoothed complexity if and only if there exists a randomized algorithm for solving Π whose expected worst-case running time is pseudo-polynomial with respect to the coefficients in the objective function.*

For example, the knapsack problem, which can be solved by dynamic programming in pseudo-polynomial time, has polynomial smoothed complexity even if the weights are fixed and only the profits are randomly perturbed. Moreover, the traveling salesman problem does not have polynomial smoothed complexity when only the distances are randomly perturbed, unless P = NP, since a simple reduction from Hamiltonian cycle shows that it is strongly NP-hard.

Open Problems

An interesting open question is whether or not other pivot rules for the simplex method also have polynomial smoothed running time. It would also be interesting to see whether the insights gained from smoothed analysis can be used to improve existing algorithms.

Cross-References

▸ Knapsack

Recommended Reading

1. Arthur D, Vassilvitskii S (2009) Worst-case and smoothed analysis of the ICP algorithm, with an application to the k-means method. SIAM J Comput 39(2):766–782
2. Arthur D, Manthey B, Röglin H (2011) Smoothed analysis of the k-means method. J ACM 58(5)
3. Banderier C, Beier R, Mehlhorn K (2003) Smoothed analysis of three combinatorial problems. In: Proceedings of the 28th international symposium on mathematical foundations of computer science (MFCS), Bratislava. Lecture notes in computer science, vol 2747. Springer, pp 198–207
4. Becchetti L, Leonardi S, Marchetti-Spaccamela A, Schäfer G, Vredeveld T (2006) Average case and smoothed competitive analysis of the multilevel feedback algorithm. Math Oper Res 31(1):85–108
5. Beier R, Vöcking B (2004) Random knapsack in expected polynomial time. J Comput Syst Sci 69(3):306–329
6. Beier R, Vöcking B (2006) Typical properties of winners and losers in discrete optimization. SIAM J Comput 35(4):855–881
7. Bläser M, Manthey B, Rao BVR (2011) Smoothed analysis of partitioning algorithms for Euclidean functionals. In: Proceedings of the 12th workshop on algorithms and data structures (WADS), New York. Lecture notes in computer science. Springer, pp 110–121
8. Blum AL, Dunagan JD (2002) Smoothed analysis of the perceptron algorithm for linear programming. In: Proceedings of the 13th annual ACM-SIAM symposium on discrete algorithms (SODA), San Francisco. SIAM, pp 905–914
9. Boros E, Elbassioni K, Fouz M, Gurvich V, Makino K, Manthey B (2011) Stochastic mean payoff games: smoothed analysis and approximation schemes. In: Proceedings of the 38th international colloquium on automata, languages and programming (ICALP), Zurich, Part I. Lecture notes in computer science, vol 6755. Springer, pp 147–158
10. Brunsch T, Röglin H (2012) Improved smoothed analysis of multiobjective optimization. In: Proceedings of the 44th annual ACM symposium on theory of computing (STOC), New York, pp 407–426
11. Chen X, Deng X, Teng SH (2009) Settling the complexity of computing two-player Nash equilibria. J ACM 56(3)
12. Damerow V, Manthey B, auf der Heide FM, Räcke H, Scheideler C, Sohler C, Tantau T (2012) Smoothed analysis of left-to-right maxima with applications. ACM Trans Algorithms 8(3): Article no 30
13. Englert M, Röglin H, Vöcking B (2007) Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. In: Proceedings of the 18th annual ACM-SIAM symposium on discrete algorithms (SODA), New Orleans. SIAM, pp 1295–1304
14. Etscheid M, Röglin H (2014) Smoothed analysis of local search for the maximum-cut problem. In: Proceedings of the 25th annual ACM-SIAM symposium on discrete algorithms (SODA), Portland pp 882–889
15. Fouz M, Kufleitner M, Manthey B, Zeini Jahromi N (2012) On smoothed analysis of quicksort and Hoare's find. Algorithmica 62(3–4):879–905
16. Manthey B, Reischuk R (2007) Smoothed analysis of binary search trees. Theor Comput Sci 378(3):292–315
17. Moitra A, O'Donnell R (2012) Pareto optimal solutions for smoothed analysts. SIAM J Comput 41(5):1266–1284
18. Röglin H, Teng SH (2009) Smoothed analysis of multiobjective optimization. In: Proceedings of the 50th annual IEEE symposium on foundations of computer science (FOCS), Atlanta. IEEE, pp 681–690
19. Sankar A, Spielman DA, Teng SH (2006) Smoothed analysis of the condition numbers and growth factors of matrices. SIAM J Matrix Anal Appl 28(2):446–476
20. Schäfer G, Sivadasan N (2005) Topology matters: smoothed competitiveness of metrical task systems. Theor Comput Sci 241(1–3):216–246
21. Spielman DA, Teng SH (2003) Smoothed analysis of termination of linear programming algorithms. Math Program 97(1–2):375–404
22. Spielman DA, Teng SH (2004) Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. J ACM 51(3):385–463
23. Vershynin R (2009) Beyond Hirsch conjecture: walks on random polytopes and smoothed complexity of the simplex method. SIAM J Comput 39(2):646–678

Snapshots in Shared Memory

Eric Ruppert
Department of Computer Science and Engineering, York University, Toronto, ON, Canada

Keywords

Atomic scan

Years and Authors of Summarized Original Work

1993; Afek, Attiya, Dolev, Gafni, Merritt, Shavit

Problem Definition

Implementing a snapshot object is an abstraction of the problem of obtaining a consistent view of several shared variables while other processes are concurrently updating those variables.

In an asynchronous shared-memory distributed system, a collection of n processes communicate by accessing shared data structures, called *objects*. The system provides basic types of shared objects; other needed types must be built from them. One approach uses locks to guarantee exclusive access to the basic objects, but this approach is not fault-tolerant, risks deadlock or livelock, and causes delays when a process holding a lock runs slowly. Lock-free algorithms avoid these problems but introduce new challenges. For example, if a process reads two shared objects, the values it reads may not be consistent if the objects were updated between the two reads.

A *snapshot object* stores a vector of m values, each from some domain D. It provides two operations: scan and update(i, v), where $1 \leq i \leq m$ and $v \in D$. If the operations are invoked sequentially, an update(i, v) operation changes the value of the ith component of the stored vector to v, and a scan operation returns the stored vector.

Correctness when snapshot operations by different processes overlap in time is described by the *linearizability* condition, which says operations should appear to occur instantaneously. More formally, for every execution, one can choose an instant of time for each operation (called its *linearization point*) between the invocation and the completion of the operation. (An incomplete operation may either be assigned no linearization point or given a linearization point at any time after its invocation.) The responses returned by all completed operations in the execution must return the same result as they would if all operations were executed sequentially in the order of their linearization points.

An implementation must also satisfy a progress property. *Wait-freedom* requires that each process completes each scan or update in a finite number of its own steps. The weaker *non-blocking* progress condition says the system cannot run forever without some operation completing.

This article describes implementations of snapshots from more basic types, which are also linearizable, without locks. Two types of snapshots have been studied. In a *single-writer* snapshot, each component is owned by a process, and only that process may update it. (Thus, for single-writer snapshots, $m = n$.) In a *multi-writer* snapshot, any process may update any component. There also exist algorithms for *single-scanner* snapshots, where only one process may scan at a time [10, 13, 14, 16]. Snapshots were introduced by Afek et al. [1], Anderson [2] and Aspnes and Herlihy [4].

Space complexity is measured by the number of basic objects used and their size (in bits). Time complexity is measured by the maximum number of steps a process must do to finish a scan or update, where a step is an access to a basic shared object. (Local computation and local memory accesses are usually not counted.) Complexity bounds will be stated in terms of $n, m, d = \log |D|$ and k, the number of operations invoked in an execution. Ordinarily, there is no bound on k.

Most of the algorithms below use read-write registers, the most elementary shared object type. A *single-writer* register may only be written by one process. A *multi-writer* register may be written by any process. Some algorithms using stronger types of basic objects are discussed in section "Wait-Free Implementations from Small, Stronger Objects".

Key Results

A Simple Non-blocking Implementation from Small Registers

Suppose each component of a single-writer snapshot object is represented by a single-writer register. Process i does an update(i, v) by writing v and a sequence number into register i, and incrementing its sequence number. Performing a scan operation is more difficult than merely reading each of the m registers, since some registers

might change while these reads are done. To scan, a process repeatedly reads all the registers. A sequence of reads of all the registers is called a *collect*. If two collects return the same vector, the scan returns that vector (with the sequence numbers stripped away). The sequence numbers ensure that, if the same value is read in a register twice, the register had that value during the entire interval between the two reads. The scan can be assigned a linearization point between the two identical collects, and updates are linearized at the write. This algorithm is non-blocking, since a scan continues running only if at least one update operation is completed during each collect. A similar algorithm, with process identifiers appended to the sequence numbers, implements a non-blocking multi-writer snapshot from m multi-writer registers.

Wait-Free Implementations from Large Registers

Afek et al. [1] described how to modify the non-blocking single-writer snapshot algorithm to make it wait-free using scans embedded within the updates. An update(i, v) first does a scan and then writes a triple containing the scan's result, v and a sequence number into register i. While a process P is repeatedly performing collects to do a scan, either two collects return the same vector (which P can return) or P will eventually have seen three different triples in the register of some other process. In the latter case, the third triple that P saw must contain a vector that is the result of a scan that started after P's scan, so P's scan outputs that vector. Updates and scans that terminate after seeing two identical collects are assigned linearization points as before. If one scan obtains its output from an embedded scan, the two scans are given the same linearization point. This is a wait-free single-writer snapshot implementation from n single-writer registers of $(n+1)d + \log k$ bits each. Operations complete within $O(n^2)$ steps. Afek et al. [1] also describe how to replace the unbounded sequence numbers with handshaking bits. This requires $n\Theta(nd)$-bit registers and n^2 1-bit registers. Operations still complete in $O(n^2)$ steps.

The same idea can be used to build multi-writer snapshots from multi-writer registers. Using unbounded sequence numbers yields a wait-free algorithm that uses m registers storing $\Theta(nd + \log k)$ bits each, in which each operation completes within $O(mn)$ steps. (This algorithm is given explicitly in [9].) No algorithm can use fewer than m registers if $n \geq m$ [9]. If handshaking bits are used instead, the multi-writer snapshot algorithm uses n^2 1-bit registers, $m(d + \log n)$-bit registers and n (md)-bit registers, and each operation uses $O(nm + n^2)$ steps [1].

Guerraoui and Ruppert [12] gave a similar wait-free multi-writer snapshot implementation that is anonymous, i.e., it does not use process identifiers and all processes are programmed identically.

Anderson [3] gave an implementation of a multi-writer snapshot from a single-writer snapshot. Each process stores its latest update to each component of the multi-writer snapshot in the single-writer snapshot, with associated timestamp information computed by scanning the single-writer snapshot. A scan is done using just one scan of the single-writer snapshot. An update requires scanning and updating the single-writer snapshot twice. The implementation involves some blow-up in the size of the components, i.e., to implement a multi-writer snapshot with domain D requires a single-writer snapshot with a much larger domain D'. If the goal is to implement multi-writer snapshots from single-writer registers (rather than multi-writer registers), Anderson's construction gives a more efficient solution than that of Afek et al.

Attiya, Herlihy and Rachman [7] defined the *lattice agreement* object, which is very closely linked to the problem of implementing a single-writer snapshot when there is a known upper bound on k. Then, they showed how to construct a single-writer snapshot (with no bound on k) from an infinite sequence of lattice agreement objects. Each snapshot operation accesses the lattice agreement object twice and does $O(n)$ additional steps. Their implementations of lattice agreement are discussed in section "Wait-Free Implementations from Small, Stronger Objects".

Attiya and Rachman [8] used a similar approach to give a single-writer snapshot implementation from large single-writer registers using $O(n \log n)$ steps per operation. Each update has an associated sequence number. A scanner traverses a binary tree of height $\log k$ from root to leaf (here, a bound on k is required). Each node has an array of n single-writer registers. A process arriving at a node writes its current vector into a single-writer register associated with the node and then gets a new vector by combining information read from all n registers. It proceeds to the left or right child depending on the sum of the sequence numbers in this vector. Thus, all scanners can be linearized in the order of the leaves they reach. Updates are performed by doing a similar traversal of the tree. The bound on k can be removed as in [7]. Attiya and Rachman also give a more direct implementation that achieves this by recycling the snapshot object that assumes a bound on k. Their algorithm has also been adapted to solve condition-based consensus [15].

Attiya, Fouren and Gafni [6] described how to adapt the algorithm of Attiya and Rachman [8] so that the number of steps required to perform an operation depends on the number of processes that actually access the object, rather than the number of processes in the system.

Attiya and Fouren [5] solve lattice agreement in $O(n)$ steps. (Here, instead of using the terminology of lattice agreement, the algorithm is described in terms of implementing a snapshot in which each process does at most one snapshot operation.) The algorithm uses, as a data structure, a two-dimensional array of $O(n^2)$ *reflectors*. A reflector is an object that can be used by two processes to exchange information. Each reflector is built from two large single-writer registers. Each process chooses a path through the array of reflectors, so that at most two processes visit each reflector. Each reflector in column i is used by process i to exchange information with one process $j < i$. If process i reaches the reflector first, process j learns about i's update (if any). If process j reaches it first, then process i learns all the information that j has already gathered. (If both reach it at about the same time, both processes learn the information described above.) As the processes move from column $i-1$ to column i, a process that enters column i at some row r will have gathered all the information that has been gathered by any process that enters column i below row r (and possibly more). This invariant is maintained by ensuring that if process i passes information to any process $j < i$ in row r of column i, it also passes that information to all processes that entered column i above row r. Furthermore, process i exits column i at a row that matches the amount of information it learns while traveling through the column. When processes have reached the rightmost column of the array, the ones in higher rows know strictly more than the ones in lower rows. Thus, the linearization order of their scans is the order in which they exit the rightmost column, from bottom to top. The techniques of Attiya, Herlihy and Rachman [7, 8], mentioned above, can be used to remove the restriction that each process performs at most one operation. The number of steps per operation is still $O(n)$.

Wait-Free Implementations from Small, Stronger Objects

All of the wait-free implementations described above use registers that can store $\Omega(m)$ bits each, and are therefore not practical when m is large. Some implementations from smaller objects equipped with stronger synchronization operations, rather than just reads and writes, are described in this section. An object is considered to be small if it can store $O(d + \log n + \log k)$ bits. This means that it can store a constant number of component values, process identifiers and sequence numbers.

Attiya, Herlihy and Rachman [7] gave an elegant divide-and-conquer recursive solution to the lattice agreement problem. The division of processes into groups for the recursion can be done dynamically using test&set objects. This provides a snapshot algorithm that runs in $O(n)$ time per operation, and uses $O(kn^2 \log n)$ small single-writer registers and $O(kn \log^2 n)$ test&set objects. (This requires modifying their implementation to replace those registers that are large, which are written only once, by many small

registers.) Using randomization, each test&set object can be replaced by single-writer registers to give a snapshot implementation from registers only with $O(n)$ expected steps per operation.

Jayanti [13] gave a multi-writer snapshot implementation from $O(mn^2)$ small compare & swap objects where updates take $O(1)$ steps and scans take $O(m)$ steps. He began with a very simple single-scanner, single-writer snapshot implementation from registers that uses a secondary array to store a copy of recent updates. A scan clears that array, collects the main array, and then collects the secondary array to find any overlooked updates. Several additional mechanisms are introduced for the general, multi-writer, multi-scanner snapshot. In particular, compare & swap operations are used instead of writes to coordinate writers updating the same component and multiple scanners coordinate with one another to simulate a single scanner. Jayanti's algorithm builds on an earlier paper by Riany, Shavit and Touitou [16], which gave an implementation that achieved similar complexity, but only for a single-writer snapshot.

Applications

Applications of snapshots include distributed databases, storing checkpoints or backups for error recovery, garbage collection, deadlock detection, debugging distributed programmes and obtaining a consistent view of the values reported by several sensors. Snapshots have been used as building blocks for distributed solutions to randomized consensus and approximate agreement. They are also helpful as a primitive for building other data structures. For example, consider implementing a counter that stores an integer and provides increment, decrement and read operations. Each process can store the number of increments it has performed minus the number of its decrements in its own component of a single-writer snapshot object, and the counter may be read by summing the values from a scan. See [10] for references on many of the applications mentioned here.

Open Problems

Some complexity lower bounds are known for implementations from registers [9], but there remain gaps between the best known algorithms and the best lower bounds. In particular, it is not known whether there is an efficient wait-free implementation of snapshots from small registers.

Experimental Results

Riany, Shavit and Touitou gave performance evaluation results for several implementations [16].

Cross-References

- ▸ Implementing Shared Registers in Asynchronous Message-Passing Systems
- ▸ Linearizability
- ▸ Registers

Recommended Reading

See also Fich's survey paper on the complexity of implementing snapshots [11].

1. Afek Y, Attiya H, Dolev D, Gafni E, Merritt M, Shavit N (1993) Atomic snapshots of shared memory. J Assoc Comput Mach 40:873–890
2. Anderson JH (1993) Composite registers. Distrib Comput 6:141–154
3. Anderson JH (1994) Multi-writer composite registers. Distrib Comput 7:175–195
4. Aspnes J, Herlihy M (1990) Wait-free data structures in the asynchronous PRAM model. In: Proceedings of the 2nd ACM symposium on parallel algorithms and architectures, Crete, July 1990. ACM, New York, pp 340–349
5. Attiya H, Fouren A (2001) Adaptive and efficient algorithms for lattice agreement and renaming. SIAM J Comput 31:642–664
6. Attiya H, Fouren A, Gafni E (2002) An adaptive collect algorithm with applications. Distrib Comput 15:87–96
7. Attiya H, Herlihy M, Rachman O (1995) Atomic snapshots using lattice agreement. Distrib Comput 8:121–132
8. Attiya H, Rachman O (1998) Atomic snapshots in O(n log n) operations. SIAM J Comput 27:319–340

9. Ellen F, Fatourou P, Ruppert E (2007) Time lower bounds for implementations of multi-writer snapshots. J Assoc Comput Mach 54(6), 30
10. Fatourou P, Kallimanis ND (2006) Single-scanner multi-writer snapshot implementations are fast! In: Proceedings of the 25th ACM symposium on principles of distributed computing, Colorado, July 2006. ACM, New York, pp 228–237
11. Fich FE (2005) How hard is it to take a snapshot? In: SOFSEM 2005: theory and practice of computer science, Liptovský Ján, Jan 2005. LNCS, vol 3381. Springer, pp 28–37
12. Guerraoui R, Ruppert E (2007) Anonymous and fault-tolerant shared-memory computing. Distrib Comput 20(3):165–177
13. Jayanti P (2005) An optimal multi-writer snapshot algorithm. In: Proceedings of the 37th ACM symposium on theory of computing, Baltimore, May 2005. ACM, New York, pp 723–732
14. Kirousis LM, Spirakis P, Tsigas P (1996) Simple atomic snapshots: a linear complexity solution with unbounded time-stamps. Inf Process Lett 58:47–53
15. Mostéfaoui A, Rajsbaum S, Raynal M, Roy M (2004) Conditionbased consensus solvability: a hierarchy of conditions and efficient protocols. Distrib Comput 17:1–20
16. Riany Y, Shavit N, Touitou D (2001) Towards a practical snapshot algorithm. Theor Comput Sci 269:163–201

Sorting by Transpositions and Reversals (Approximate Ratio 1.5)

Chin Lung Lu
Institute of Bioinformatics and Department of Biological Science and Technology, National Chiao Tung University, Hsinchu, Taiwan

Keywords

Genome rearrangements

Problem Definition

One of the most promising ways to determine evolutionary distance between two organisms is to compare the order of appearance of identical (e.g., orthologous) genes in their genomes. The resulting genome rearrangement problem calls for finding a shortest sequence of rearrangement operations that sorts one genome into the other. In this work [8], Hartman and Sharan provide a 1.5-approximation algorithm for the problem of sorting by transpositions, transreversals, and revrevs, improving on a previous 1.75 ratio for this problem. Their algorithm is also faster than current approaches and requires $O(n^{3/2}\sqrt{\log n})$ time for n genes.

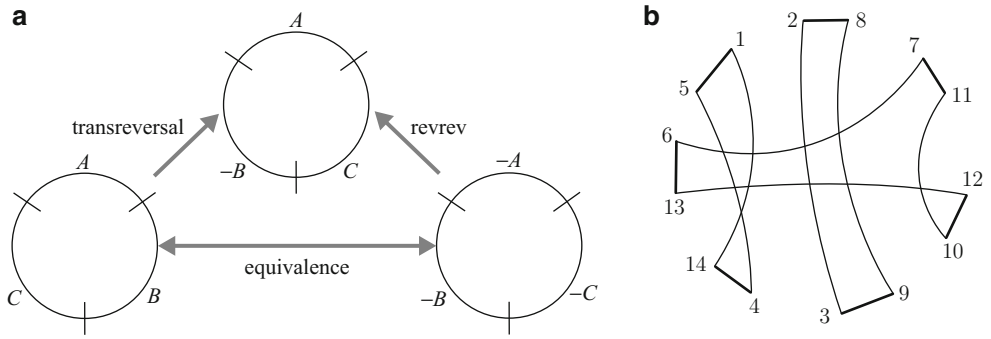
Notations and Definition

A *signed permutation* $\pi = [\pi_1, \pi_2, \ldots, \pi_n]$ on $n(\pi) \equiv n$ elements is a permutation in which each element is labeled by a sign of plus or minus. A *segment* of π is a sequence of consecutive elements $\pi_i, \pi_{i+1}, \ldots, \pi_k$, where $1 \leq i \leq k \leq n$. A *reversal* ρ is an operation that reverses the order of the elements in a segment and also flips their signs. Two segments $\pi_i, \pi_{i+1}, \ldots, \pi_k$ and $\pi_j, \pi_{j+1}, \ldots, \pi_l$ are said to be *contiguous* if $j = k+1$ or $i = l+1$. A *transposition* τ is an operation that exchanges two contiguous (disjoint) segments. A *transreversal* $\tau\rho_{A,B}$ (respectively, $w\tau\rho_{B,A}$) is a transposition that exchanges two segments A and B and also reverses A (respectively, B). A *revrev* operation $\rho\rho$ reverses each of the two contiguous segments (without transposing them). The problem of finding a shortest sequence of transposition, transreversal, and revrev operations that transforms a permutation into the identity permutation is called *sorting by transpositions, transreversals, and revrevs*. The *distance* of a permutation π, denoted by $d(\pi)$, is the length of the shortest sorting sequence.

Key Results

Linear vs. Circular Permutations

An operation is said to *operate* on the affected segments as well as on the elements in those segments. Two operations μ and μ' are *equivalent* if they have the same rearrangement result, i.e., $\mu \cdot \pi = \mu' \cdot \pi$ for all π. In this work [8], Hartman and Sharan showed that for an element x of a circular permutation π, if μ is an operation that operates on x, then there exists an equivalent operation μ' that does not operate on x. Based on this property, they further proved that the problem of sorting by transpositions, transreversals, and revrevs is



Sorting by Transpositions and Reversals (Approximate Ratio 1.5), Fig. 1 (**a**) The equivalence of transreversal and revrev on circular permutations. (**b**) The breakpoint graph $G(\pi)$ of the permutation $\pi = [1, -4, 6, -5, 2, -7, -3]$, for which $f(\pi) = [1, 2, 8, 7, 11, 12, 10, 9, 3, 4, 14, 13, 6, 5]$. It is convenient to draw $G(\pi)$ on a circle such that *black edges* (i.e., *thick lines*) are on the circumference and *gray edges* (i.e., *thin lines*) are chords

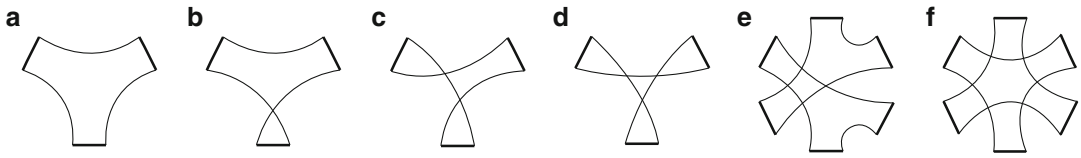
equivalent for linear and circular permutations. Moreover, they observed that revrevs and transreversals are equivalent operations for circular permutations (as illustrated in Fig. 1a), implying that the problem of sorting a linear/circular permutation by transpositions, transreversals, and revrevs can be reduced to that of sorting a circular permutation by transpositions and transreversals only.

The Breakpoint Graph

Given a signed permutation π on {1,2,...,n} of n elements, it is transformed into an unsigned permutation $f(\pi) = \pi' = [\pi'_1, \pi'_2, \ldots, \pi'_{2n}]$ on {1,2,...,2 n} of $2n$ elements by replacing each positive element i with two elements $2i - 1, 2i$ (in this order) and each negative element $-i$ with $2i, 2i - 1$. The extended $f(\pi)$ is considered here as a circular permutation by identifying $2n + 1$ and 1 in both indices and elements. To ensure that every operation on $f(\pi)$ can be mimicked by an operation on π, only operations that cut before odd position are allowed for $f(\pi)$. The *breakpoint graph* $(G\pi)$ is an edge-colored graph on $2n$ vertices $\{1, 2, \ldots, 2n\}$, in which for every $1 \le i \le n, \pi'_{2i}$ is joined to π'_{2i+1} by a black edge and $2i$ is joined to $2i + 1$ by a gray edge (e.g., see Fig. 1b). Since the degree of each vertex in $G(\pi)$ is exactly 2, $G(\pi)$ uniquely decomposes into cycles. A *k-cycle* (i.e., a cycle of *lengthk*) is a cycle with k black edges, and it is *odd* if k is odd. The number of odd cycles in $G(\pi)$ is denoted by $c_{\text{odd}}(\pi)$. It is not hard to verify that $G(\pi)$ consists of n 1-cycles, and hence, $c_{\text{odd}}(\pi) = n$, if π is an identity permutation $[1, 2, \ldots, n]$. Gu et al. [5] have shown that $c_{\text{odd}}(\mu \cdot \pi) \le c_{\text{odd}}(\pi) + 2$ for all linear permutations π and operations μ. In this work [8], Hartman and Sharan further noted that the above result holds also for circular permutations and proved that the lower bound of $d(\pi)$ is $(n(\pi) - c_{\text{odd}}(\pi))/2$.

Transformation into 3-Permutations

A permutation is called *simple* if its breakpoint graph contains only k-cycle, where $k \le 3$. A simple permutation is also called a *3-permutation* if it contains no 2-cycles. A transformation from π to $\hat{\pi}$ is said to be *safe* if $n(\pi) - c_{\text{odd}}(\pi) = n(\hat{\pi}) - c_{\text{odd}}(\hat{\pi})$. It has been shown that every permutation π can be transformed into a simple one π' by safe transformations and, moreover, every sorting of π' mimics a sorting of π with the same number of operations [6, 11]. Here, Hartman and Sharan [8] further showed that every simple permutation π' can be transformed into a 3-permutation $\hat{\pi}$ by safe paddings (of transforming those 2-cycles into 1-twisted 3-cycles) and, moreover, every sorting of $\hat{\pi}$ mimics a sorting of π' with the same number of operations. Hence, based on these two properties, an arbitrary permutation π can be transformed into a 3-permutation $\hat{\pi}$ such that every sorting of $\hat{\pi}$ mimics a sorting of π with the



Sorting by Transpositions and Reversals (Approximate Ratio 1.5), Fig. 2 Configurations of 3-cycles. (**a**) Unoriented, 0-twisted 3-cycle. (**b**) Unoriented, 1-twisted 3-cycle. (**c**) Oriented, 2-twisted 3-cycle. (**d**) Oriented, 3-twisted 3-cycle. (**e**) A pair of intersecting 3-cycles. (**f**) A pair of interleaving 3-cycles

same number of operations, suggesting that one can restrict attention to circular 3-permutations only.

Cycle Types

An operation that cuts some black edges is said to *act* on these edges. An operation is further called a *k-operation* if it increases the number of odd cycles by k. A (0, 2,2)-*sequence* is a sequence of three operations, of which the first is a 0-operation and the next two are 2-operations. An odd cycle is called *oriented* if there is a 2-operation that acts on three of its black edges; otherwise, it is *unoriented*. A *configuration* of cycles is a subgraph of the breakpoint graph that contains one ore more cycles. As shown in Fig. 2a–d, there are four possible configurations of single 3-cycles. A black edge is called *twisted* if its two adjacent gray edges cross each other in the circular breakpoint graph. A cycle is k-twisted if k of its black edges is twisted. For example, the 3-cycles in Fig. 2a–d are 0-, 1-, 2-, and 3-twisted, respectively. Hartman and Sharan observed that a 3-cycle is oriented if and only if it is 2- or 3-twisted.

Cycle Configurations

Two pairs of black edges are called *intersecting* if they alternate in the order of their occurrence along the circle. A pair of black edges *intersects* with cycle C, if it intersects with a pair of black edges that belong to C. Cycles C and *Dintersect* if there is a pair of black edges in C that intersects with D (see Fig. 2e). Two intersecting cycles are called *interleaving* if their black edges alternate in their order of occurrence along the circle (see Fig. 2f). Clearly, the relation between two cycles is one of (1) nonintersecting, (2) intersecting but non-interleaving, and (3) interleaving. A pair of black edges is *coupled* if they are connected by a gray edge and when reading the edges along the cycle, they are read in the same direction. For example, all pairs of black edges in Fig. 2a are coupled. Gu et al. [5] have shown that given a pair of coupled black edges (b_1, b_2), there exists a cycle C that intersects with (b_1, b_2). A *1-twisted pair* is a pair of 1-twisted cycles, whose twists are consecutive on the circle in a configuration that consists of these two cycles only. A 1-twisted cycle is called *closed* in a configuration if its two coupled edges intersect with some other cycle in the configuration. A configuration is *closed* if at least one of its 1-twisted cycles is closed; otherwise, it is called *open*.

The Algorithm

The basic ideas of the Hartman and Sharan's 1.5-approximation algorithm [8] for the problem of sorting by transpositions, transreversals, and revrevs are as follows. Hartman and Sharan reduced the problem to that of sorting a circular 3-permutation by transpositions and transreversals only and then focused on transforming the 3-cycles into 1-cycles in the breakpoint graph of this 3-permutation. By definition, an oriented (i.e., 2- or 3-twisted) 3-cycle admits a 2-operation and, therefore, they continued to consider unoriented (i.e., 0- or 1-twisted) 3-cycles only. Since configurations involving only 0-twisted 3-cycles were handled with (0,2,2)-sequences in [7], Hartman and Sharan restricted their attention to those configurations that consist of 0- and 1-twisted 3-cycles. They showed that these configurations are all closed and that it can be sorted by a (0,2,2)-sequence of operations for each of the following five possible closed configurations: (1) a closed configuration with two unoriented, interleaving 3-cycles that do not form a 1-twisted pair; (2) a closed configuration with two intersecting, 0-twisted 3-cycles; (3) a closed configuration with two intersecting, 1-twisted 3-cycles; (4) a closed

configuration with a 0-twisted 3-cycles that intersects with the coupled edges of a 1-twisted 3-cycle; and (5) a closed configuration that contains $k \geq 2$ mutually interleaving 1-twisted 3-cycles such that all their twists are consecutive on the circle and k is maximal with this property. As a result, the sequence of operations used by Hartman and Sharan in their algorithm contains only 2-operations and (0,2,2)-sequences. Since every sequence of three operations increases the number of odd cycles by at least 4 out of 6 possible in 3 steps, the ratio of their approximation algorithm is 1.5. Furthermore, Hartman and Sharan showed that their algorithm can be implemented in $O(n^{3/2}\sqrt{\log n})$ time using the data structure of Kaplan and Verbin [10], where n is the number of elements in the permutation.

Theorem 1 *The problem of sorting linear permutations by transpositions, transreversals, and revrevs is linearly equivalent to the problem of sorting circular permutations by transpositions, transreversals, and revrevs.*

Theorem 2 *There is a 1.5-approximation algorithm for sorting by transpositions, transreversals, and revrevs, which runs in $O(n^{3/2}\sqrt{\log n})$ time.*

Applications

When trying to determine evolutionary distance between two organisms using genomic data, biologists may wish to reconstruct the sequence of evolutionary events that have occurred to transform one genome into the other. One of the most promising ways to do this phylogenetic study is to compare the order of appearance of identical (e.g., orthologous) genes in two different genomes [9, 12]. This comparison of computing global rearrangement events (such as reversals, transpositions, and transreversals of genome segments) may provide more accurate and robust clues to the evolutionary process than the analysis of local point mutations (i.e., substitutions, insertions, and deletions of nucleotides/amino acids). Usually, the two genomes being compared are represented by signed permutations, with each element standing for a gene and its sign representing the (transcriptional) direction of the corresponding gene on a chromosome. Then the goal of the resulting genome rearrangement problem is to find a shortest sequence of rearrangement operations that transforms (or, equivalently, *sorts*) one permutation into the other. Previous work focused on the problem of sorting a permutation by reversals. This problem has been shown by Capara [2] to be NP-hard, if the considered permutation is unsigned. However, for signed permutations, this problem becomes tractable and Hannenhalli and Pevzer [6] gave the first polynomial-time algorithm for it. On the other hand, there has been less progress on the problem of sorting by transpositions. Thus far, the complexity of this problem is still open, although several 1.5-approximation algorithms [1, 3, 7] have been proposed for it. Recently, the approximation ratio of sorting by transpositions was further improved to 1.375 by Elias and Hartman [4]. Gu et al. [5] and Lin and Xue [11] gave quadratic-time 2-approximation algorithms for sorting signed, linear permutations by transpositions and transreversals. In [11], Lin and Xue considered the problem of sorting signed, linear permutations by transpositions, transreversals, and revrevs and proposed a quadratic-time 1.75-approximation algorithm for it. In this work [8], Hartman and Sharan further showed that this problem is equivalent for linear and circular permutations and can be reduced to that of sorting signed, circular permutations by transpositions and transreversals only. In addition, they provided a 1.5-approximation algorithm that can be implemented in $O(n^{3/2}\sqrt{\log n})$ time.

Cross-References

- Sorting Signed Permutations by Reversal (Reversal Distance)
- Sorting Signed Permutations by Reversal (Reversal Sequence)

Recommended Reading

1. Bafna V, Pevzner PA (1998) Sorting by transpositions. SIAM J Discret Math 11:224–240

2. Caprara A (1999) Sorting permutations by reversals and Eulerian cycle decompositions. SIAM J Discret Math 12:91–110
3. Christie DA (1999) Genome rearrangement problems. Ph.D. thesis, Department of Computer Science. University of Glasgow, U.K.
4. Elias I, Hartman T (2006) A 1.375-approximation algorithm for sorting by transpositions. IEEE/ACM Trans Comput Biol Bioinform 3:369–379
5. Gu QP, Peng S, Sudborough H (1999) A 2-approximation algorithm for genome rearrangements by reversals and transpositions. Theor Comput Sci 210:327–339
6. Hannenhalli S, Pevzner PA (1999) Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. J Assoc Comput Mach 46:1–27
7. Hartman T, Shamir R (2006) A simpler and faster 1.5-approximation algorithm for sorting by transpositions. Inf Comput 204:275–290
8. Hartman T, Sharan R (2004) A 1.5-approximation algorithm for sorting by transpositions and transreversals. In: Proceedings of the 4th workshop on algorithms in bioinformatics (WABI'04), Bergen, pp 50–61, 17–21 Sept (2004)
9. Hoot SB, Palmer JD (1994) Structural rearrangements, including parallel inversions, within the chloroplast genome of Anemone and related genera. J Mol Evol 38:274–281
10. Kaplan H, Verbin E (2003) Efficient data structures and a new randomized approach for sorting signed permutations by reversals. In: Proceedings of the 14th annual symposium on combinatorial pattern matching (CPM'03), Morelia, pp 170–185, 25–27 June (2003)
11. Lin GH, Xue G (2001) Signed genome rearrangements by reversals and transpositions: models and approximations. Theor Comput Sci 259:513–531
12. Palmer JD, Herbon LA (1986) Tricircular mitochondrial genomes of Brassica and Raphanus: reversal of repeat configurations by inversion. Nucleic Acids Res 14:9755–9764

Sorting Signed Permutations by Reversal (Reversal Distance)

David A. Bader
College of Computing, Georgia Institute of Technology, Atlanta, GA, USA

Keywords

Inversion distance; Reversal distance; Sorting by reversals

Years and Authors of Summarized Original Work

2001; Bader, Moret, Yan

Problem Definition

This entry describes algorithms for finding the minimum number of steps needed to sort a signed permutation (also known as inversion distance, reversal distance). This is a real-world problem and, for example, is used in computational biology.

Inversion distance is a difficult computational problem that has been studied intensively in recent years [1, 4, 6–10]. Finding the inversion distance between unsigned permutations is NP-hard [7], but with signed ones, it can be done in linear time [1].

Key Results

Bader et al. [1] present the first worst-case linear-time algorithm for computing the reversal distance that is simple and practical and runs faster than previous methods. Their key innovation is a new technique to compute connected components of the overlap graph using only a stack, which results in the simple linear-time algorithm for computing the inversion distance between two signed permutations. Bader et al. provide ample experimental evidence that their linear-time algorithm is efficient in practice as well as in theory: they coded it as well as the algorithm of Berman and Hannenhalli, using the best principles of algorithm engineering to ensure that both implementations would be as efficient as possible and compared their running times on a large range of instances generated through simulated evolution.

Bafna and Pevzner introduced the cycle graph of a permutation [3], thereby providing the basic data structure for inversion distance computations. Hannenhalli and Pevzner then developed the basic theory for expressing the inversion distance in easily computable terms

(number of breakpoints minus number of cycles plus number of hurdles plus a correction factor for a fortress [3, 15]-hurdles and fortresses are easily detectable from a connected component analysis). They also gave the first polynomial-time algorithm for sorting signed permutations by reversals [9]; they also proposed a $O(n^4)$ implementation of their algorithm which runs in quadratic time when restricted to distance computation. Their algorithm requires the computation of the connected components of the overlap graph, which is the bottleneck for the distance computation. Berman and Hannenhalli later exploited some combinatorial properties of the cycle graph to give a $O(n\alpha(n))$ algorithm to compute the connected components, leading to a $O(n^2\alpha(n))$ implementation of the sorting algorithm [6], where α is the inverse Ackerman function. (The later Kaplan-Shamir-Tarjan (KST) algorithm [10] reduces the time needed to compute the shortest sequence of inversions, but uses the same algorithm for computing the length of that sequence.)

No algorithm that actually builds the overlap graph can run in linear time, since that graph can be of quadratic size. Thus, Bader's key innovation is to construct an *overlap forest* such that two vertices belong to the same tree in the forest exactly when they belong to the same connected component in the overlap graph. An overlap forest (the composition of its trees is unique, but their structure is arbitrary) has exactly one tree per connected component of the overlap graph and is thus of linear size. The linear-time step for computing the connected components scans the permutation twice. The first scan sets up a trivial forest in which each node is its own tree, labeled with the beginning of its cycle. The second scan carries out an iterative refinement of this first forest, by adding edges and so merging trees in the forest; unlike a Union-Find, however, this algorithm does not attempt to maintain the trees within certain shape parameters. This step is the key to Bader's linear-time algorithm for computing the reversal distance between signed permutations.

Applications

Some organisms have a single chromosome or contain single-chromosome organelles (such as mitochondria or chloroplasts), the evolution of which is largely independent of the evolution of the nuclear genome. Given a particular strand from a single chromosome, whether linear or circular, we can infer the ordering and directionality of the genes, thus representing each chromosome by an ordering of oriented genes. In many cases, the evolutionary process that operates on such single-chromosome organisms consists mostly of inversions of portions of the chromosome; this finding has led many biologists to reconstruct phylogenies based on gene orders, using as a measure of evolutionary distance between two genomes the inversion distance, i.e., the smallest number of inversions needed to transform one signed permutation into the other [11, 12, 14].

The linear-time algorithm is in wide use (as it has been cited nearly 200 times within the first several years of its publication). Examples include the handling multichromosomal genome rearrangements [16], genome comparison [5], parsing RNA secondary structure [13], and phylogenetic study of the HIV-1 virus [2].

Open Problems

Efficient algorithms for computing minimum distances with weighted inversions, transpositions, and inverted transpositions are open.

Experimental Results

Bader et al. give experimental results in [1].

URL to Code

An implementation of the linear-time algorithm is available as *C* code from www.cc.gatech.edu/~bader. Two other dominated implementations are available that are designed to compute the shortest sequence of inversions as well as its length:

one, due to Hannenhalli that implements his first algorithm [9], which runs in quadratic time when computing distances, while the other, a Java applet written by Mantin (http://www.math.tau.ac.il/~rshamir/GR/), that implements the KST algorithm [10], but uses an explicit representation of the overlap graph and thus also takes quadratic time. The implementation due to Hannenhalli is very slow and implements the original method of Hannenhalli and Pevzner and not the faster one of Berman and Hannenhalli. The KST applet is very slow as well since it explicitly constructs the overlap graph.

Cross-References

For finding the actual sorting sequence, see the entry:

- ▸ Sorting Signed Permutations by Reversal (Reversal Sequence)

Recommended Reading

1. Bader DA, Moret BME, Yan M (2001) A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. J Comput Biol 8(5):483–491. An earlier version of this work appeared In: The proceedings of 7th Int'l workshop on algorithms and data structures (WADS 2001)
2. Badimo A, Bergheim A, Hazelhurst S, Papathanasopolous M, Morris L (2003) The stability of phylogenetic tree construction of the HIV-1 virus using genome-ordering data versus env gene data. In: Proceedings of ACM annual research conference of the South African institute of computer scientists and information technologists on enablement through technology (SAICSIT 2003), Port Elizabeth, Sept 2003, vol 47. ACM, Fourways, South Africa, pp 231–240
3. Bafna V, Pevzner PA (1993) Genome rearrangements and sorting by reversals. In: Proceedings of 34th annual IEEE symposium on foundations of computer science (FOCS93), Palo Alto, CA, pp 148–157. IEEE Press
4. Bafna V, Pevzner PA (1996) Genome rearrangements and sorting by reversals. SIAM J Comput 25:272–289
5. Bergeron A, Stoye J (2006) On the similarity of sets of permutations and its applications to genome comparison. J Comput Biol 13(7):1340–1354
6. Berman P, Hannenhalli S (1996) Fast sorting by reversal. In: Hirschberg DS, Myers EW (eds) Proceedings of 7th annual symposium combinatorial pattern matching (CPM96), Laguna Beach, June 1996. Lecture notes in computer science, vol 1075. Springer, pp 168–185
7. Caprara A (1997) Sorting by reversals is difficult. In: Proceedings of 1st conference on computational molecular biology (RECOMB97), Santa Fe. ACM, pp 75–83
8. Caprara A (1999) Sorting permutations by reversals and Eulerian cycle decompositions. SIAM J Discret Math 12(1):91–110
9. Hannenhalli S, Pevzner PA (1995) Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In: Proceedings of 27th annual symposium on theory of computing (STOC95), Las Vegas. ACM, pp 178–189
10. Kaplan H, Shamir R, Tarjan RE (1999) A faster and simpler algorithm for sorting signed permutations by reversals. SIAM J Comput 29(3):880–892. First appeared In: Proceedings of 8th annual symposium on discrete algorithms (SODA97), New Orleans. ACM, pp 344–351
11. Olmstead RG, Palmer JD (1994) Chloroplast DNA systematics: a review of methods and data analysis. Am J Bot 81:1205–1224
12. Palmer JD (1992) Chloroplast and mitochondrial genome evolution in land plants. In: Herrmann R (ed) Cell organelles. Springer, Vienna, pp 99–133
13. Rastegari B, Condon A (2005) Linear time algorithm for parsing RNA secondary structure. In: Casadio R, Myers E (eds) Proceedings of 5th workshop algorithms in bioinformatics (WABI'05), Mallorca. Lecture notes in computer science, vol 3692. Springer, Mallorca, Spain, pp 341–352
14. Raubeson LA, Jansen RK (1992) Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants. Science 255:1697–1699
15. Setubal JC, Meidanis J (1997) Introduction to computational molecular biology. PWS, Boston
16. Tesler G (2002) Efficient algorithms for multichromosomal genome rearrangements. J Comput Syst Sci 63(5):587–609

Sorting Signed Permutations by Reversal (Reversal Sequence)

Eric Tannier
LBBE Biometry and Evolutionary Biology, INRIA Grenoble Rhône-Alpes, University of Lyon, Lyon, France

Keywords

Bioinformatics; Computational biology; Genome evolution; Genome rearrangements; Inversion distance; Sorting by inversions

Years and Authors of Summarized Original Work

2004; Tannier, Sagot

Problem Definition

A *signed permutation* π of size n is a permutation over $\{-n, \ldots, -1, 1 \ldots n\}$, where $\pi_{-i} = -\pi_i$ for all i. We note $\pi = (\pi_1, \ldots, \pi_n)$.

The *reversal* $\rho = \rho_{i,j} (1 \leq i \leq j \leq n)$ is an operation that reverses the order and flips the signs of the elements $\pi_i, \ldots, \pi_j$ in a permutation π:

$$\pi \cdot \rho = (\pi_1, \ldots, \pi_{i-1}, -\pi_j, \ldots, -\pi_i \pi_{j+1}, \ldots, \pi_n).$$

If $\rho_1, \ldots, \rho_k$ is a sequence of reversals, it is said to *sort* a permutation π if $\pi \cdots \rho_1 \cdots \rho_k = Id$, where $Id = (1, 2, \ldots, n)$ is the identity permutation. The length of a shortest sequence of reversals sorting π is called the *reversal distance* of π and is denoted by $d(\pi)$.

If the computation of $d(\pi)$ is solved in linear time [3] (see the entry ▸ Sorting Signed Permutations by Reversal (Reversal Distance)), the computation of a sequence $\rho^1, \ldots, \rho^k$ of size $k = d(\pi)$ that sorts π is more complicated, and no linear time algorithm is known so far. The best complexity is currently achieved by the subquadratic solution of Tannier and Sagot [17], which has later been improved by Tannier, Bergeron and Sagot [18], and Han [9].

Key Results

The $O(n^4)$ Self-Reduction

Recall there is a linear algorithm to compute the reversal distance thanks to the formula $d(\pi) = n + 1 - c(\pi) + t(\pi) + h(\pi) + f(\pi)$, where $c(\pi)$ is the number of cycles in the breakpoint graph and $h(\pi) + f(\pi)$ is computed from the unoriented components of the permutation (see the entry ▸ Sorting Signed Permutations by Reversal (Reversal Distance)). Once this is known, the self-reduction technique trivially computes a sequence of size $d(\pi)$: try every possible reversal ρ at one step, until you find one such that $d(\pi \cdot \rho) = d(\pi) - 1$. Such a reversal is called a *sorting reversal*. This necessitates $O(n)$ computations for every possible reversal. There are at most $n(n+1)/2 = O(n^2)$ reversals to try, so iterating this to find a sequence yields an $O(n^4)$ algorithm.

The first polynomial algorithm by Hannenhalli and Pevzner [10] was not achieving a better complexity, and the algorithmic study of finding the shortest sequences of reversals began its history.

The Quadratic Roof

All the published solutions for the computations of a sorting sequence are divided into two, following the division of the distance formula into its parameters: a first part computes a sequence of reversals so that the resulting permutation has no unoriented component, and a second part sorts all oriented components.

The first part was given its best solution by Kaplan, Shamir, and Tarjan [12], whose algorithm runs in linear time when coupled with the linear distance computation [3], and it is based on Hannenhalli and Pevzner's [10] early results.

The second part is the bottleneck of the whole procedure. At this point, if there is no unoriented component, the distance is $d(\pi) = n + 1 - c(\pi)$, so a sorting reversal is one that increases $c(\pi)$ and does not create unoriented components.

A reversal that increases $c(\pi)$ is called *oriented*. Finding an oriented reversal is an easy part: any two consecutive numbers that have different signs in the permutation define one. This can easily be done in linear time or sublinear with ad hoc data structures to maintain the permutation during the scenario. The hard part is to make sure it does not create unoriented components.

The quadratic solutions (see, e.g., the one of Kaplan, Shamir, and Tarjan [12]) are based on the linear recognition of sorting reversals. No better algorithm is known so far to recognize sorting reversals, and it seemed that a lower bound had been reached, as witnessed by a survey

of Ozery-Flato and Shamir [15] in which they wrote that "a central question in the study of genome rearrangements is whether one can obtain a subquadratic algorithm for sorting by reversals." This was obtained by Tannier and Sagot [17], who proved that the recognition of sorting reversal at each step is not necessary, but only the recognition of oriented reversals.

A Promising New but Still Quadratic Method

The algorithm is based on the following theorem, taken from [18]. A sequence of oriented reversals $\rho_1, \ldots, \rho_k$ is said to be *maximal* if there is no oriented reversal in $\pi \cdot \rho_1 \ldots \rho_k$. In particular a sorting sequence is maximal, but the converse is not true.

Theorem 1 *If S is a maximal but not a sorting sequence of oriented reversals for a permutation, then there exists a nonempty sequence S' of oriented reversals such that S may be split into two parts $S = S_1, S_2$, and S_1, S', S_2 is a sequence of oriented reversal.*

This allows to construct sequences of oriented reversals instead of sorting reversals, increase their size by adding reversals inside the sequence instead of at the end, and obtain a sorting sequence.

This algorithm, with a classical data structure to represent permutations (e.g., as an array), has still an $O(n^2)$ complexity, because at each step it has to test the presence of an oriented reversal and apply it to the permutation.

Composing with Data Structures

The slight modification of a data structure invented by Kaplan and Verbin [11] allows to pick and apply an oriented reversal in $O(\sqrt{n \log n})$, and using this, Tannier and Sagot's algorithm achieves $O(n^{3/2}\sqrt{\log n})$ time complexity.

Han [9] announced another data structure that allows to pick and apply an oriented reversal in $O(\sqrt{n})$ time, and integrating this to the algorithm can plausibly decrease the complexity of the overall method to $O(n^{3/2})$. Swenson et al. [16] gave an $O(n \log n)$ solution for picking oriented reversals, but their attempts of integrating it to the overall procedure seems to fail on worst cases.

Extensions

Once sorting by reversals has reached its best solutions, there are natural extensions guided by the main motivation for the problem in computational biology: sample among optimal solutions, and handle several permutations and more operations than just the reversal.

Counting optimal solutions is conjectured to be #P-complete [14], but sampling almost uniformly from the solution space is still open, and has been given a heuristic solution [14], including suboptimal solutions in the sample.

Algorithms to enumerate all sorting reversals at one step have also been worked out [4], which provides a way for enumeration. A structure of the solution space was proposed, but with a possibly exponential number of objects to enumerate [5].

The median problem consists in handling more than one permutation and is a particular case of the so-called small parsimony problem, which consists in reconstructing ancestral states in a phylogenetic context. Additional operations can be transpositions, duplications, or many others. Many generalizations and variants have been listed in a book on Combinatorics of Genome Rearrangements [8]. Almost all are NP-hard.

Applications

The motivation as well as the main application of this problem is in computational biology. Signed permutations are an adequate object to model the relative position and orientation of homologous segments of DNA in two species.

Reversal scenarios were used to test some evolutionary properties, like the propension of rearrangement to cut around the replication origin [1] or the fragility of certain genomic regions [2]. But evolutionary hypotheses can hardly be

tested from a single optimal solution; this would necessitate a better view of the solution space.

The gain of complexity for sorting by reversals inspired many other algorithmic works, and several problems in genome rearrangement found a better solution thanks to the subquadratic gain described here. But the computational difficulties of the problem (parameters $h(\pi)$ and $f(\pi)$, additional complexity for generating a scenario compared to the distance calculation, NP-completeness of every generalization with more operations, more permutations, more realistic models) lead most computational biologists to progressively abandon the reversal model for simpler ones (DCJ [19], SCJ [7]).

Sometimes heroic gains in complexity are worth for computer science but seem just like going a bit further in a dead end for applications. Research consists in breaking walls without always knowing if behind there is a space for a community to work in or another thicker wall.

Open Problems

Still there are a couple of questions that remain unsolved before closing (or reopening?) this entry:

- I conjecture that the "real" complexity of giving a reversal scenario is $O(n \log n)$. It is more or less what Swenson et al. [16] also claim, but without giving a full proof.
- Counting and sampling, even approximately, are open. I learned this interesting conjecture from Istvan Miklos: is it possible to walk in the entire space of sequences of sorting reversals by small transformations of scenarios, consisting at each step to change at most 4 reversals? This would be a first step to design an almost uniform sampler.

Experimental Results

To my knowledge the data structure that allows the subquadratic complexity described in this entry has never been implemented. The size of the data, as well as the limited possibilities of applications of handling only two genomes and a single optimal solution, makes the subquadratic version, while a good piece of algorithmics, not really worth for applications.

URL to Code

- There are a few old programs still able to give a sorting sequence of reversals: in San Diego http://grimm.ucsd.edu/GRIMM/, New Mexico www.cs.unm.edu/~moret/GRAPPA/, or Tel Aviv www.math.tau.ac.il/~rshamir/GR/ and more recent ones in Lyon http://doua.prabi.fr/software/luna or Bielefeld http://bibiserv.techfak.uni-bielefeld.de/dcj/welcome.html.
- The standard software for Bayesian sampling in the space of sorting sequences (including nonoptimal ones) is Badger http://bibiserv.techfak.uni-bielefeld.de/dcj/welcome.html, and there is also one biased to optimal solutions called DCJ2HP http://www.renyi.hu/~miklosi/DCJ2HP/ that uses a parallel tempering between DCJ solutions (easier to sample) and reversals solutions.

Cross-References

▸ Sorting Signed Permutations by Reversal (Reversal Distance)

Recommended Reading

1. Ajana Y, Lefebvre J-F, Tillier E, El-Mabrouk N (2002) Exploring the set of all minimal sequences of reversals – an application to test the replication-directed reversal hypothesis. In: Proceedings of the second workshop on algorithms in bioinformatics. Lecture notes in computer science, vol 2452. Springer, Berlin, pp 300–315
2. Attie O, Darling A, Yancopoulos Y (2011) The rise and fall of breakpoint reuse depending on genome resolution. BMC Bioinform 12(supp 9):S1
3. Bader DA, Moret BME, Yan M (2001) A linear-time algorithm for computing inversion distance between

signed permutations with an experimental study. J Comput Biol 8(5):483–491
4. Badr G, Swenson KM, Sankoff D (2011) Listing all parsimonious **reversal** sequences: new algorithms and perspectives. J Comput Biol 18:1201–1210
5. Braga MDV, Sagot MF, Scornavacca C, Tannier E (2008) Exploring the solution space of sorting by reversals with experiments and an application to evolution. IEEE-ACM Trans Comput Biol Bioinform 5:348–356
6. Darling AE, Miklós I, Ragan MA (2008) Dynamics of genome rearrangement in bacterial populations. PLoS Genet (7):e1000128
7. Feijão P, Meidanis J (2011) SCJ: a breakpoint-like distance that simplifies several rearrangement problems. IEEE/ACM Trans Comput Biol Bioinform 8(5):1318–1329
8. Fertin G, Labarre A, Rusu I, Tannier E, Vialette S (2009) Combinatorics of genome rearrangements. MIT, Cambridge
9. Han Y (2006) Improving the efficiency of sorting by reversals. In: Proceedings of the 2006 international conference on bioinformatics and computational biology, Las Vegas
10. Hannenhalli S, Pevzner PA (1999) Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. J ACM (JACM) 46:1–27
11. Kaplan H, Verbin E (2003) Efficient data structures and a new randomized approach for sorting signed permutations by reversals. In: Proceedings of CPM'03. Lecture notes in computer science, vol 2676. Springer, Berlin/Heidelberg, pp 170–185
12. Kaplan H, Shamir R, Tarjan RE (1999) Faster and simpler algorithm for sorting signed permutations by reversals. SIAM J Comput 29:880–892
13. Larget B, Simon DL, Kadane JB (2002) On a Bayesian approach to phylogenetic inference from animal mitochondrial genome arrangements (with discussion). J R Stat Soc B 64:681–693
14. Miklós I, Tannier E (2010) Bayesian sampling of genomic rearrangement scenarios via double cut and join. Bioinformatics 26:3012–3019
15. Ozery-Flato M, Shamir R (2003) Two notes on genome rearrangement. J Bioinform Comput Biol 1:71–94
16. Swenson KM, Rajan V, Lin Y, Moret BME (2010) Sorting signed permutations by inversions in $O(n \log n)$ time. J Comput Biol 17:489–501
17. Tannier E, Sagot M-F (2004) Sorting by reversals in subquadratic time. In: Proceedings of CPM'04. Lecture notes in computer science, vol 3109. Springer, Berlin/Heidelberg, pp 1–13
18. Tannier E, Bergeron A, Sagot M-F (2006) Advances on sorting by reversals. Discret Appl Math 155:881–888
19. Yancopoulos S, Attie O, Friedberg R (2005) Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 21:3340–3346

Spanning Trees with Low Average Stretch

Ittai Abraham[1] and Ofer Neiman[2]
[1]Microsoft Research, Silicon Valley, Palo Alto, CA, USA
[2]Department of Computer Science, Ben-Gurion University of the Negev, Beer Sheva, Israel

Keywords

Embedding; Spanning tree; Stretch

Years and Authors of Summarized Original Work

2012; Abraham, Neiman

Problem Definition

Let $G = (V, E)$ be an undirected graph, with nonnegative weights on the edges $w : E \to \mathbb{R}_+$. Let d_G be the shortest-path metric on G, with respect to the weights. For a spanning (subgraph) tree T of G, define the stretch of an edge $\{u, v\} \in E$ in T as $\text{stretch}_T(u, v) = \frac{d_T(u,v)}{d_G(u,v)}$ and the average stretch as

$$\text{avg} - \text{stretch}_T(G) = \frac{1}{|E|} \sum_{e \in E} \text{stretch}_T(e) .$$

We shall consider the problem of finding a tree T whose average stretch is small. We also study the problem of finding a distribution over spanning trees, such that for all $e \in E$, $\mathbb{E}_T[\text{stretch}_T(e)]$ is small.

Key Results

Low-stretch spanning trees were first studied by [3], who showed that any graph on n vertices has a spanning tree with average stretch $2^{O(\sqrt{\log n \log\log n})}$ and showed a family of graphs

that requires $\Omega(\log n)$ average stretch. Their result was substantially improved by [7], who showed an upper bound of $O(\log^2 n \log\log n)$, and later [1] improved this to a near optimal $\tilde{O}(\log n)$.

The main result discussed here is from [2]:

Theorem 1 *For any graph G with n vertices and m edges, there is a deterministic algorithm that constructs a spanning tree T, such that* $\text{avg} - \text{stretch}_T(G) \leq O(\log n \log\log n)$. *The running time of the algorithm is* $O(m \log n \log\log n)$.

We also show an efficient algorithm to sample from a distribution over spanning trees, such that the expected stretch of any edge is bounded by $O(\log n \log\log n \log\log\log n)$.

Applications

An important problem in algorithm design is obtaining fast algorithms for solving linear systems. For many applications, the matrix is sparse, and while little is known for general sparse matrices, the case of symmetric diagonally dominant (SDD) matrices has received a lot of attention recently. In a seminal sequence of results, Spielman and Teng [12] showed a near-linear time solver for this important case. This solver has proven a powerful algorithmic tool and is used to calculate eigenvalues, obtain spectral graph sparsifiers [11], and approximate maximum flow [6] and many other applications. A basic step in solving these systems $Ax = b$ is combinatorial preconditioning. If one uses the Laplacian matrix corresponding to a spanning tree (and a few extra edges) of the graph whose Laplacian matrix is A, then the condition number depends on the total stretch of the tree. This will improve the run-time of iterative methods, such as conjugate gradient or Chebyshev iterations. See [9, 10] for the latest progress on this direction. In this work we show that one can construct such a spanning tree with both run-time and total stretch bounded by $O(m \log n \log\log n)$.

Probabilistic embedding into trees, introduced by [4], has been a successful paradigm in algorithm design. Many hard optimization problems on graphs can be reduced, via embedding, to a similar problem on a tree, which is often considerably easier. This framework can be applied to approximation algorithms, online algorithms, network design, and other settings. Some of the notable examples are metrical task system, buy-at-bulk network design, the k-server problem, group Steiner tree, etc. An asymptotical optimal result of expected $O(\log n)$ distortion for probabilistic embedding into trees was given by [8]. The trees in the support of the FRT distribution are not subgraphs of the input graph and may contain Steiner nodes and new edges. While this is fine for most applications, there are some that must have trees which are subgraphs, such as minimum cost communication spanning tree: Given a weighted graph $G = (V, E)$ and a requirement matrix $R = (r_{uv})$, the objective is to find a spanning tree T that minimizes $\sum_{u,v \in V} r_{uv} \cdot d_T(u, v)$. Our result implies a $\tilde{O}(\log n)$ approximation.

Petal Decomposition

A basic tool that is often used in constructing tree metrics and spanning trees with low stretch is *sparse graph decomposition*. The idea is to partition the graph into small diameter pieces, such that few edges are cut. Each cluster of the decomposition is partitioned recursively, which yields a hierarchical decomposition. Creating a tree recursively on each cluster of the decomposition, and connecting these in a tree structure, will yield a spanning tree of the graph. The edges cut by the decomposition are potentially stretched by a factor proportional to the diameter of the created tree. The construction has to balance between these two goals: cut a small number of edges and maintain small diameter in the created tree.

One of the main difficulties in such a spanning tree construction is that the radius (The radius of a graph is the maximal distance from a designated center.) may increase by a small factor at every application of the decomposition, which

translates to increased stretch. If we drop the requirement that the tree is a *spanning tree* of the graph and just require a tree metric, then this difficulty does not appear, and indeed, optimal $\Theta(\log n)$ bound is known on the average stretch [5,8]. Our *petal decomposition* allows essentially optimal control on the radius increase of the spanning tree; it increases by at most a factor of 4 over all the recursion levels.

Highways

One of the components in the decomposition scheme is highways. Each cluster $X \subseteq V$ in our decomposition scheme has a designated center $x_0 \in X$ and a "target" $t \in X$. It is guaranteed that the shortest path from x_0 to t will be fully contained in the *final* spanning tree T. This path is called the petal's highway. Intuitively, the highway will provide short paths from the center x_0 to many of the points in the cluster.

Cones and Petals

A cone is a generalization of a ball; the notion of cones was introduced in [7] and was used also in [1] for low-stretch spanning trees. Informally, a cone $C(t,r)$ of radius r centered at t (with respect to the cluster center x_0) contains all the points $z \in X$ such that $d(z,t) + d(t,x_0) \leq d(z,x_0) + r$ (here d is the shortest-path metric on X). In other words, the cone contains all the points for which the path to x_0 through t is not much longer than the direct shortest path to x_0. The parameter r is a bound on the radius increase in the current decomposition.

One way to define a petal is as a *union of cones*. The petal $P(t,r)$ around a target t with radius r is defined as $\bigcup_{0 \leq k \leq r} C(p_k, k/2)$, where p_k is the point of distance $r-k$ from t on the shortest path from t to x_0. The center of the petal is defined as $x = p_0$, and the path from x to t is the petal's highway. The `petal-decomposition` algorithm iteratively picks an arbitrary target of distance at least $3\Delta/4$ (where Δ is the radius of X) away from x_0, generates a petal for it, and removes the petal from the graph. When there are no longer such points, the remaining points will form the central cluster (the stigma). The first petal requires extra care in its target choice, as it may contain the designated target of the cluster, which implies we cannot allow the shortest path to this target to be cut by this or subsequent petals. The radii of the petals are chosen by a region-growing argument that cuts few edges, where the length of the possible range for the radius is $\approx\Delta$. This is in contrast with the previous work, where in order to give an appropriate bound on the radius increase, the range was much smaller than Δ, which immediately translates to a loss in the stretch. The precise method for choosing r is essentially given in [7], and we also give a randomized version similar in spirit to the one in [1].

Fast Petal Construction

The alternative way to define petals and cones is as balls in an appropriately defined *directed* graph created from G. This suggests that we can use a variant of Dijkstra to compute a petal in nearly linear time in the sum of degrees of its vertices. Let $\tilde{G} = (V, A, \bar{w})$ be the weighted directed graph induced by adding the two directed edges $(u,v), (v,u) \in A$ for each $\{u,v\} \in E$ and setting $\bar{w}(u,v) = d(u,v) - (d(v,x_0) - d(u,x_0))$. The cone $C(t,r)$ is simply the ball around t of radius r in $\tilde{G}$. The petal $P(t,r)$ is the ball around t of radius $r/2$ in $\tilde{G}$ with one change: the weight of each edge on the path from t to $x = p_0$ is changed to be $1/2$ of its original weight (i.e., $1/2$ of its weight in G).

Ideas in the Analysis

Informally, the crucial property of a petal and its highway is the following: Assume $z \in P(t,r)$, and $P_{x_0 z}$ is the shortest path from the original center x_0 to z. By forming the petal, we remove all edges between $P(t,r)$ and $X \setminus P(t,r)$ except for the edge from the petal center x toward x_0. Hence, any path from x_0 to z must go through the petal center x. If the new shortest path $P'_{x_0 z}$ (after forming the petal) is (additively) $k/2$ longer than the length of $P_{x_0 z}$, then $z \in C(p_k, k/2)$ and so $P'_{x_0 z}$ will contain part of the new petal's highway of length at least k. Such a property could allow the following wishful thinking: Suppose that in each iteration we increase the distance of a point to the center by at most α but also mark a new

portion of the path of length 2α as edges that are guaranteed to appear in the final tree (part of a highway). In such a case, it is easy to see that the final path will have stretch at most 2: If the original distance was b, once the total increase is b, we have marked $2b$ – all of the path – as a highway that will appear in the tree. Unfortunately, the path from x to z in the final tree may not use the prescribed highway of the parent cluster so the above "wishful thinking" argument does not work.

The key algorithmic idea to alleviate this problem is to decrease the weight of an edge by half when it becomes part of a highway (we ensure that this happens at most once for every edge). This reweighting signals later iterations to use the prescribed highway, as this must remain the shortest path. We maintain the invariant that in every cluster, the highway edges are the only cluster edges which have been reweighted. Now, in every petal (except for maybe the first), we create a *new* petal highway when we form $P(t, r)$. For any $z \in P(t, r)$, *the length of the path from* x_0 *to* z *does not increase at all* (after reweighting the highway): For some $k \leq r$, it increased by at most $k/2$, but a highway length of at least k was reduced by $1/2$.

We have to take care of radius increase generated by the very first petal as well, where it could be that no new highway is created (this petal's highway may be a part of the highway of the original cluster). In this case, we use the fact that the path from x_0 to x_1 (the center of the first petal) must also be on the highway of the original cluster and that its length is at least $\Delta/2$. This implies that even though we may have increased the radius, at least half of the path is guaranteed not to increase ever again. We use a subtle inductive argument to make this intuition precise, and in fact we lose a factor of 2 for each of these cases, so the maximal increase is by a factor of 4.

Recommended Reading

1. Abraham I, Bartal Y, Neiman O (2008) Nearly tight low stretch spanning trees. In: FOCS '08: Proceedings of the 2008 49th annual IEEE symposium on foundations of computer science, Philadelphia. IEEE Computer Society, Washington, DC, pp 781–790
2. Abraham I, Neiman O (2012) Using petal-decompositions to build a low stretch spanning tree. In: Proceedings of the forty-fourth annual ACM symposium on theory of computing, STOC '12, New York. ACM, pp 395–406
3. Alon N, Karp RM, Peleg D, West D (1995) A graph-theoretic game and its application to the k-server problem. SIAM J Comput 24(1):78–100
4. Bartal Y (1996) Probabilistic approximation of metric spaces and its algorithmic applications. In: Proceedings of the 37th annual symposium on foundations of computer science, Burlington. IEEE Computer Society, Washington, DC, pp 184–
5. Bartal Y (2004) Graph decomposition lemmas and their role in metric embedding methods. In: Albers S, Radzik T (eds) Proceedings of the 12th annual European symposium on algorithms – ESA 2004, Bergen, 14–17 Sept 2004. Volume 3221 of Lecture notes in computer science. Springer, pp 89–97
6. Christiano P, Kelner JA, Madry A, Spielman DA, Teng S-H (2011) Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs. In: Proceedings of the 43rd annual ACM symposium on theory of computing, STOC '11, San Jose. ACM, New York, pp 273–282
7. Elkin M, Emek Y, Spielman DA, Teng S-H (2005) Lower-stretch spanning trees. In: Proceedings of the thirty-seventh annual ACM symposium on theory of computing, STOC '05, Baltimore. ACM, New York, pp 494–503
8. Fakcharoenphol J, Rao S, Talwar K (2003) A tight bound on approximating arbitrary metrics by tree metrics. In: Proceedings of the thirty-fifth annual ACM symposium on theory of computing, STOC '03, San Diego. ACM, New York, pp 448–455
9. Koutis I, Miller GL, Peng R (2010) Approaching optimality for solving SDD linear systems. In: 51th annual IEEE symposium on foundations of computer science, 23–26 Oct 2010, Las Vegas, pp 235–244
10. Koutis I, Miller GL, Peng R (2011) A nearly $O(m \log n)$ time solver for SDD linear systems. In: 52th annual IEEE symposium on foundations of computer science, Palm Springs
11. Spielman DA, Srivastava N (2008) Graph sparsification by effective resistances. In: Proceedings of the 40th annual ACM symposium on theory of computing, STOC '08, Victoria. ACM, New York, pp 563–568
12. Spielman DA, Teng S-H (2004) Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In: Proceedings of the thirty-sixth annual ACM symposium on theory of computing, STOC '04, Chicago. ACM, New York, pp 81–90

Sparse Fourier Transform

Eric Price
Department of Computer Science, The University of Texas, Austin, TX, USA

Keywords

Compressed sensing; Sparse recovery

Years and Authors of Summarized Original Work

2012; Hassanieh, Indyk, Katabi, Price

Problem Definition

Suppose that we have access to a vector $x \in \mathbb{C}^n$. How much time does it take to compute its Fourier transform $\hat{x}$? One can do this with the Fast Fourier Transform (FFT) in $O(n \log n)$ time. But can we do better?

We do not know the answer in general, but some classes of algorithms cannot do better [1, 20] and certainly one cannot do better than $O(n)$ time for arbitrary signals x. But the Fourier transform is ubiquitous in signal processing, appearing in compression of audio, images, and video, in manipulation of audio, and in recovery of radio or MRI signals, so we would really like to do better. If we cannot improve on the FFT in general, then perhaps we can for the signals commonly seen in these applications. To do this, we need some notion for how the signals we typically see are "easier" than arbitrary ones.

One such notion is *sparsity*. The main reason to use the Fourier transform in compression is because it concentrates the energy of the signal into a few large (or "heavy") coordinates and many small ones; signals with such concentrated coordinates are called *sparse*. One can then throw out the small coordinates and only store the heavy ones; this is the main principle behind lossy compression such as MP3 or JPEG. In fact, in all the applications discussed in the previous paragraph, the signals typically have an approximately sparse Fourier transform. This brings us to the problem described in this entry: can we speed up the Fourier transform for signals when the result is approximately sparse?

Moreover, as with lossy compression, we often only care about the heavy coordinates and are willing to tolerate an error proportional to the energy in the small coordinates. This relaxation will allow us to compute the sparse Fourier transform in *sublinear* time.

Formal Definition

The discrete Fourier transform $\hat{x} \in \mathbb{C}^n$ of a vector $x \in \mathbb{C}^n$ is given by

$$\hat{x}_j = \sum_{i=1}^{n} \omega^{ij} \text{ for } \omega = e^{2\pi \mathbf{i}/n}$$

We say that $\hat{x}$ is *exactly* k-sparse if it has at most k nonzero coordinates, i.e., $|\text{supp}(x)| \leq k$. We say that $\hat{x}$ is *approximately* k-sparse if most of the energy is contained in the heaviest k coordinates, in particular

$$\text{Err}_k(x) := \min_{k\text{-sparse } \hat{y}} \|\hat{x} - \hat{y}\|_2$$

is small relative to $\|\hat{x}\|_2$. A sparse Fourier transform algorithm can access $x \in \mathbb{C}^n$ in arbitrary positions and outputs a vector $\hat{x}'$ such that

$$\|\hat{x} - \hat{x}'\|_2 \leq C \,\text{Err}_k(x) + \delta \|\hat{x}\|_2 \qquad (1)$$

for some approximation factor $C > 1$ and $\delta \ll 1$. An algorithm for the exactly sparse case would do this for $C = \infty$, while robust algorithms can achieve $C = O(1)$ or even $C = 1 + \varepsilon$. The algorithms we will discuss will feature a logarithmic dependence on $1/\delta$, so one typically sets $\delta = 1/\text{poly}(n)$, and for typical signals, the right-hand side of (1) will be dominated by the $C \,\text{Err}_k(x)$ term; we will assume this for the rest of the entry.

We would like to optimize both the sample complexity – the number of positions of x that are accessed by the algorithm – and the

running time. Optimizing sample complexity is important for applications such as spectrum sensing or MRIs, which do not have the input x in memory but must sample it at some expense.

We also allow the algorithm to be randomized and to fail with some small probability p. For simplicity we set p to a small constant; for any algorithm one can amplify this probability with a $O(\log \frac{1}{p})$ overhead in sample complexity and time. It is an open question whether the algorithms that achieve the best known time and sample complexities can be modified to avoid this overhead.

Related Work

The modern research on sparse Fourier transforms is closely related to work on sparse recovery from general linear measurements. In this problem, one would like to (approximately) recover an (approximately) sparse vector x from linear measurements Ax for some "measurement" matrix A with fewer rows than columns. The sparse Fourier transform is precisely this where A is a subset of rows of the inverse Fourier matrix.

Broadly speaking, there are two conceptual classes of algorithms and results for the general linear measurement setting. The first class, often called *compressed sensing* and first studied in [2, 3, 7], generally (1) involves independent random linear measurements,;(2) shows with high probability, the measurement matrix gives good recovery *for all* vectors x; (3) optimizes the sample complexity but not the running time, which is superlinear or polynomial in n; and (4) give algorithms that work for general classes of measurements and work for both random Gaussian and random Fourier matrices at the same time. These papers often refer to properties like the restricted isometry property that measurement matrices may have and use either convex optimization (e.g., L1 minimization or the LASSO) or iterative greedy methods (e.g., IHT or CoSaMP) to perform the recovery.

The second class, more often called *sparse recovery*, is largely an outgrowth of the streaming algorithms literature [4, 5]. These results generally (1) involve more structured linear measurements that use randomness and also have dependencies among the samples; (2) show *for each* vector x that, with high probability, the measurement matrix gives good recovery; (3) optimize both the sample complexity and the running time, so both may be sublinear in n; and (4) give algorithms that are closely connected to the measurement matrix and would not work for matrices with different structure. These papers often construct the matrix to emulate hash tables and use medians to perform robust recovery.

These statements are generalizations, and not every algorithm matches the trend in all four ways, but they hold more often than not. Our algorithm falls in the second class, which for Fourier measurements can achieve both better sample complexity and better running time than algorithms in the first class.

There's a much older collection of algorithms that can do sparse Fourier transforms in the exact setting when $|\text{supp}(\hat{x})| \leq k$. These include Prony's method from 1795, the matrix pencil method, and Berlekamp-Massey syndrome decoding. These can achieve the optimal sample complexity of $2k$ and recovery time poly(k) (down to $O(k^2 + k \log^c \log n)$ [8]). Additionally, they use a deterministic set of samples and work for all vectors x. However, it is not known how to make the techniques in these algorithms robust to approximately sparse signals, so they do not apply to the signals appearing in typical applications.

Noise-tolerant sparse Fourier transforms were first studied over the Boolean cube, also known as the Hadamard transform. In this setting, Goldreich and Levin [12, 18] showed how to get $O(k \log(n/k))$ samples and $O(k \log^c n)$ time, which is essentially optimal. Mansour [19] extended this to the $\mathbb{C}^n$ setting that we consider in this entry but with more than k^2 sample complexity. Over the next couple decades, a number of subsequent works, including [9, 10, 13, 14, 16], have improved our understanding of the $\mathbb{C}^n$ setting.

Key Results

At present, the two best sparse Fourier transform algorithms are [13], which is fastest at $O(k \log(n/k) \log n)$ time and sample complexity, and [14], which has nearly optimal $O(k \log n)$ sample complexity at the cost of $\tilde{O}(n)$ running time. These works build on [9, 10, 17].

We know that the optimal *nonadaptive* sample complexity – that is, among algorithms that choose the sample set Ω independently of the vector x – is $\Omega(k \log(n/k))$ [6], which matches [14] for $k < n^{0.99}$. One could imagine constructing an algorithm that uses adaptive samples, where one uses the first few samples to decide where to look in future samples. In the general sparse recovery setting, this adaptivity can lead to significant improvements [15], but we know that $\Omega(k \log(n/k)/\log\log n)$ Fourier samples remain necessary in the adaptive setting [13].

Algorithm Overview

At a high level, sparse recovery algorithms are built in three stages: one-sparse recovery, where we solve the problem for $k = 1$; partial k-sparse recovery, where we find and estimate *most* (say, 90 %) of the heavy coordinates or of the energy; and full k-sparse recovery, where we get a good approximation to the entire signal and achieve (1). Each stage uses the previous as (nearly) a black box. This architecture generally holds for the class of "sparse recovery" algorithms; in the sparse Fourier transform setting, the pieces change, but the architecture does not. We will go through each in turn.

One-Sparse Recovery

Let us consider the one-sparse setting for $C = O(1)$. We have access to $x_j = v\omega^{i^* j} + g_j$ for some "signal" $(v, i^*) \in \mathbb{C} \times [n]$ and "noise" $g \in \mathbb{C}^n$ with $\|g\|_2 \le c|v|\sqrt{n}$ for a sufficiently small constant c. To satisfy (1), we would like to find i^* exactly and find v to within $O(\|g\|/\sqrt{n})$.

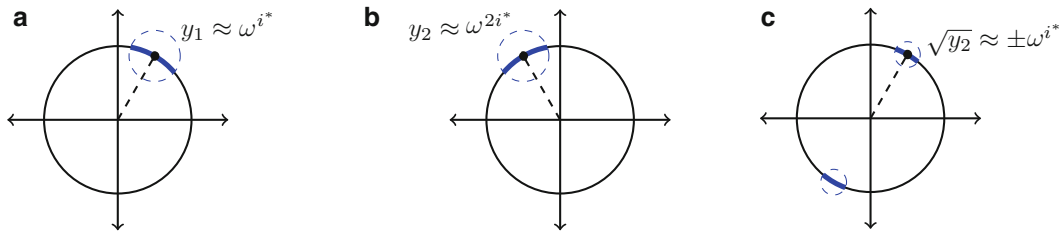
The tricky bit is to find i^*; once we know i^*, then $x_j \omega^{-i^* j}$ is a good estimator of v. In particular, for a random $j \in [n]$, we have $\mathbb{E}_j |x_j \omega^{-i^* j} - v|^2 = \|g\|^2/n$, so taking the median of several such estimates will have $O(\|g\|/\sqrt{n})$ error with large probability. So that just leaves us to find i^*.

As a first step, consider for a fixed $a \in [n]$ looking at the random variable

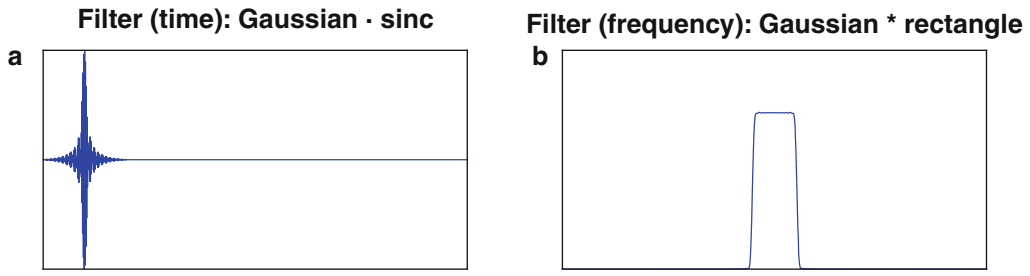
$$y_a := x_{a+j}/x_j \approx \omega^{i^* a}$$

as a distribution over random $j \in [n]$, where addition of indices is taken modulo n. This allows us to remove the influence of v and focus on i^*. We can show that $y_a - \omega^{i^* a} < O(c)$ with large (say, 3/4) probability. Suppose this were instead true with probability 1.

By knowing $\omega^{i^* a}$ to within $O(c)$, we know $i^* a \bmod n$ to with $\pm O(cn)$. For small enough c, this is within $\pm n/4$. Then we could look at y_1 to learn i^* to within $\pm n/4$, y_2 to refine the estimate to $\pm n/8$, and y_4 to refine to $\pm n/16$, until we identify i^* using $\log n$ different y_a. This is illustrated in Fig. 1.



Sparse Fourier Transform, Fig. 1 The first two steps of estimating i^* using y_1 and y_2. Using y_1 we can identify i^* to an $O(cn)$ size region. With y_2 we learn $2i^* \bmod n$ to within $O(cn)$, which tells us that i^* is within one of two antipodal regions of half the size. Based on y_1, we can throw out the spurious region and narrow our estimate of i^* (**a**) Error in y_1. (**b**) Error in y_2. (**c**) Set of ω^{i^*} consistent with y_2



Sparse Fourier Transform, Fig. 2 Filters used in [13]. (**a**) In time domain: $O(k \log n)$ sparse. (**b**) In frequency domain: width $O(n/k)$ rectangle

In reality y_a has a small constant chance of failure at each stage. One could fix this by taking $O(\log \log n)$ different samples of y_a at each stage and using the median, which would give an algorithm with $O(\log n \log \log n)$ sample complexity and time. An alternative, as used in [13], is to learn i^* in chunks of $O(\log \log n)$ bits at a time, which gets the optimal $O(\log n)$ sample complexity using $O(\log^{1.1} n)$ running time.

Partial *k*-Sparse Recovery

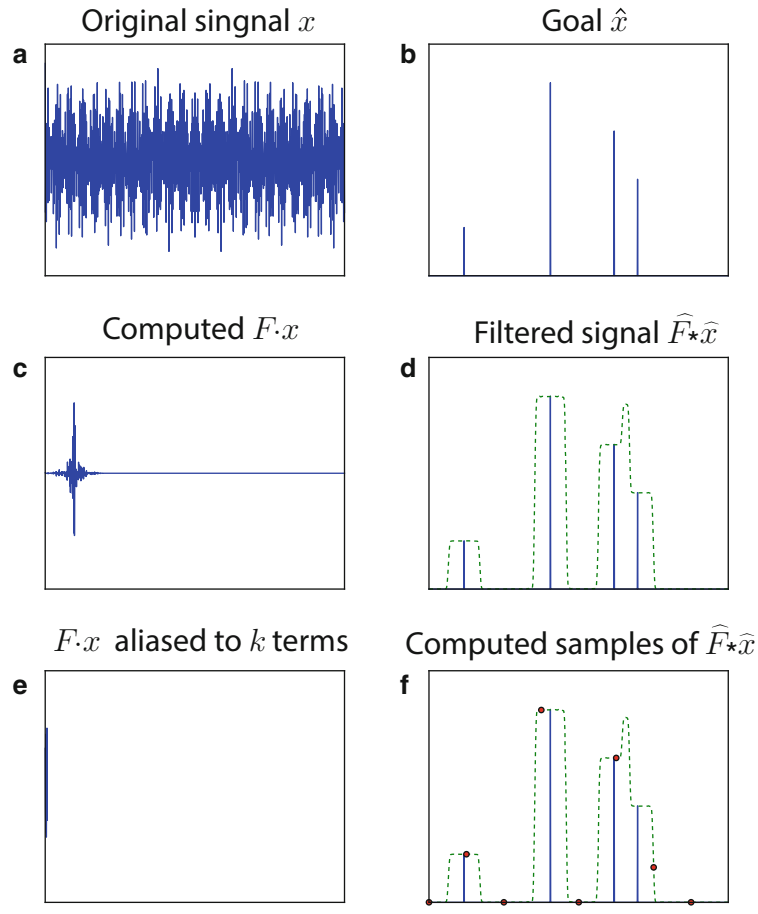
The goal of partial k-sparse recovery is to find *most* of the heavy coordinates of $\hat{x}$. The general idea is to "hash" the coordinates randomly into $B = O(k)$ bins in a way that lets us take measurements of the signal restricted to frequencies within each bin. By taking the measurements corresponding to the one-sparse recovery algorithm, we recover frequencies that are alone in their bin. This will happen with a large constant (say, 90 %) probability for each heavy frequency, so we recover *most* of the heavy frequencies well.

To see how this is done, we start with a deterministic way of hashing the frequencies into bins and then show how to randomize it. Hashing is based on filters that are sparse in both time and frequency domain. The filter F is designed to be as close as possible to a rectangular filter in frequency domain while still being sparse in frequency domain. Figure 2 demonstrates the filter used in [13], where F is a sinc function times a (truncated) Gaussian with support size $O(k \log n)$. In frequency domain, $\hat{F}$ approximates a rectangle of width $O(n/k)$, matching it up to a small transition region between the passband and the stopband and with $1/n^c$ error inside the passband and stopband.

Using these filters, Fig. 3 demonstrates a method for learning information about the signal. Given the signal x, we compute the $O(k \log n)$-size vector $F \cdot x$. We then "alias" it down to $B = O(k)$ elements – adding up terms $1, B+1, 2B+1, \ldots$ – and take the B-dimensional DFT. This lets us compute the red points in Fig. 3f in $O(k \log n + B \log B) = O(k \log n)$ time. The red points are B evenly spaced samples of $\hat{x} * \hat{F}$.

We can think of the ith red point in a different way. The ith red point is the sum of all the entries of $\hat{x} \cdot \text{shift}(\hat{F}, in/B)$, where $\text{shift}(\hat{F}, in/B)$ denotes shifting $\hat{F}$ to the right by in/B. This equals the zeroth time domain coefficient of the vector with Fourier coefficients given by $\hat{x} \cdot \text{shift}(\hat{F}, in/B)$. And if our algorithm looks not at $y_j = F_j x_j$ but $y_j^{(a)} = F_j x_{j+a}$ when computing the red points, then the ith red point will equal the ath time domain coefficient of the vector with Fourier coefficients given by $\hat{x} \cdot \text{shift}(\hat{F}, in/B)$. This lets us sample from the time domain representation of the vectors with Fourier coefficients given by $\hat{x} \cdot \text{shift}(\hat{F}, in/B)$ for $i \in [B]$. It takes $O(k \log n)$ time to get these samples, for $O(\log n)$ overhead (in time and samples) per "effective" sample.

Now, we simply choose our samples a from the distribution requested by the one-sparse recovery algorithm. In every bucket for which $\hat{x} \cdot \text{shift}(\hat{F}, in/B)$ is one-sparse, this procedure will let us recover the heavy frequency. Because the different shifts of $\hat{F}$ give B different buckets,



Sparse Fourier Transform, Fig. 3 The algorithm for hashing used in [13]. For simplicity, the illustrations do not include noise. (**a**) The signal in time domain. (**b**) Corresponds to this signal in frequency domain. (**c**) We observe $\boldsymbol{F} \cdot \boldsymbol{x}$ for a sparse $\boldsymbol{F}$. (**d**) Which has the dashed $\boldsymbol{n}$-dimensional DFT. (**e**) We alias from $\boldsymbol{O(k \log n)}$ terms to $\boldsymbol{O(k)}$. (**f**) And compute the $\boldsymbol{O(k)}$-dimensional DFT (*dots*)

if the frequencies were randomly distributed, this technique would get us partial sparse recovery. The one-sparse recovery algorithm only takes $O(\log(n/k))$ samples because each frequency is known to lie within an $n/B = O(n/k)$ size region; hence the overall method takes $O(k \log n \log(n/k))$ time and samples.

We would like the algorithm to work for arbitrary input signals, so we need a way of randomizing the frequencies. To do this, we further refine the algorithm to choose a random $\sigma, b \in [n]$ with σ relatively prime to n. Then we have the algorithm look at $y_j^{(a)} = F_j x_{\sigma(j+a)} \omega^{-\sigma j b}$. The effect of σ, b is to apply an hash function $j \to \sigma^{-1} j + b$ in frequency domain; this is approximately pairwise independent, so the frequencies become effectively randomly distributed. Each frequency then has a good chance of landing alone in its bucket, so we can recover most frequencies in $O(k \log(n/k) \log n)$ time and samples.

Full *k*-Sparse Recovery

Once we have partial k-sparse recovery, one can naively achieve full k-sparse recovery by repeating the algorithm $O(\log k)$ times. Since each heavy frequency is recovered with 90 % probability in each stage, the median of all the estimations will recover all the heavy frequencies – and in fact achieve (1) – with high probability. This method is simple but loses a $\log k$ factor in running time and sample complexity, which more intricate techniques can avoid.

One such technique, used in [13] and based off [11], is to use smaller and smaller k in successive iterations. Once we have performed partial sparse recovery on $\hat{x}$ to get $\hat{x}^{(1)}$ that contains

90 % of the heavy hitters, we can then perform sparse recovery on the residual $\hat{x} - \hat{x}^{(1)}$. The residual is then roughly $k/10$-sparse, so we run a partial $k/10$-sparse recovery algorithm in the second stage that is much faster than in the first stage. Similar geometric decay happens in later stages, so the total time spent will be dominated by the first stage. This gives $O(k \log(n/k) \log n)$ time and sample complexity for the problem.

Recommended Reading

1. Ailon N (2014) An n\log n lower bound for Fourier transform computation in the well conditioned model. CoRR. abs/1403.1307, http://arxiv.org/abs/1403.1307
2. Candes E, Tao T (2006) Near optimal signal recovery from random projections: universal encoding strategies. IEEE Trans Info Theory
3. Candes E, Romberg J, Tao T (2006) Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. IEEE Trans Info Theory 52:489–509
4. Charikar M, Chen K, Farach-Colton M (2002) Finding frequent items in data streams. In: ICALP
5. Cormode G, Muthukrishnan S (2004) Improved data stream summaries: the count-min sketch and its applications. In: LATIN
6. Do Ba K, Indyk P, Price E, Woodruff D (2010) Lower bounds for sparse recovery. In: SODA
7. Donoho D (2006) Compressed sensing. IEEE Trans Info Theory 52(4):1289–1306
8. Ghazi B, Hassanieh H, Indyk P, Katabi D, Price E, Shi L (2013) Sample-optimal average-case sparse Fourier transform in two dimensions. In: Allerton
9. Gilbert A, Guha S, Indyk P, Muthukrishnan M, Strauss M (2002) Near-optimal sparse Fourier representations via sampling. In: STOC
10. Gilbert A, Muthukrishnan M, Strauss M (2005) Improved time bounds for near-optimal space Fourier representations. In: SPIE conference on wavelets
11. Gilbert AC, Li Y, Porat E, Strauss MJ (2010) Approximate sparse recovery: optimizing time and measurements. In: STOC, pp 475–484
12. Goldreich O, Levin L (1989) A hard-corepredicate for allone-way functions. In: STOC, pp 25–32
13. Hassanieh H, Indyk P, Katabi D, Price E (2012) Nearly optimal sparse Fourier transform. In: STOC
14. Indyk P, Kapralov M (2014) Sample-optimal Fourier sampling in any constant dimension. In: 2014 IEEE 55th annual symposium on foundations of computer science (FOCS). IEEE, pp 514–523
15. Indyk P, Price E, Woodruff D (2011) On the power of adaptivity in sparse recovery. In: FOCS
16. Iwen MA (2010) Combinatorial sublinear-time Fourier algorithms. Found Comput Math 10:303–338
17. Kushilevitz E, Mansour Y (1991) Learning decision trees using the Fourier spectrum. In: STOC
18. Levin L (1993) Randomness and non-determinism. J Symb Logic 58(3):1102–1103
19. Mansour Y (1992) Randomized interpolation and approximation of sparse polynomials. In: ICALP
20. Morgenstern J (1973) Note on a lower bound on the linear complexity of the fast Fourier transform. J ACM (JACM) 20(2):305–306

Sparse Graph Spanners

Michael Elkin
Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Keywords

$(1+\epsilon, \beta)$-spanners; Almost additive spanners

Years and Authors of Summarized Original Work

2004; Elkin, Peleg

Problem Definition

For a pair of numbers α, β, $\alpha \geq 1$, $\beta \geq 0$, a subgraph $G' = (V, H)$ of an unweighted undirected graph $G = (V, E)$, $H \subseteq E$, is an (α, β)-spanner of G if for every pair of vertices $u, w \in V$, $\text{dist}_{G'}(u, w) \leq \alpha \cdot \text{dist}_G(u, w) + \beta$, where $\text{dist}_G(u, w)$ stands for the distance between u and w in G. It is desirable to show that for every n-vertex graph there exists a sparse (α, β)-spanner with as small values of α and β as possible. The problem is to determine asymptotic tradeoffs between α and β on one hand, and the sparsity of the spanner on the other.

Key Results

The main result of Elkin and Peleg [8] establishes the existence and efficient constructibility of

$(1+\epsilon,\beta)$-spanners of size $O(\beta n^{1+1/\kappa})$ for every n-vertex graph G, where $\beta=\beta(\epsilon,\kappa)$ is constant whenever κ and ϵ are. The specific dependence of β on κ and ϵ is $\beta(\kappa,\epsilon)=\kappa^{\log\log\kappa-\log\epsilon}$.

An important ingredient of the construction of [8] is a partition of the graph G into regions of small diameter in such a way that the supergraph induced by these regions is sparse. The study of such partitions was initiated by Awerbuch [3], that used them for network synchronization. Peleg and Schäffer [10] were the first to employ such partitions for constructing spanners. Specifically, they constructed $(O(\kappa),1)$-spanners with $O(n^{1+1/\kappa})$ edges. Althofer et al. [2] provided an alternative proof of the result of Peleg and Schäffer that uses an elegant greedy argument. This argument also enabled Althofer et al. to extend the result to weighted graphs, to improve the constant hidden by the O-notation in the result of Peleg and Schäffer, and to obtain related results for planar graphs.

Applications

Efficient algorithms for computing sparse $(1+\epsilon,\beta)$-spanners were devised in [7] and [13]. The algorithm of [7] was used in [7, 9, 12] for computing almost shortest paths in centralized, distributed, streaming, and dynamic centralized models of computations. The basic approach used in these results is to construct a sparse spanner, and then to compute exact shortest paths on the constructed spanner. The sparsity of the latter guarantees that the computation of shortest paths in the spanner is far more efficient than in the original graph.

Open Problems

The main open question is whether it is possible to achieve similar results with $\epsilon=0$. More formally, the question is: Is it true that for any $\kappa\geq 1$ and any n-vertex graph G there exists $(1,\beta(\kappa))$-spanner of G with $O(n^{1+1/\kappa})$ edges? This question was answered in affirmitive for κ equal to 2, 5/2, and 3 [1, 4-6, 8]. Some lower bounds were recently proved by Woodruff [14].

A less challenging problem is to improve the dependence of β on ϵ and κ. Some progress in this direction was achieved by Thorup and Zwick [13], and very recently by Pettie [11].

Cross-References

▸ Synchronizers, Spanners

Recommended Reading

1. Aingworth D, Chekuri D, Indyk P, Motwani R (1999) Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM J Comput 28(4):1167–1181
2. Althofer I, Das G, Dobkin DP, Joseph D, Soares J (1993) On sparse spanners of weighted graphs. Discret Comput Geom 9:81–100
3. Awerbuch B (1985) Complexity of network synchronization. J ACM 4:804–823
4. Baswana S, Kavitha T, Mehlhorn K, Pettie S (2010) Additive spanners and (alpha, beta)-spanners. ACM Trans Algorithms 7:Article 1
5. Chechik S (2013) New additive spanners. In: Proceedings 24th annual ACM-SIAM symposium on discrete algorithms, New Orleans, Jan 2013, pp 498–512.
6. Dor D, Halperin S, Zwick U (2000) All pairs almost shortest paths. SIAM J Comput 29:1740–1759
7. Elkin M (2005) Computing almost shortest paths. Trans Algorithm 1(2):283–323
8. Elkin M, Peleg D (2004) $(1+\epsilon,\beta)$-spanner constructions for general graphs. SIAM J Comput 33(3):608–631
9. Elkin M, Zhang J (2006) Efficient algorithms for constructing $(1+\epsilon,\beta)$-spanners in the distributed and streaming models. Distrib Comput 18(5):375–385
10. Peleg D, Schäffer A (1989) Graph spanners. J Graph Theory 13:99–116
11. Pettie S (2009) Low-distortion spanners. ACM Trans Algorithms 6:Article 1
12. Roditty L, Zwick U (2004) Dynamic approximate all-pairs shortest paths in undirected graphs. In: Proceedings of symposium on foundations of computer science, Rome, Oct 2004, pp 499–508
13. Thorup M, Zwick U (2006) Spanners and emulators with sublinear distance errors. In: Proceedings of symposium on discrete algorithms, Miami, Jan 2006, pp 802–809

14. Woodruff D (2006) Lower bounds for additive spanners, emulators, and more. In: Proceedings of symposium on foundations of computer science, Berckeley, Oct 2006, pp 389–398

Sparsest Cut

Shuchi Chawla
Department of Computer Science, University of Wisconsin–Madison, Madison, WI, USA

Keywords

Minimum ratio cut

Years and Authors of Summarized Original Work

2004; Arora, Rao, Vazirani

Problem Definition

In the Sparsest Cut problem, informally, the goal is to partition a given graph into two or more large pieces while removing as few edges as possible. Graph partitioning problems such as this one occupy a central place in the theory of network flow, geometric embeddings, and Markov chains, and form a crucial component of divide-and-conquer approaches in applications such as packet routing, VLSI layout, and clustering.

Formally, given a graph $G = (V, E)$, the *sparsity* or *edge expansion* of a non-empty set $S \subset V$, $|S| \leq \frac{1}{2}|V|$, is defined as follows:

$$\alpha(S) = \frac{|E(S, V \setminus S)|}{|S|} .$$

The sparsity of the graph, $\alpha(G)$, is then defined as follows:

$$\alpha(G) = \min_{S \subset V, |S| \leq \frac{1}{2}|V|} \alpha(S) .$$

The goal in the Sparsest Cut problem is to find a subset $S \subset V$ with the minimum sparsity, and to determine the sparsity of the graph.

The first approximation algorithm for the Sparsest Cut problem was developed by Leighton and Rao in 1988 [13]. Employing a linear programming relaxation of the problem, they obtained an $O(\log n)$ approximation, where n is the size of the input graph. Subsequently Arora, Rao and Vazirani [4] obtained an improvement over Leighton and Rao's algorithm using a semi-definite programming relaxation, approximating the problem to within an $O(\sqrt{\log n})$ factor.

In addition to the Sparsest Cut problem, Arora et al. also consider the closely related Balanced Separator problem. A partition $(S, V \setminus S)$ of the graph G is called a c-balanced separator for $0 < c \leq \frac{1}{2}$, if both S and $V \setminus S$ have at least $c|V|$ vertices. The goal in the Balanced Separator problem is to find a c-balanced partition with the minimum sparsity. This sparsity is denoted $\alpha_c(G)$.

Key Results

Arora et al. provide an $O(\sqrt{\log n})$ *pseudo-approximation* to the balanced-separator problem using semi-definite programming. In particular, given a constant $c \in (0, \frac{1}{2}]$, they produce a separator with balance c' that is slightly worse than c (that is, $c' < c$), but sparsity within an $O(\sqrt{\log n})$ factor of the sparsity of the optimal c-balanced separator.

Theorem 1 *Given a graph $G = (V, E)$, let $\alpha_c(G)$ be the minimum edge expansion of a c-balanced separator in this graph. Then for every fixed constant $a < 1$, there exists a polynomial-time algorithm for finding a c'-balanced separator in G, with $c' \geq ac$, that has edge expansion at most $O(\sqrt{\log n}\alpha_c(G))$.*

Extending this theorem to include unbalanced partitions, Arora et al. obtain the following:

Theorem 2 *Let $G = (V, E)$ be a graph with sparsity $\alpha(G)$. Then there exists a polynomial-time*

algorithm for finding a partition $(S, V \setminus S)$, *with* $S \subset V$, $S \neq \emptyset$, *having sparsity at most* $O(\sqrt{\log n}\alpha(G))$.

An important contribution of Arora et al. is a new geometric characterization of vectors in n-dimensional space endowed with the squared-Euclidean metric. This result is of independent significance and has lead to or inspired improved approximation factors for several other partitioning problems (see, for example, [1, 5, 6, 7, 11]).

Informally, the result says that if a set of points in n-dimensional space is randomly projected on to a line, a good separator on the line is, with high probability, a good separator (in terms of squared-Euclidean distance) in the original high-dimensional space. Separation on the line is related to separation in the original space via the following definition of stretch.

Definition 1 (Def. 4 in [4]) Let $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n$ be a set of n points in $\mathcal{R}^n$, equipped with the squared-Euclidean metric $\mathrm{d}(x, y) = \|x - y\|_2^2$. The set of points is said to be (t, γ, β) *-stretched at scale* ℓ, if for at least a γ fraction of all the n-dimensional unit vectors u, there is a partial matching $M_u = \{(x_i, y_i)\}_i$ among these points, with $|M_u| \geq \beta n$, such that for all $(x, y) \in M_u$, $\mathrm{d}(x, y) \leq \ell^2$ and $\langle u, \vec{x} - \vec{y} \rangle \geq t\ell/\sqrt{n}$. Here $\langle \cdot, \cdot \rangle$ denotes the dot product of two vectors.

Theorem 3 *For any* $\gamma, \beta > 0$, *there is a constant* $C = C(\gamma, \beta)$ *such that if* $t > C \log^{1/3} n$, *then no set of n points in* $\mathcal{R}^n$ *can be* (t, γ, β)*-stretched for any scale* ℓ.

In addition to the SDP-rounding algorithm, Arora et al. provide an alternate algorithm for finding approximate sparsest cuts, using the notion of expander flows. This result leads to fast (quadratic time) implementations of their approximation algorithm [3].

Applications

One of the main applications of balanced separators is in improving the performance of divide and conquer algorithms for a variety of optimization problems.

One example is the Minimum Cut Linear Arrangement problem. In this problem, the goal is to order the vertices of a given n vertex graph G from 1 through n in such a way that the capacity of the largest of the cuts $(\{1, 2, \cdots, i\}, \{i + 1, \cdots, n\})$, $i \in [1, n]$, is minimized. Given a ρ-approximation to the balanced separator problem, the following divide and conquer algorithm gives an $O(\rho \log n)$-approximation to the Minimum Cut Linear Arrangement problem: find a balanced separator in the graph, then recursively order the two parts, and concatenate the orderings. The approximation follows by noting that if the graph has a balanced separator with expansion $\alpha_c(G)$, only $O(\rho n \alpha_n(G))$ edges are cut at every level, and given that a balanced separator is found at every step, the number of levels of recursion is at most $O(\log n)$.

Similar approaches can be used for problems such as VLSI layout and Gaussian elimination. (See the survey by Shmoys [14] for more details on these topics.)

The Sparsest Cut problem is also closely related to the problem of embedding squared-Euclidean metrics into the Manhattan (ℓ_1) metric with low distortion. In particular, the integrality gap of Arora et al.'s semi-definite programming relaxation for Sparsest Cut (generalized to include weights on vertices and capacities on edges) is exactly equal to the worst-case distortion for embedding a squared-Euclidean metric into the Manhattan metric. Using the technology introduced by Arora et al., improved embeddings from the squared-Euclidean metric into the Manhattan metric have been obtained [5, 7].

Open Problems

Hardness of approximation results for the Sparsest Cut problem are fairly weak. Recently Chuzhoy and Khanna [9] showed that this problem is APX-hard, that is, there exists a constant $\epsilon > 0$, such that a $(1 + \epsilon)$-approximation

algorithm for Sparsest Cut would imply P = NP. It is conjectured that the weighted version of the problem is NP-hard to approximate better than $O((\log \log n)^c)$ for some constant c, but this is only known to hold true assuming a version of the so-called Unique Games conjecture [8, 12]. On the other hand, the semi-definite programming relaxation of Arora et al. is known to have an integrality gap of $\Omega(\log \log n)$ even in the unweighted case [10]. Proving an unconditional super-constant hardness result for weighted or unweighted Sparsest Cut, or obtaining $o(\sqrt{\log n})$-approximations for these problems remain open.

The directed version of the Sparset Cut problem has also been studied, and is known to be hard to approximate within a $2^{\Omega(\log^{1-\epsilon} n)}$ factor [9]. On the other hand, the best approximation known for this problem only achieves a polynomial factor of approximation–a factor of $O(n^{11/23} \log^{O(1)} n)$ due to Aggarwal, Alon and Charikar [2].

Recommended Reading

1. Agarwal A, Charikar M, Makarychev K, Makarychev Y (2005) Proceedings of the 37th ACM symposium on theory of computing (STOC), Baltimore, May 2005, pp 573–581
2. Aggarwal A, Alon N, Charikar M (2007) Improved approximations for directed cut problems. In: Proceedings of the 39th ACM symposium on theory of computing (STOC), San Diego, June 2007, pp 671–680
3. Arora S, Hazan E, Kale S (2004) Proceedings of the 45th IEEE symposium on foundations of computer science (FOCS), Rome, 17–19 Oct 2004, pp 238–247
4. Arora S, Rao S, Vazirani U (2004) Expander flows, geometric embeddings, and graph partitionings. In: Proceedings of the 36th ACM symposium on theory of computing (STOC), Chicago, June 2004, pp 222–231
5. Arora S, Lee J, Naor A (2005) Euclidean distortion and the sparsest cut. In: Proceedings of the 37th ACM Symposium on Theory of Computing (STOC), Baltimore, May 2005, pp 553–562
6. Arora S, Chlamtac E, Charikar M (2006) New approximation guarantees for chromatic number. In: Proceedings of the 38th ACM symposium on theory of computing (STOC), Seattle, May 2006, pp 215–224
7. Chawla S, Gupta A, Räcke H (2005) Embeddings of negative-type metrics and an improved approximation to generalized sparsest cut. In: Proceedings of the ACM-SIAM symposium on discrete algorithms (SODA), Vancouver, Jan 2005, pp 102–111
8. Chawla S, Krauthgamer R, Kumar R, Rabani Y, Sivakumar D (2005) On the hardness of approximating sparsest cut and multicut. In: Proceedings of the 20th IEEE conference on computational complexity (CCC), San Jose, June 2005, pp 144–153
9. Chuzhoy J, Khanna S (2007) Polynomial flow-cut gaps and hardness of directed cut problems. In: Proceedings of the 39th ACM symposium on theory of computing (STOC), San Diego, June 2007, pp 179–188
10. Devanur N, Khot S, Saket R, Vishnoi N (2006) Integrality gaps for sparsest cut and minimum linear arrangement problems. In: Proceedings of the 38th ACM symposium on theory of computing (STOC), Seattle, May 2006, pp 537–546
11. Feige U, Hajiaghayi M, Lee J (2005) Improved approximation algorithms for minimum-weight vertex separators. In: Proceedings of the 37th ACM symposium on theory of computing (STOC), Baltimore, May 2005, pp 563–572
12. Khot S, Vishnoi N (2005) Proceedings of the 46th IEEE symposium on foundations of computer science (FOCS), Pittsburgh, Oct 2005, pp 53–62
13. Leighton FT, Rao SB (1988) An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In: Proceedings of the 29th IEEE symposium on foundations of computer science (FOCS), White Plains, Oct 1988, pp 422–431
14. Shmoys DB (1997) Cut problems and their application to divide-and-conquer. In: Hochbaum DS (ed) Approximation algorithms for NP-hard problems. PWS Publishing, Boston, pp 192–235

Speed Scaling

Kirk Pruhs
Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

Keywords

Frequency scaling; Speed scaling; Voltage scaling

Years and Authors of Summarized Original Work

1995; Yao, Demers, Shenker

Problem Definition

Speed scaling is a power management technique in modern processor that allows the processor to run at different speeds. There is a power function $P(s)$ that specifies the power, which is energy used per unit of time, as a function of the speed. In CMOS-based processors, the cube-root rule states that $P(s) \approx s^3$. This is usually generalized to assume that $P(s) = s^\alpha$ form some constant α. The goals of power management are to reduce temperature and/or to save energy. Energy is power integrated over time. Theoretical investigations to date have assumed that there is a fixed ambient temperature and that the processor cools according to Newton's law, that is, the rate of cooling is proportional to the temperature difference between the processor and the environment.

In the resulting scheduling problems, the scheduler must not only have a job-selection policy to determine the job to run at each time, but also a speed scaling policy to determine the speed at which to run that job. The resulting problems are generally dual objective optimization problems. One objective is some quality of service measure for the schedule, and the other objective is temperature or energy.

We will consider problems where jobs arrive at the processor over time. Each job i has a release time r_i when it arrives at the processor, and a work requirement w_i. A job i run at speed s takes w_i/s units of time to complete.

Key Results

Yao et al. [5] initiated the theoretical algorithmic investigation of speed scaling problems. Yao et al. [5] assumed that each job i had a deadline d_i, and that the quality of service measure was deadline feasibility (each job completes by its deadline). Yao et al. [5] gives a greedy algorithm YDS to find the minimum energy feasible schedule. The job selection policy for YDS is to run the job with the earliest deadline. To understand the speed scaling policy for YDS, define the intensity of a time interval to be the work that must be completed in this time interval divided by the length of the time interval. YDS then finds the maximum intensity interval, runs the jobs that must be run in this interval at constant speed, eliminates these jobs and this time interval from the instance, and proceeds recursively. Yao et al. [5] gives two online algorithms: OA and AVR. In OA the speed scaling policy is the speed that YDS would run at, given the current state and given that no more jobs will be released in the future. In AVR, the rate at which each job is completed is constant between the time that a job is released and the deadline for that job. Yao et al. [5] showed that AVR is $2^{\alpha-1}\alpha^\alpha$-competitive with respect to energy.

The results in [5] were extended in [2]. Bansal et al. [2] showed that OA is α^α-competitive with respect to energy. Bansal et al. [2] proposed another online algorithm, BKP. BKP runs at the speed of the maximum intensity interval containing the current time, taking into account only the work that has been released by the current time. They show that the competitiveness of BKP with respect to energy is at most $2(\alpha/(\alpha-1))^\alpha \mathrm{e}^\alpha$. They also show that BKP is e-competitive with respect to the maximum speed.

Bansal et al. [2] initiated the theoretical algorithmic investigation of speed scaling to manage temperature. Bansal et al. [2] showed that the deadline feasible schedule that minimizes maximum temperature can in principle be computed in polynomial time. Bansal et al. [2] showed that the competitiveness of BKP with respect to maximum temperature is at most $2^{\alpha+1}\mathrm{e}^\alpha(6(\alpha/(\alpha-1))^\alpha + 1)$.

Pruhs et al. [4] initiated the theoretical algorithmic investigation into speed scaling when the quality-of-service objective is average/total flow time. The flow time of a job is the delay from when a job is released until it is completed. Pruhs et al. [4] give a rather complicated polynomial-time algorithm to find the optimal flow time schedule for unit work jobs, given

a bound on the energy available. It is easy to see that no $O(1)$-competitive algorithm exists for this problem.

Albers and Fujiwara [1] introduce the objective of minimizing a linear combination of energy used and total flow time. This has a natural interpretation if one imagines the user specifying how much energy he is willing to use to increase the flow time of a job by a unit amount. Albers and Fujiwara [1] give an $O(1)$-competitive online algorithm for the case of unit work jobs. Bansal et al. [3] improves upon this result and gives a 4-competitive online algorithm. The speed scaling policies of the online algorithms in [1] and [3] essentially run as power equal to the number of unfinished jobs (in each case modified in a particular way to facilitate analysis of the algorithm). Bansal et al. [3] extend these results to apply to jobs with arbitrary work, and even arbitrary weight. The speed scaling policy is essentially to run at power equal to the weight of the unfinished work. The expression for the resulting competitive ratio is a bit complicated but is approximately 8 when the cube-root rule holds.

The analysis of the online algorithms in [2] and [3] heavily relied on amortized local competitiveness. An online algorithm is locally competitive for a particular objective if for all times the rate of increase of that objective for the online algorithm, plus the rate of change of some potential function, is at most the competitive ratio times the rate of increase of the objective in any other schedule.

Applications

None

Open Problems

The outstanding open problem is probably to determine if there is an efficient algorithm to compute the optimal flow time schedule given a fixed energy bound.

Recommended Reading

1. Albers S, Fujiwara H (2006) Energy-efficient algorithms for flow time minimization. In: STACS. Lecture notes in computer science, vol 3884. Springer, Berlin, pp 621–633
2. Bansal N, Kimbrel T, Pruhs K (2007) Speed scaling to manage energy and temperature. J ACM 54(1)
3. Bansal N, Pruhs K, Stein C (2007) Speed scaling for weighted flow. In: ACM/SIAM symposium on discrete algorithms
4. Pruhs K, Uthaisombut P, Woeginger G (2004) Getting the best response for your ERG. In: Scandanavian workshop on algorithms and theory
5. Yao F, Demers A, Shenker S (1995) A scheduling model for reduced CPU energy. In: IEEE symposium on foundations of computer science, p 374

Sphere Packing Problem

Danny Z. Chen
Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA

Keywords

Ball packing; Disk packing

Years and Authors of Summarized Original Work

2001; Chen, Hu, Huang, Li, Xu

Problem Definition

The sphere packing problem seeks to pack spheres into a given geometric domain. The problem is an instance of geometric packing. Geometric packing is a venerable topic in mathematics. Various versions of geometric packing problems have been studied, depending on the shapes of packing domains, the types of packing objects, the position restrictions on the objects, the optimization criteria, the

dimensions, etc. It also arises in numerous applied areas. The sphere packing problem under consideration here finds applications in radiation cancer treatment using Gamma Knife systems. Unfortunately, even very restricted versions of geometric packing problems (e.g., regular-shaped objects and domains in lower dimensional spaces) have been proved to be NP-hard. For example, for *congruent packing* (i.e., packing copies of the same object), it is known that the 2-D cases of packing fixed-sized congruent squares or disks in a simple polygon are NP-hard [7]. Baur and Fekete [2] considered a closely related *dispersion problem* of packing k congruent disks in a polygon of n vertices such that the radius of the disks is maximized; they proved that the dispersion problem cannot be approximated arbitrarily well in polynomial time unless P = NP, and gave a $\frac{2}{3}$-approximation algorithm for the L_∞ disk case with a time bound of $O(n^{38})$.

Chen et al. [4] proposed a practically efficient heuristic scheme, called *pack-and-shake*, for the **congruent sphere packing** problem, based on computational geometry techniques. The problem is defined as follows.

The Congruent Sphere Packing Problem

Given a d-D polyhedral region $R(d = 2, 3)$ of n vertices and a value $r > 0$, find a packing SP of R using spheres of radius r, such that (i) each sphere is contained in R, (ii) no two distinct spheres intersect each other in their interior, and (iii) the ratio (called the packing density) of the covered volume in R by SP over the total volume of R is maximized.

In the above problem, one can view the spheres as "solid" objects. The region R is also called the *domain* or *container*. Without loss of generality, let $r = 1$.

Much work on congruent sphere packing studied the case of packing spheres into an unbounded domain or even the whole space [5]. There are also results on packing congruent spheres into a bounded region. Hochbaum and Maass [8] presented a unified and powerful *shifting technique* for designing pseudo-polynomial time approximation schemes for packing congruent squares into a rectilinear polygon. But, the high time complexities associated with the resulting algorithms restrict their applicability in practice. Another approach is to formulate a packing problem as a non-linear optimization problem, and resort to an available optimization software to generate packings; however, this approach works well only for small problem sizes and regular-shaped domains.

To reduce the running time yet achieve a dense packing, a common idea is to consider objects that form a certain lattice or double-lattice. A number of results were given on lattice packing of congruent objects in the whole (especially high dimensional) space [5]. For a bounded rectangular 2-D domain, Milenkovic [10] adopted a method that first finds the densest translational lattice packing for a set of polygonal objects in the whole plane, and then uses some heuristics to extract the actual bounded packing.

Key Results

The *pack-and-shake* scheme of Chen et al. [4] for packing congruent spheres in an irregular-shaped 2-D or 3-D bounded domain R consists of three phases. In the first phase, the d-D domain R is partitioned into a set of convex subregions (called *cells*). The resulting set of cells defines a dual graph G_D, such that each vertex v of G_D corresponds to a cell $C(v)$ and an edge connects two vertices if and only if their corresponding cells share a $(d-1)$-D face. In the second phase, the algorithm repeats the following *trimming and packing* process until $G_D = \emptyset$: Remove the lowest degree vertex v from G_D and pack the cell $C(v)$. In the third phase, a *shake* procedure is applied to globally adjust the packing to obtain a denser one.

The objective of the trimming and packing procedure is that after each cell is packed, the remaining "packable" subdomain R' of R is always kept as a connected region. The rationale for maintaining the connectivity of R' is as follows. To pack spheres in a bounded domain R, two

typical approaches have been used: (a) packing spheres layer by layer going from the boundary of R towards its interior [9], and (b) packing spheres starting from the "center" of R, such as its medial axis, towards its boundary [3, 13, 14]. Due to the shape irregularity of R, both approaches may fragment the remaining "packable" subdomain R' into more and more disconnected regions; however, at the end of packing each such region, a small "unpackable" area may eventually remain that allows no further packing. It could fit more spheres if the "packable" subdomain R' is lumped together instead of being divided into fragments, which is what the trimming and packing procedure aims to achieve.

Due to the packing of its adjacent cells that have been done by the trimming and packing procedure, the boundary of a cell $C(v)$ that is to be packed may consist of both line segments and arcs (from packed spheres). Hence, a key problem is to pack spheres in a cell bounded by curves of low degrees. Chen et al.'s algorithms [4] for packing each cell are based on certain lattice structures and allow the cell to both translate and rotate. Their algorithms have fairly low time bounds. In certain cases, they even run in nearly linear time.

An interesting feature of the cell packings generated by the trimming and packing procedure is that the resulted spheres cluster together in the middle of the cells of the domain R, leaving some small unpackable areas scattered along the boundary of R. The "shake" procedure in [4] thus seeks to collect these small areas together by "pushing" the spheres towards the boundary of R, in the hope of obtaining some "packable" region in the middle of R.

The approach in [4] is to first obtain a densest lattice unit sphere packing $LSP(C)$ for each cell C of R, and then use a "shake" procedure to globally adjust the resulting packing of R to generate a denser packing SP in R. Suppose the plane P is already packed by infinitely many unit spheres whose center points form a lattice (e.g., the hexagonal lattice). To obtain a densest packing $LSP(C)$ for a cell C from the lattice packing of the plane P, a position and orientation of C on P need to be computed such that C contains the maximum number of spheres from the lattice packing of P. There are two types of algorithms in [4] for computing an optimal placement of C on P: translational algorithms that allow C to be translated only, and translational/rotational algorithms that allow C to be both translated and rotated.

Let $n = |C|$, the number of bounding curves of C, and m be the number of spheres along the boundary of C in a sought optimal packing of C.

Theorem 1 *Given a polygonal region C bounded by n algebraic curves of constant degrees, a densest lattice unit sphere packing of C based only on translational motion can be computed in $O(N \log N + K)$ time, where $N = f(n, m)$ is a function of n and m, and K is the number of intersections between N planar algebraic curves of constant degrees that are derived from the packing instance.*

Note: In the worst case, $N = f(n, m) = n \times m$. But in practice, N may be much smaller. The N planar algebraic curves in Theorem 1 form a structure called *arrangement*. Since all these curves are of a constant degree, any two such curves can intersect each other at most a constant number of times. In the worst case, the number K of intersections between the N algebraic curves, which is also the *size* of the arrangement, is $O(N^2)$. The arrangement of these curves can be computed by the algorithms [1, 6] in $O(N \log N + K)$ time.

Theorem 2 *Given a polygonal region C bounded by n algebraic curves of constant degrees, a densest lattice unit sphere packing of C based on both translational and rotational motions can be computed in $O(T(n) + (N + K') \log N)$ time, where $N = f(n, m)$ is a function of n and m, K' is the size of the arrangement of N*

pseudo-plane surfaces in 3-D that are derived from the packing instance, and T(n) is the time for solving $O(n^2)$ quadratic optimization problem instances associated with the packing instance.

In Theorem 2, $K' = O(N^3)$ in the worst case. In practice, K' can be much smaller.

The results on 2-D sphere packing in [4] can be extended to d-D for any constant integer $d \geq 3$, so long as a good d-D lattice packing of the d-D space is available.

Applications

Recent interest in the considered congruent sphere packing problem was motivated by medical applications in Gamma Knife radiosurgery [4, 11, 12]. Radiosurgery is a minimally invasive surgical procedure that uses radiation to destroy tumors inside human body while sparing the normal tissues. The Gamma Knife is a radiosurgical system that consists of 201 Cobalt-60 sources [3, 14]; the gamma-rays from these sources are all focused on a common center point, thus creating a spherical volume of radiation field. The Gamma Knife treatment normally applies high radiation dose. In this setting, overlapping spheres may result in overdose regions (called *hot spots*) in the target treatment domain, while a low packing density may cause underdose regions (called *cold spots*) and a non-uniform dose distribution. Hence, one may view the spheres used in Gamma Knife packing as "solid" spheres. Therefore, a key geometric problem in Gamma Knife treatment planning is to fit multiple spheres into a 3-D irregular-shaped tumor [3, 13, 14]. The total treatment time crucially depends on the number of spheres used. Subject to a given packing density, the minimum number of spheres used in the packing (i.e., treatment) is desired. The Gamma Knife currently produces spheres of four different radii (4, 8, 14, and 18 mm), and hence the Gamma Knife sphere packing is in general not congruent. In practice, a commonly used approach is to pack larger spheres first, and then fit smaller spheres into the remaining subdomains, in the hope of reducing the total number of spheres involved and thus shortening the treatment time. Therefore, congruent sphere packing can be used as a key subroutine for such a common approach.

Open Problems

An open problem is to analyze the quality bounds of the resulting packing for the algorithms in [4]; such packing quality bounds are currently not yet known. Another open problem is to reduce the running time of the packing algorithms in [4], since these algorithms, especially for sphere packing problems in higher dimensions, are still very time-consuming. In general, it is highly desirable to develop efficient sphere packing algorithms in d-D ($d \geq 2$) with guaranteed good packing quality.

Experimental Results

Some experimental results of the 2-D pack-and-shake sphere packing algorithms were given in [4]. The planar hexagonal lattice was used for the lattice packing. On packings whose sizes are in the hundreds, the C++ programs of the algorithms in [4] based only on translational motion run very fast (a few minutes), while those of the algorithms based on both translation and rotation take much longer time (hours), reflecting their respective theoretical time bounds, as expected. On the other hand, the packing quality of the translation-and-rotation based algorithms is a little better than the translation based algorithms. The packing densities of all the algorithms in the experiments are well above 70 % and some are even close to or above 80 %. Comparing with the nonconvex programming methods, the packing algorithms in [4] seemed to run faster based on the experiments.

Cross-References

▸ Local Approximation of Covering and Packing Problems

Recommended Reading

1. Amato NM, Goodrich MT, Ramos EA (2000) Computing the arrangement of curve segments: divide-and-conquer algorithms via sampling. In: Proceedings of the 11th annual ACM-SIAM symposium on discrete algorithms, pp 705–706
2. Baur C, Fekete SP (2001) Approximation of geometric dispersion problems. Algorithmica 30(3):451–470
3. Bourland JD, Wu QR (1996) Use of shape for automated, optimized 3D radiosurgical treatment planning. In: Proceedings of the SPIE international symposium on medical imaging, pp 553–558
4. Chen DZ, Hu X, Huang Y, Li Y, Xu J (2001) Algorithms for congruent sphere packing and applications. In: Proceedings of the 17th annual ACM symposium on computational geometry, pp 212–221
5. Conway JH, Sloane NJA (1988) Sphere packings, lattices and groups. Springer, New York
6. Edelsbrunner H, Guibas LJ, Pach J, Pollack R, Seidel R, Sharir M (1992) Arrangements of curves in the plane: topology, combinatorics, and algorithms. Theor Comput Sci 92:319–336
7. Fowler RJ, Paterson MS, Tanimoto SL (1981) Optimal packing and covering in the plane are NP-complete. Inf Process Lett 12(3):133–137
8. Hochbaum DS, Maass W (1985) Approximation schemes for covering and packing problems in image processing and VLSI. J ACM 32(1):130–136
9. Li XY, Teng SH, Üngör A (2000) Biting: advancing front meets sphere packing. Int J Numer Methods Eng 49(1–2):61–81
10. Milenkovic VJ (2000) Densest translational lattice packing of nonconvex polygons. In: Proceedings of the 16th ACM annual symposium on computational geometry, pp 280–289
11. Shepard DM, Ferris MC, Ove R, Ma L (2000) Inverse treatment planning for Gamma Knife radiosurgery. Med Phys 27(12):2748–2756
12. Sutou A, Dai Y (2002) Global optimization approach to unequal sphere packing problems in 3D. J Optim Theory Appl 114(3):671–694
13. Wang J (1999) Medial axis and optimal locations for min-max sphere packing. J Comb Optim 3:453–463
14. Wu QR (1996) Treatment planning optimization for Gamma unit radiosurgery. Ph.D. thesis, The Mayo Graduate School

Split Decomposition via Graph-Labelled Trees

Christophe Paul
CNRS, Laboratoire d'Informatique Robotique et Microélectronique de Montpellier, Université Montpellier 2, Montpellier, France

Keywords

Circle graphs; Distance hereditary graphs; LexBFS; Parity graphs; Permutation graphs; Split decomposition

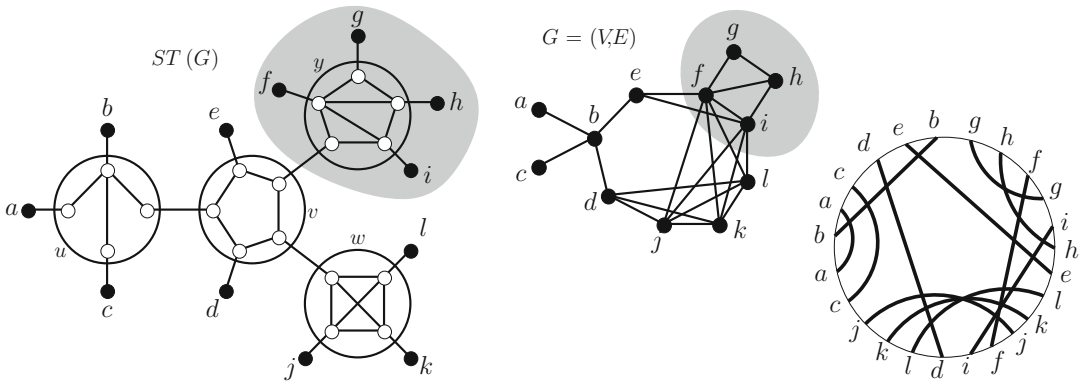
Years and Authors of Summarized Original Work

2012; Gioan, Paul
2014; Gioan, Paul, Tedder, Corneil

Problem Definition

Introduced by Cunningham and Edmonds [11], the *split decomposition*, also known as the *join (or 1-join) decomposition*, ranges among the classical graph decomposition schemes. Given a graph $G = (V, E)$, a bipartition (A, B) of the vertex set V (with $|A| \geq 2$ and $|B| \geq 2$) is a *split* if there are subsets $A' \subseteq A$ and $B' \subseteq B$, called *frontiers*, such that there is an edge between a vertex $u \in A$ and $v \in B$ if and only if $u \in A'$ and $v \in B'$ (see Fig. 1). A graph is *prime* if it does not contain any split. Observe that an induced cycle of length at least 5 is a prime graph. A graph is *degenerate* if every bipartition (A, B) with $|A| \geq 2$ and $|B| \geq 2$ is a split. It can be shown that a degenerate graphs are either *cliques* or *stars*. The split decomposition consists in recursively decompose a graph into a set of disjoint graphs $\{G_1, \ldots G_k\}$, called *split components*, each of which is either prime or degenerate. There are two cases:

1. If G is prime or degenerate, then return the set $\{G\}$;



Split Decomposition via Graph-Labelled Trees, Fig. 1 A circle graph G with a chord diagram on the *right* and its split decomposition tree $ST(G)$ on the *left*. The nodes v and y are prime nodes, whereas u is a star node and w a clique node. The bipartition $(\{f, g, h, i\}, V \setminus \{f, g, h, i\})$ forms a split of G and corresponds to a tree edge of $ST(G)$. The frontiers are $\{f, i\}$ on one side and $\{e, j, k, l\}$ on the other. Observe that $(\{k, l\}, V \setminus \{k, l\})$ is also a split but which is not represented by the tree edge between nodes Y and Z in $ST(G)$. Because G is not a prime graph, it can be represented with several chord diagram. For example, exchanging the chord of y with the chord of z yields an alternative chord diagram

2. If G is neither prime nor degenerate, it contains a split (A, B), with frontiers A' and B'. The split components of G is then the union of the split components of the graphs $G[A] + (a, A')$ and $G[B] + (b, B')$, where a and b are new vertices, called *markers*.

Observe that the split decomposition process naturally defines a decomposition tree whose nodes represent the split components. This decomposition tree can be represented by a *graph-labeled tree* (GLT) (see [16,18]) defined as a pair $(T, \mathcal{F})$, where T is a tree and $\mathcal{F}$ a set of graphs, such that each node u of T is *labeled* by the graph $G(u) \in \mathcal{F}$, and there exists a bijection ρ_u between the edges of T incident to u and the vertices of $G(u)$, called *marker vertices*. We say that two leaves ℓ_a and ℓ_b of T are *accessible* if for every pair of consecutive tree edges uv and vw on the path from ℓ_a and ℓ_b in T, $\rho_v(uv)$ and $\rho_v(vw)$ are adjacent in $G(v)$. From a GLT $(T, \mathcal{F})$, we define an *accessibility graph* $\mathcal{G}(T, \mathcal{F})$ whose vertex set is the leaf set of T and two vertices a and b are adjacent if the corresponding leaves ℓ_a and ℓ_b are accessible. It is easy to observe that every tree edge e of a GLT $(T, \mathcal{F})$ defines a split (A, B) of $\mathcal{G}(T, \mathcal{F})$ where A and B respectively contain the vertices corresponding to the leaves of the two connected components of $T-e$. Cunningham and Edmonds [11] formalized the family of splits as an example of *partite family of bipartitions* thereby implying that every graph admits a canonical split decomposition tree (see Fig. 1). In terms of GLTs, this translates as follows:

Theorem 1 ([11, 16, 18]) *Let G be a connected graph. There exists a unique GLT $(T, \mathcal{F})$ whose labels are either prime or degenerate, having a minimal number of nodes and such that $G = \mathcal{G}(T, \mathcal{F})$. This GLT is called the* split tree *of G and denoted $ST(G)$.*

The problem we are interested in is to efficiently compute the split tree $ST(G)$ of a graph $G = (V, E)$. The first polynomial-time algorithm was and runs in time $O(nm)$, where $n = |V|$ and $m = |E|$. Ma and Spinrad [23] later developed an $O(n^2)$ algorithm. Finally Dahlhaus [12] designed the first linear-time algorithm which was recently revisited by Charbit et al. [5].

Key Results

As mentioned above, the split tree of a graph can be computed in linear time. The algorithm we

describe here is nearly optimal, that is, runs in time $O(n+m)\cdot\alpha(n+m)$, where α is the inverse Ackermann function. The fact that this algorithm incrementally builds the split tree is responsible of the small additional complexity cost. More precisely, updating the tree structure of the GLT representing the split tree relies on the union-find data structure [15]. But having an incremental split decomposition algorithm allows an extension of the algorithm, within the same time complexity, to the circle graph recognition [17], a problem for which computing the split decomposition is a corner step. But so far, a subquadratic time complexity cannot be reached using the previous linear (or quadratic) split decomposition algorithms.

Theorem 2 ([18]) *The split tree $ST(G)$ of a graph $G = (V, E)$, with $|V| = n$ and $|E| = m$, can be built incrementally according to an LBFS ordering in time $O(n + m) \cdot \alpha(n + m)$, where α is the inverse Ackermann function.*

It is important to observe that to reach the expected complexity, the algorithm inserts the vertices according to a *LexBFS ordering* [25]. These orderings, resulting from a *lexicographic breadth-first search*, appear in a number of recognition algorithms, such as *chordal graphs* [25], *comparability graphs* [20], *interval graphs* [22], and *cographs* [3]. The idea is that structural properties can be shown on the last vertex visited by a LexBFS. For example, in chordal graphs the last vertex is simplicial; in comparability graphs it is a source of some transitive orientation. LexBFS, introduced in [25], works as follows: it numbers the vertices decreasingly from $n = |V|$ down to 1; initially every vertex receives an empty label; then iteratively, an arbitrary unnumbered vertex x with lexicographically largest label is selected and numbered i, and i is appended to the label of every unnumbered neighbor of x. On the graph of Fig. 1, $\sigma = b, a, e, d, c, f, i, j, k, l, h, g$ is a LexBFS ordering.

Applications

Many graph classes can be characterized by means of the split decomposition. Below, we review the most important of these classes. Finally, we discuss the links between split decomposition and other decomposition approaches.

Graph Classes

Distance Hereditary Graphs

The family of graphs for which the split tree does not contain any prime node is called *totally decomposable* (or *totally separable*). This terminology follows from the observation that for every subgraph of size at least 4, every nontrivial bipartition of the vertex set forms a split. A graph G is *distance hereditary* [1] if for every induced connected subgraph H of G and every pair of vertices x and y of H, the distance between x and y is the same in H and G. It turns out that a graph G is totally decomposable if and only if it is *distance hereditary* [1]. In other words, a graph G is distance hereditary if and only if every node of $ST(G)$ is either a star or a clique node. The first linear-time recognition algorithm of distance hereditary graphs, due to [21], relies on a breadth-first search characterization (see also [13]). More recently, a linear-time algorithm has been designed to update the split tree of a distance hereditary graph under vertex and edge insertion, leading to an alternative (vertex incremental) linear-time recognition algorithm for distance hereditary graphs.

Theorem 3 ([16]) *Let $ST(G)$ be the split tree of a distance hereditary graph $G = (V, E)$, $S \subseteq V$ be a subset of vertices of G and $e = (x, y) \notin E$ be a non-edge of G. Then:*

- *In $O(1)$-time, we can compute $ST(G + e)$ where $G+e = (V, E\cup\{e\})$ if $G+e$ is distance hereditary;*
- *In $O(|S|)$-time, we can compute $ST(G + (x, S)$ where $G + (x, S) = (V \cup \{x\}, E \cup \{(x, y) \mid y \in S\}$ if $G + (x, S)$ is distance hereditary.*

Subclasses of Totally Decomposable Graphs

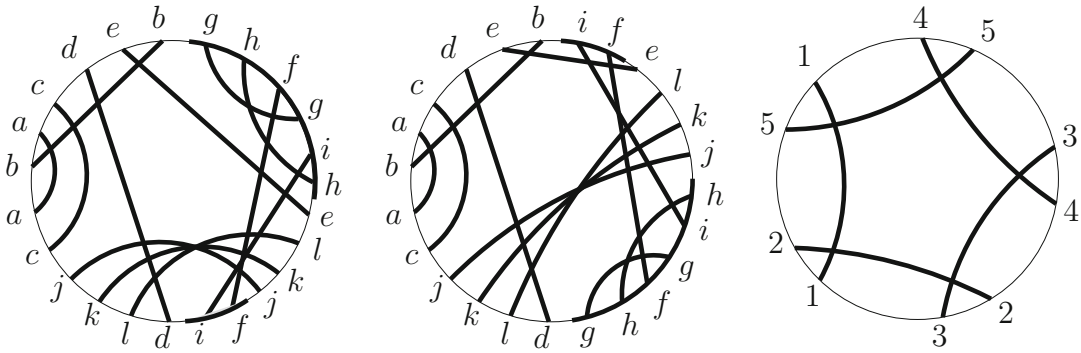
A GLT is called *clique-star tree* if its nodes are labeled either with cliques or stars. As a consequence of the discussion of the previous paragraph, distance hereditary graphs are the graphs corresponding the clique-star trees. Imposing any constraint on a clique-star tree thereby immediately defines a subclass of distance hereditary graphs. It turns out that many important graph subclasses of distance hereditary graphs can be characterized with the split decomposition.

The *cographs*, also known as *complement-reducible graphs* [8] or P_4*-free graphs*, are probably the most studied subclass of distance hereditary graphs. Cographs are also known as the class of graphs totally decomposable with respect to the modular decomposition [19], and their combinatorial structure is captured by the so-called cotree. As noticed in [16], it is easy to observe that a graph G is a cograph if and only if its split tree $ST(G)$ is a clique-star tree that can be rooted either at a node or at a tree edge such that every star node is "oriented" toward that root (that is the marker vertex corresponding the center of every star node is oriented toward the root).

The class of *ptolemaic graphs* or *3-leaf power* are also interesting. The class of ptolemaic graphs is defined as the intersection of distance hereditary graphs and chordal graphs. Chordal graphs are the graphs without induced chordless cycles of length four or more. It follows that a graph G is ptolemaic if and only if $ST(G)$ is a clique-star tree such that for every pair of star nodes u and v, not both extremities of the path from u to v in $ST(G)$ are attached to the center marker vertex of u and v (otherwise this would generate a chordless 4-cycle). As a subclass of chordal graph, 3-leaf powers inherit the restrictions of ptolemaic graphs on the split tree with the additive constraint that no clique node lies on the path between two star nodes (see [16] for details).

Circle Graphs

The split decomposition plays an important role in the context of *circle graphs* defined as intersection graphs of a set of chords in a circle. The main reason is that a graph G is a circle graph if and only if every split component of G is a circle graph. In other words, as clique and stars are circle graphs, G is a circle graph if and only if the prime nodes of $ST(G)$ are labeled with circle graphs. Observe that this characterization shows that distance hereditary graphs form a subclass of circle graphs. By the way the first quadratic time circle graph recognition algorithm was obtained by computing the split decomposition of the input graph and reducing the problem to the recognition of prime circle graphs [23, 26]. The key property is that a prime circle graph has a unique (up to mirror) *chord diagram* [2, 14] (see Fig. 2).



Split Decomposition via Graph-Labelled Trees, Fig. 2 On the *left*, two distinct chord diagrams of the graph G depicted in Fig. 1 results from symmetric insertion of the chords representing the vertices $\{f, g, h, i\}$ (remind that $(\{f, g, h, i\}, V \setminus \{f, g, h, i\})$ form a split). On the *right*, the chord diagram on $\{1, 2, 3, 4, 5\}$ is the unique (up to rotation and mirror) chord diagram of the 5-cycle, which is a prime graph

The linear-time split decomposition algorithm [12], proposed in the mid-1990s, did not lead to a linear-time circle graph recognition algorithm. For almost two decades, the quadratic time complexity [27] remained the best known complexity. The quadratic barrier has been broken using the almost linear-time split decomposition algorithm of [17]. The key ingredient was to insert the vertices according to a LexBFS ordering. Indeed, in the unique chord diagram of a prime circle graph G, the neighborhood of the last vertex x of a LexBFS ordering satisfies a sort of consecutiveness property. More precisely, the chord diagram of G contains a set of consecutive chord extremities starting and ending with the extremities of x's chord and containing one and only one chord extremity per neighbor of x and no chord extremity of non-neighbors of x. This property is used to incrementally build the split tree of a circle graph using chord diagrams to represent prime nodes. It is worth to observe that the split tree of a circle graph G together with the chord diagrams of each of its prime nodes provides a canonical (linear space) representation of the set of (exponentially many) chord diagrams of G.

Theorem 4 ([17]) *Let $G = (V, E)$ be a graph such that $|V| = n$ and $|E| = m$. There exists a $O(n + m) \cdot \alpha(n + m)$-time algorithm, where α is the inverse Ackermann function, deciding whether G is a circle graph. Moreover, if G is a circle graph, the algorithm outputs a split-tree representation G from which any chord diagram of G can be extracted in linear time.*

Perfect Graphs

The recent proof [6] of the famous conjecture of Berge on perfect graphs states that a graph is *perfect* if and only if it does not contain an odd cycle of length at least 5 nor its complement as induced subgraph. It is easy to observe that a graph is perfect if and only if its prime components are perfect graphs. The split decomposition does not formally appear in the structural decomposition theorem of perfect graphs [6, 28] as it is subsumed by the so-called balanced skew partition. In the context of perfect graphs, *parity graphs* [4] form a nice example of class of graphs simply characterized through their split decomposition. A graph is a parity graph if for every pair x, y of vertices, the length of every chordless path between x and y is of the same parity. This constraint can be translated into a condition on odd cycles or into a condition on their split tree. Indeed it can be proved that a graph is a parity graph if and only if its prime nodes are labeled with bipartite graphs [7].

Related Graph Decompositions

Modular Decomposition

The split decomposition is often introduced as a generalization of the *modular decomposition* (also known as homogeneous decomposition) [19]. A *module* in a graph $G = (V, E)$ is a subset M of vertices such that every vertex not in M is either fully adjacent or fully nonadjacent to the vertices of M. Clearly, if M is a module of size at least 2, then $(M, V \setminus M)$ defines a split. Indeed the split decomposition is sometimes used to further decompose graphs that are primes with respect to the modular decomposition.

Width Parameters

Rank-width [24] and *clique-width* [10] are two important width parameters both sharing some connections with the split decomposition. As the rank-width of a graph is small if its clique-width is small and vice versa, we only briefly describe the former parameter. A rank-decomposition of a graph G is defined as a ternary tree whose leaves are in one-to-one correspondence with the vertices of G. It follows that every internal tree edge defines a bipartition, say (A, B) of the vertices of G. The rank-width of a bipartition (A, B) is defined as the rank of the incidence matrix between A and B, and the width of a rank-decomposition is the maximum width over its bipartitions. The rank-width of a graph G is then the minimum width over its rank-decompositions. Observe that the every split is a rank-width 1 bipartition. It follows that the rank-width of a graph is the maximum rank-width of its prime components. As a consequence rank-width one graphs are exactly distance hereditary graphs. To conclude, let us

mention that computing the split decomposition of a graph is a key step in the polynomial-time recognition algorithm of clique-width three graphs [9].

Recommended Reading

1. Bandelt H-J, Mulder HM (1986) Distance hereditary graphs. J Comb Theory Ser B 41:182–208
2. Bouchet A (1987) Reducing prime graphs and recognizing circle graphs. Combinatorica 7:243–254
3. Bretscher A, Corneil D, Habib M, Paul C (2008) A simple linear time lexbfs cograph recognition algorithm. SIAM J Discret Math 22(4):1277–1296
4. Burlet M, Uhry JP (1984) Parity graphs. Ann Discret Math 21:253–277
5. Charbit P, de Montgolfier F, Raffinot M (2012) Linear time split decomposition revisited. SIAM J Discret Math 26(2):499–514
6. Chudnovsky M, Robertson N, Seymour P, Thomas R (2006) The strong perfect graph theorem. Ann Math 161:51–229
7. Cicerone S, Di Stefano G (1999) On the extension of bipartite to parity graphs. Discret Appl Math 95:181–195
8. Corneil D, Lerchs H, Stewart-Burlingham LK (1981) Complement reducible graphs. Discret Appl Math 3(1):163–174
9. Corneil D, Habib M, Lanlignel JM, Reed B, Rotics U (2012) Polynomial-time recognition of clique-width $\leqslant 3$ graphs. Discret Appl Math 160(6):834–865
10. Courcelle B, Engelfriet J, Rozenberg G (1993) Handle rewriting hypergraph grammars. J Comput Syst Sci 46:218–270
11. Cunningham WH, Edmonds J (1980) A combinatorial decomposition theory. Can J Math 32(3):734–765
12. Dahlhaus E (1994) Efficient parallel and linear time sequential split decomposition (extended abstract). In: Foundations of software technology and theoretical computer science – FSTTCS, Madras. Volume 880 of lecture notes in computer science, pp 171–180
13. Damiand G, Habib M, Paul C (2001) A simple paradigm for graph recognition: application to cographs and distance hereditary graphs. Theor Comput Sci 263:99–111
14. Gabor CP, Hsu WL, Suppovit KJ (1989) Recognizing circle graphs in polynomial time. J ACM 36:435–473
15. Gabow H, Tarjan R (1983) A linear-time algorithm for a special case of disjoint set union. In: Annual ACM symposium on theory of computing (STOC), Boston, pp 246–251
16. Gioan E, Paul C (2012) Split decomposition and graph-labelled trees: characterizations and fully dynamic algorithms for totally decomposable graphs. Discret Appl Math 160(6):708–733
17. Gioan E, Paul C, Tedder M, Corneil D (2013) Circle graph recognition in time $O(n+m)\alpha(n+m)$. Algorithmica 69(4): 759–788 (2014)
18. Gioan E, Paul C, Tedder M, Corneil D (2013) Practical split-decomposition via graph-labelled trees. Algorithmica 69(4): 789–843 (2014)
19. Habib M, Paul C (2010) A survey on algorithmic aspects of modular decomposition. Comput Sci Rev 4:41–59
20. Habib M, McConnell RM, Paul C, Viennot L (2000) Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. Theor Comput Sci 234:59–84
21. Hammer P, Maffray F (1990) Completely separable graphs. Discret Appl Math 27:85–99
22. Korte N, Möhring R (1989) An incremental linear-time algorithm for recongizing interval graphs. SIAM J Comput 18(1):68–81
23. Ma T-H, Spinrad J (1994) An $O(n^2)$ algorithm for undirected split decomposition. J Algorithms 16:145–160
24. Oum SI (2005) Graphs of bounded rank-width. PhD thesis, Princeton University
25. Rose DJ, Tarjan RE, Lueker GS (1976) Algorithmic aspects of vertex elimination on graphs. SIAM J Comput 5(2):266–283
26. Spinrad J (1989) Prime testing for the split decomposition of a graph. SIAM J Discret Math 2(4):590–599
27. Spinrad J (1994) Recognition of circle graphs. J Algorithms 16:264–282
28. Trotignon N (2013) Perfect graphs: a survey. Technical report 1301.5149, arxiv

Squares and Repetitions

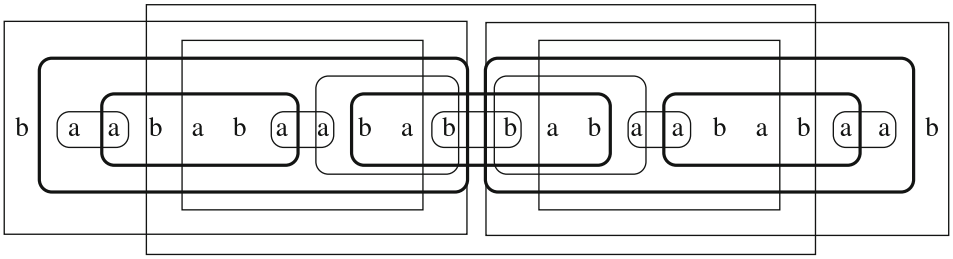
Maxime Crochemore[1,2,4] and Wojciech Rytter[3]
[1]Department of Computer Science, King's College London, London, UK
[2]Laboratory of Computer Science, University of Paris-East, Paris, France
[3]Institute of Informatics, Warsaw University, Warsaw, Poland
[4]Université de Marne-la-Vallée, Champs-sur-Marne, France

Keywords

Powers; Runs; Tandem repeats

Years and Authors of Summarized Original Work

1999; Kolpakov, Kucherov



Squares and Repetitions, Fig. 1 The structure of $RUNS(x)$ where $x = \texttt{baababaababbabaababaab} = \texttt{b}z^2(z^R)^2\texttt{b}$, for $z = \texttt{aabab}$. The operation $\cdot^R$ is reversing the string

Problem Definition

Periodicities and repetitions in strings have been extensively studied and are important both in theory and practice (combinatorics of words, pattern-matching, computational biology). The words of the type ww and www, where w is a nonempty primitive (not of the form u^k for an integer $k > 1$) word, are called squares and cubes, respectively. They are well-investigated objects in combinatorics on words [16] and in string-matching with small memory [5].

A string w is said to be periodic iff $period(w) \leq |w|/2$, where $period(w)$ is the smallest positive integer p for which $w[i] = w[i + p]$ whenever both sides of the equality are defined. In particular each square and cube is periodic.

A repetition in a string $x = x_1 x_2 \ldots x_n$ is an interval $[i \,..\, j] \subseteq [1 \,..\, n]$ for which the associated factor $x[i \,..\, j]$ is periodic. It is an occurrence of a periodic word $x[i \,..\, j]$, also called a positioned repetition. A word can be associated with several repetitions, see Fig. 1.

Initially people investigated mostly positioned squares, but their number is $\Omega(n \log n)$ [2], hence algorithms computing all of them cannot run in linear time, due to the potential size of the output. The optimal algorithms reporting all positioned squares or just a single square were designed in [1, 2, 3, 19]. Unlike this, it is known that only $O(n)$ (un-positioned) squares can appear in a string of length n [8].

The concept of maximal repetitions, called runs (equivalent terminology) in [14], has been introduced to represent all repetitions in a succinct manner. The crucial property of runs is that there are only $O(n)$ runs in a word of length n [15, 21].

A *run* in a string x is an interval $[i \,..\, j]$ such that both the associated string $x[i \,..\, j]$ has period $p \leq (j - i + 1)/2$, and the periodicity cannot be extended to the right nor to the left: $x[i - 1] \neq x[x + p - 1]$ and $x[j - p + 1] \neq x[j + 1]$ when the elements are defined. The set of runs of x is denoted by $RUNS(x)$. An example is displayed in Fig. 1.

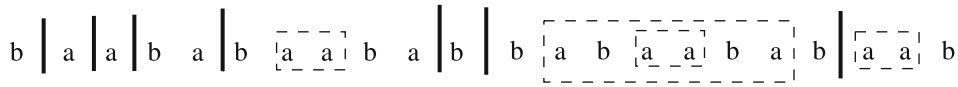
Key Results

The main results concern fast algorithms for computing positioned squares and runs, as well as combinatorial estimation on the number of corresponding objects.

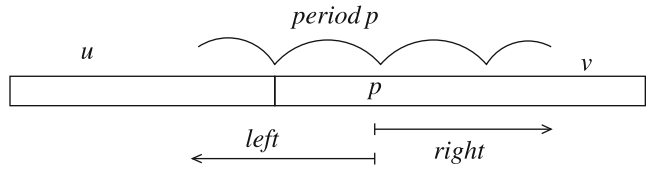
Theorem 1 (Crochemore [1], Apostolico-Preparata [2], Main-Lorentz [19]) *There exists an $O(n \log n)$ worst-case time algorithm for computing all the occurrences of squares in a string of length n.*

Techniques used to design the algorithms are based on partitioning, suffix trees, and naming segments. A similar result has been obtained by Franek, Smyth, and Tang using suffix arrays [11]. The key component in the next algorithm is the function described in the following lemma.

Lemma 2 (Main-Lorentz [19]) *Given two square-free strings u and v, reporting if uv contains a square centered in u can be done in worst-case time $O(|u|)$.*



Squares and Repetitions, Fig. 2 The f-factorization of the example string $x = \texttt{baababaababbabaababaab}$ and the set of its internal runs; all other runs overlap factorization points



Squares and Repetitions, Fig. 3 If an overlapping run with period p starts in u, ends in v, and its part in v is of size at least p then it is easily detectable by computing continuations of the periodicity p in two directions: left and right

Using suffix trees or suffix automata together with the function derived from the lemma, the following fact has been shown.

Theorem 3 (Crochemore [3], Main-Lorentz [19]) *Testing the square-freeness of a string of length n can be done in worst-case time $O(n \log a)$, where a is the size of the alphabet of the string.*

As a consequence of the algorithms and of the estimation on the number of squares, the most important result related to repetitions can be formulated as follows.

Theorem 4 (Kolpakov-Kucherov [15], Rytter [21], Crochemore-Ilie [4])

(1) *All runs in a string can be computed in linear time (on a fixed-size alphabet).*
(2) *The number of all runs is linear in the length of the string.*

The point (2) is very intricate, it is of purely combinatorial nature and has nothing to do with the algorithm. We sketch shortly the basic components in the constructive proof of the point (1). The main idea is to use, as for the previous theorem, the f-factorization (see [3]): a string x is decomposed into factors $u_1, u_2, \ldots, u_k$, where u_i is the longest segment which appears before (possibly with overlap) or is a single letter if the segment is empty.

The runs which fit in a single factor are called internal runs, other runs are called here overlapping runs. There are three crucial facts:

- all overlapping runs can be computed in linear time,
- each internal run is a copy of an earlier overlapping run,
- the f-factorization can be computed in linear time (on a fixed-size alphabet) if we have the suffix tree or suffix automaton of the string. Figure 2 shows f-factorization and internal runs of an example string.

It follows easily from the definition of the f-factorization that if a run overlaps two (consecutive) factors u_{k-1} and u_k then its size is at most twice the total size of these two s factors.

Figure 3 shows the basic idea for computing runs that overlap $u\,v$ in time $O(|u| + |v|)$. Using similar tables as in the Morris–Pratt algorithm (border and prefix tables), see [6], we can test the continuation of a period p from position p in v to the left and to the right. The corresponding tables can be constructed in linear time in a preprocessing phase. After computing all overlapping runs the internal runs can be copied from their earlier occurrences by processing the string from left to right.

Another interesting result concerning periodicities is the following lemma and its fairly immediate corollary.

Lemma 5 (Three Prefix Squares, Crochemore-Rytter [5]) *If u, v, and w are three primitive words satisfying: $|u| < |v| < |w|$, uu is a prefix of vv, and vv is a prefix of ww, then $|u| + |v| \leq |w|$*

Corollary 1 *Any nonempty string x possesses less than $\log_\Phi |y|$ prefixes that are squares.*

In the configuration of the lemma, a second consequence is that uu is a prefix of w. Therefore, a position in a string x cannot be the largest position of more than two squares, which yields the next corollary. A simple direct proof of it is by Ilie [13], see also [17].

Corollary 2 (Fraenkel and Simpson [8]) *Any string x contains at most $2|x|$ (different) squares, that is: card$\{u \mid u$ primitive and u^2 factor of $y\} \leq 2|x|$.*

The structure of all squares and of un-positioned runs has been also computed within the same time complexities as above in [18] and [12].

Applications

Detecting repetitions in strings is an important element of several questions: pattern matching, text compression, and computational biology to quote a few. Pattern-matching algorithms have to cope with repetitions to be efficient as these are likely to slow down the process; the large family of dictionary-based text compression methods use a weaker notion of repeats (like the software gzip); repetitions in genomes, called satellites, are intensively studied because, for example, some over-repeated short segments are related to genetic diseases; some satellites are also used in forensic crime investigations.

Open Problems

The most intriguing question remains the asymptotically tight bound for the maximum number $\rho(n)$ of runs in a string of size n. The first proof (by painful induction) was quite difficult and has not produced any *concrete* constant coefficient in the $O(n)$ notation. This subject has been studied in [9, 10, 22, 23]. The best-known lower bound of approximately $0.927\,n$ is from [10]. The exact number of runs has been considered for special strings: *Fibonacci words* and (more generally) *Sturmian words* [7, 14, 20]. It is proved in a structural and intricate manner in the full version of [21] that $\rho(n) \leq 3.44\,n$, by introducing a *sparse-neighbors technique*. The neighbors are runs for which both the distance between their starting positions is small and the difference between their periods is also proportionally small (according to some fixed coefficient of proportionality). The occurrences of neighbors satisfy certain *sparsity* properties which imply the linear upper bound. Several variations for the definitions of neighbors and sparsity are possible. Considering runs having close centers the bound has been lowered to $1.6\,n$ in [4].

As a conclusion, we believe that the following fact is valid.

Conjecture: A string of length n contains less than n runs, i.e., $|\mathrm{RUNS}|(n) < n$.

Cross-References

Elements of the present entry are of main importance for run-length compression as well as for ▸ Multidimensional Compressed Pattern Matching. They are also related to the ▸ Approximate Tandem Repeats entries because "tandem repeat" is a synonym of repetition and "power."

Recommended Reading

1. Apostolico A, Preparata FP (1983) Optimal off-line detection of repetitions in a string. Theor Comput Sci 22(3):297–315
2. Crochemore M (1981) An optimal algorithm for computing the repetitions in a word. Inf Process Lett 12(5):244–250

3. Crochemore M (1986) Transducers and repetitions. Theor Comput Sci 45(1):63–86
4. Crochemore M, Ilie L (2007) Analysis of maximal repetitions in strings. J Comput Sci
5. Crochemore M, Rytter W (1995) Squares, cubes, and time-space efficient string searching. Algorithmica 13(5):405–425
6. Crochemore M, Rytter W (2003) Jewels of stringology. World Scientific, Singapore
7. Franek F, Karaman A, Smyth WF (2000) Repetitions in Sturmian strings. Theor Comput Sci 249(2):289–303
8. Fraenkel AS, Simpson RJ (1998) How many squares can a string contain? J Comb Theory Ser A 82:112–120
9. Fraenkel AS, Simpson RJ (1999) The exact number of squares in fibonacci words. Theor Comput Sci 218(1):95–106
10. Franek F, Simpson RJ, Smyth WF (2003) The maximum number of runs in a string. In: Proceedings of the 14-th Australian workshop on combinatorial algorithms. Curtin University Press, Perth, pp 26–35
11. Franek F, Smyth WF, Tang Y (2003) Computing all repeats using suffix arrays. J Autom Lang Comb 8(4):579–591
12. Gusfield D, Stoye J (2004) Linear time algorithms for finding and representing all the tandem repeats in a string. J Comput Syst Sci 69(4):525–546
13. Ilie L (2005) A simple proof that a word of length n has at most 2n distinct squares. J Comb Theory Ser A 112(1):163–164
14. Iliopoulos C, Moore D, Smyth WF (1997) A characterization of the squares in a Fibonacci string. Theor Comput Sci 172:281–291
15. Kolpakov R, Kucherov G (1999) Finding maximal repetitions in a word in linear time. In: Proceedings of the 40th symposium on foundations of computer science. IEEE Computer Society, Los Alamitos, pp 596–604
16. Lothaire M (ed) (2002) Algebraic combinatorics on words. Cambridge University Press, Cambridge
17. Lothaire M (ed) (2005) Applied combinatorics on words. Cambridge University Press, Cambridge
18. Main MG (1989) Detecting leftmost maximal periodicities. Discret Appl Math 25:145–153
19. Main MG, Lorentz RJ (1984) An O(n log n) algorithm for finding all repetitions in a string. J Algorithms 5(3):422–432
20. Rytter W (2006) The structure of subword graphs and suffix trees of Fibonacci words. In: Implementation and application of automata, CIAA 2005. Lecture notes in computer science, vol 3845. Springer, Berlin, pp 250–261
21. Rytter W (2006) The number of runs in a string: improved analysis of the linear upper bound. In: Proceedings of the 23rd annual symposium on theoretical aspects of computer science. Lecture notes in computer science, vol 3884. Springer, Berlin, pp 184–195
22. Smyth WF (2000) Repetitive perhaps, but certainly not boring. Theor Comput Sci 249(2):343–355
23. Smyth WF (2003) Computing patterns in strings. Addison-Wesley, Boston

Stable Marriage

Robert W. Irving
School of Computing Science, University of Glasgow, Glasgow, UK

Keywords

Stable matching

Years and Authors of Summarized Original Work

1962; Gale, Shapley

Problem Definition

The objective in *stable matching problems* is to match together pairs of elements of a set of participants, taking into account the preferences of those involved and focusing on a stability requirement. The stability property ensures that no pair of participants would both prefer to be matched together rather than to accept their allocation in the matching. Such problems have widespread application, for example, in the allocation of medical students to hospital posts, students to schools or colleges, etc.

An instance of the classical *stable marriage problem* (SM), introduced by Gale and Shapley [2], involves a set of $2n$ participants comprising n men $\{m_1, \ldots, m_n\}$ and n women $\{w_1, \ldots, w_n\}$. Associated with each participant is a *preference list*, which is a total order over the participants of the opposite sex. A man m_i *prefers* woman w_j to woman w_k if w_j precedes w_k on the preference list of m_i and similarly for the women.

A *matching* M is a bijection between the sets of men and women, in other words a set of man-woman pairs so that each man and each woman belongs to exactly one pair of M. For a man m_i, $M(m_i)$ denotes the *partner* of m_i in M, i.e., the unique woman w_j such that (m_i, w_j) is in M. Similarly, $M(w_j)$ denotes the partner of woman w_j in M. A matching M is *stable* if there is no *blocking pair*, namely, a pair (m_i, w_j) such that m_i prefers w_j to $M(m_i)$ and w_j prefers m_i to $M(w_j)$.

Relaxing the requirements that the numbers of men and women are equal and that each participant should rank *all* of the members of the opposite sex gives the *stable marriage problem with incomplete lists* (SMI). So an instance of SMI comprises a set of n_1 men $\{m_1, \ldots, m_{n1}\}$ and a set of n_2 women $\{w_1, \ldots, w_{n2}\}$, and each participant's preference list is a total order over a *subset* of the participants of the opposite sex. The implication is that if woman w_j does not appear on the list of man m_i, then she is not an acceptable partner for m_i and vice versa. A man-woman pair is *acceptable* if each member of the pair is on the preference list of the other, and a matching M is now a set of acceptable pairs such that each man and each woman is in *at most* one pair of M. In this context, a blocking pair for matching M is an acceptable pair (m_i, w_j) such that m_i either is unmatched in M or prefers w_j to $M(m_i)$ and, likewise, w_j either is unmatched or prefers m_i to $M(w_j)$. A matching is stable if it has no blocking pair. So in an instance of SMI, a stable matching need not match all of the participants.

Gale and Shapley also introduced a many-one version of stable marriage, which they called the *college admissions problem*, but which is now more usually referred to as the ▸ Hospitals/Residents Problem (HR) because of its well-known applications in the medical employment field. This problem is covered in detail in Entry 150 of this volume.

A comprehensive treatment of many aspects of the stable marriage problem, as of 1989, appears in the monograph of Gusfield and Irving [5]. A more recent detailed exposition is given by Manlove [14].

Key Results

Theorem 1 *For every instance of SM or SMI, there is at least one stable matching.*

Theorem 1 was proved constructively by Gale and Shapley [2] as a consequence of the algorithm that they gave to find a stable matching.

Theorem 2 *1. For a given instance of SM involving n men and n women, there is a $O(n^2)$ time algorithm that finds a stable matching.*
2. For a given instance of SMI in which the combined length of all the preference lists is a, there is a $O(a)$ time algorithm that finds a stable matching.

The algorithm for SMI is a simple extension of that for SM. Each can be formulated in a variety of ways, but is most usually expressed in terms of a sequence of "proposals" from the members of one sex to the members of the other. A pseudocode version of the SMI algorithm appears in Fig. 1, in which the traditional approach of allowing men to make proposals is adopted.

The complexity bound of Theorem 2(1) first appeared in Knuth's monograph on stable marriage [12]. The fact that this algorithm is asymptotically optimal was subsequently established by Ng and Hirschberg [17] via an adversary argument. On the other hand, Wilson [21] proved that the average running time, taken over all possible instances of SM, is $O(n \log n)$.

The algorithm of Fig. 1, in its various guises, has come to be known as the Gale-Shapley algorithm. The variant of the algorithm given here is called *man oriented*, because men have the advantage of proposing. Reversing the roles of men and women gives the *woman-oriented* variant. The "advantage" of proposing is remarkable, as spelled out in the next theorem.

Theorem 3 *The man-oriented version of the Gale-Shapley algorithm for SM or SMI yields the man-optimal stable matching in which each man has the best partner that he can have in any stable matching, but in which each woman has her worst possible partner. The woman-oriented version yields the woman-optimal stable matching, which has analogous properties favoring the women.*

```
M = ∅;
assign each person to be free;      /* i. e., not a member of a pair in M */
while (some man m is free and has not proposed to every woman on his list)
    m proposes to w, the first woman on his list to whom he has not proposed;
    if (w is free)
        add (m, w) to M;            /* w accepts m */
    else if (w prefers m to her current partner m′)
        remove (m′, w) from M;      /* w rejects m′, setting m′ free */
        add (m, w) to M;            /* w accepts m */
    else
        M remains unchanged;        /* w rejects m */
return M;
```

Stable Marriage, Fig. 1 The Gale-Shapley algorithm

The optimality property of Theorem 3 was established by Gale and Shapley [2], and the corresponding "pessimality" property was first observed by McVitie and Wilson [16].

As observed earlier, a stable matching for an instance of SMI need not match all of the participants. But the following striking result was established by Gale and Sotomayor [3] and Roth [19] (in the context of the more general HR problem).

Theorem 4 *In an instance of SMI, all stable matchings have the same size and match exactly the same subsets of the men and women.*

For a given instance of SM or SMI, there may be many different stable matchings. Indeed Knuth [12] showed that the maximum possible number of stable matchings grows exponentially with the number of participants. He also pointed out that the set of stable matchings forms a distributive lattice under a natural dominance relation, a result attributed to Conway. This powerful algebraic structure that underlies the set of stable matchings can be exploited algorithmically in a number of ways. For example, Gusfield [4] showed how all k stable matchings for an instance of SM can be generated in $O(n^2 + kn)$ time (▸ Optimal Stable Marriage).

Extensions of these problems that are important in practice, so-called SMT and SMTI (extensions of SM and SMI, respectively), allow the presence of *ties* in the preference lists. In this context, three different notions of stability have been defined [7] – *weak*, *strong*, and *super-stability*, depending on whether the definition of a blocking pair requires that both members should improve, or at least one member improves and the other is no worse off, or merely that neither member is worse off. The following theorem summarizes the basic algorithmic results for these three varieties of stable matchings.

Theorem 5 *For a given instance of SMT or SMTI:*

1. *A weakly stable matching is guaranteed to exist and can be found in $O(n^2)$ or $O(a)$ time, respectively.*
2. *A super-stable matching may or may not exist; if one does exist, it can be found in $O(n^2)$ or $O(a)$ time, respectively.*
3. *A strongly stable matching may or may not exist; if one does exist, it can be found in $O(n^3)$ or $O(na)$ time, respectively.*

Theorem 5 parts (1) and (2) are due to Irving [7] (for SMT) and Manlove [13] (for SMTI). Part (3) is due to Kavitha et al. [11], who improved earlier algorithms of Irving and Manlove.

It turns out that, in contrast to the situation described by Theorem 4, weakly stable matchings in SMTI can have different sizes. The natural problem of finding a maximum cardinality

weakly stable matching, even under severe restrictions on the ties, is NP-hard [15]. ▸ Stable Marriage with Ties and Incomplete Lists explores this problem further.

Interesting special cases of SM and its variants arise when the preference lists on one or both sides are derived from a "master" list that ranks participants (e.g.,, according to some objective criterion). Such problems are explored by Irving et al. [10].

The stable marriage problem is an example of a *bipartite* matching problem. The extension in which the bipartite requirement is dropped is the so-called *stable roommates* (SR) problem.

Gale and Shapley had observed that, unlike the case of SM, an instance of SR may or may not admit a stable matching, and Knuth [12] posed the problem of finding an efficient algorithm for SR or proving it NP-complete. Irving [6] established the following theorem via a nontrivial extension of the Gale-Shapley algorithm.

Theorem 6 *For a given instance of SR, there exists a $O(n^2)$ time algorithm to determine whether a stable matching exists and if so to find such a matching.*

Variants of SR may be defined, as for SM, in which preference lists may be incomplete and/or contain ties – these are denoted by SRI, SRT, and SRTI – and in the presence of ties, the three flavors of stability, weak, strong, and super, are again relevant.

Theorem 7 *For a given instance of SRT or SRTI:*

1. *A weakly stable matching may or may not exist, and it is an NP-complete problem to determine whether such a matching exists.*
2. *A super-stable matching may or may not exist; if one does exist, it can be found in $O(n^2)$ or $O(a)$ time, respectively.*
3. *A strongly stable matching may or may not exist; if one does exist, it can be found in $O(n^4)$ or $O(a^2)$ time, respectively.*

Theorem 7 part (1) is due to Ronn [18], part (2) is due to Irving and Manlove [9], and part (3) is due to Scott [20].

Applications

Undoubtedly the best known and most important applications of stable matching algorithms are in centralized matching schemes in the medical and educational domains. ▸ Hospitals/Residents Problem includes a summary of some of these applications.

Open Problems

The parallel complexity of stable marriage remains open. The best known parallel algorithm for SMI is due to Feder et al. [1] and has $O(\sqrt{a}\log^3 a)$ running time using a polynomially bounded number of processors. It is not known whether the problem is in NC, but nor is there a proof of P-completeness.

One of the open problems posed by Knuth in his early monograph on stable marriage [12] was that of determining the maximum possible number x_n of stable matchings for any SM instance involving n men and n women. This problem remains open, although Knuth himself showed that x_n grows exponentially with n. Irving and Leather [8] conjecture that, when n is a power of 2, this function satisfies the recurrence

$$x_n = 3x_{n/2}^2 - 2x_{x/4}^4.$$

Many open problems remain in the setting of weak stability, such as finding a good approximation algorithm for a maximum cardinality weakly stable matching – see ▸ Stable Marriage with Ties and Incomplete Lists – and enumerating all weakly stable matchings efficiently.

Cross-References

▸ Hospitals/Residents Problem
▸ Optimal Stable Marriage

S

▸ Ranked Matching
▸ Stable Marriage and Discrete Convex Analysis
▸ Stable Marriage with Ties and Incomplete Lists
▸ Stable Partition Problem

Recommended Reading

1. Feder T, Megiddo N, Plotkin SA (2000) A sublinear parallel algorithm for stable matching. Theor Comput Sci 233(1–2):297–308
2. Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69:9–15
3. Gale D, Sotomayor M (1985) Some remarks on the stable matching problem. Discret Appl Math 11:223–232
4. Gusfield D (1987) Three fast algorithms for four problems in stable marriage. SIAM J Comput 16(1):111–128
5. Gusfield D, Irving RW (1989) The stable marriage problem: structure and algorithms. MIT, Cambridge
6. Irving RW (1985) An efficient algorithm for the stable roommates problem. J Algorithms 6:577–595
7. Irving RW (1994) Stable marriage and indifference. Discret Appl Math 48:261–272
8. Irving RW, Leather P (1986) The complexity of counting stable marriages. SIAM J Comput 15(3):655–667
9. Irving RW, Manlove DF (2002) The stable roommates problem with ties. J Algorithms 43:85–105
10. Irving RW, Manlove DF, Scott S (2008) The stable marriage problem with master preference lists. Discret Appl Math 156:2959–2977
11. Kavitha T, Mehlhorn K, Michail D, Paluch K (2004) Strongly stable matchings in time O(nm), and extension to the H/R problem. In: Proceedings of the 21st symposium on theoretical aspects of computer science (STACS 2004). Lecture notes in computer science, vol 2996, pp 222–233. Springer, Berlin
12. Knuth DE (1976) Mariages stables. Les Presses de L'Université de Montréal, Montréal
13. Manlove DF (1999) Stable marriage with ties and unacceptable partners. Technical report TR-1999-29, Department of Computing Science, University of Glasgow
14. Manlove DF (2013) Algorithmics of matching under preferences. World Scientific, Singapore
15. Manlove DF, Irving RW, Iwama K, Miyazaki S, Morita Y (2002) Hard variants of stable marriage. Theor Comput Sci 276(1–2):261–279
16. McVitie D, Wilson LB (1971) The stable marriage problem. Commun ACM 14:486–490
17. Ng C, Hirschberg DS (1990) Lower bounds for the stable marriage problem and its variants. SIAM J Comput 19:71–77
18. Ronn E (1990) NP-complete stable matching problems. J Algorithms 11:285–304
19. Roth AE (1984) The evolution of the labor market for medical interns and residents: a case study in game theory. J Polit Econ 92(6):991–1016
20. Scott S (2005) A study of stable marriage problems with ties. Ph.D. thesis, Department of Computing Science, University of Glasgow
21. Wilson LB (1972) An analysis of the stable marriage assignment algorithm. BIT 12:569–575

Stable Marriage and Discrete Convex Analysis

Akihisa Tamura
Department of Mathematics, Keio University, Yokohama, Japan

Keywords

Stable matching

Years and Authors of Summarized Original Work

2000; Eguchi, Fujishige, Tamura, Fleiner

Problem Definition

In the stable marriage problem first defined by Gale and Shapley [7], there is one set each of men and women having the same size, and each person has a strict preference order on persons of the opposite gender. The problem is to find a matching such that there is no pair of a man and a woman who prefer each other to their partners in the matching. Such a matching is called a *stable marriage* (or *stable matching*). Gale and Shapley showed the existence of a stable marriage and gave an algorithm for finding one. Fleiner [4] extended the stable marriage problem to the framework of matroids, and Eguchi, Fujishige, and Tamura [3] extended this formulation to a more general one in terms of discrete convex analysis, which was developed by Murota [8, 9]. Their formulation is described as follows.

Let M and W be sets of men and women who attend a dance party at which each person dances a waltz T times and the number of times that

he/she can dance with the same person of the opposite gender is unlimited. The problem is to find an "agreeable" allocation of dance partners, in which each person is assigned at most T persons of the opposite gender with possible repetition. Let $E = M \times W$, i.e., the set of all man-woman pairs. Also define $E_{(i)} = \{i\} \times W$ for all $i \in M$ and $E_{(j)} = M \times \{j\}$ for all $j \in W$. Denoting by $x(i, j)$ the number of dances between man i and woman j, an allocation of dance partners can be described by a vector $x = (x(i, j) : i \in M, j \in W) \in \mathbf{Z}^E$, where $\mathbf{Z}$ denotes the set of all integers. For each $y \in \mathbf{Z}^E$ and $k \in M \cup W$, denote by $y_{(k)}$ the restriction of y on $E_{(k)}$. For example, for an allocation $x \in \mathbf{Z}^E$, $x_{(k)}$ represents the allocation of person k with respect to x. Each person k describes his/her preferences on allocations by using a value function $f_k : \mathbf{Z}^{E_{(k)}} \to \mathbf{R} \cup (-\infty)$, where $\mathbf{R}$ denotes the set of all reals and $f_k(y) = -\infty$ means that allocation $y \in \mathbf{Z}^{E_{(k)}}$ is unacceptable for k. Note that the valuation of each person on allocations is determined only by his/her allocations. Let dom $f_k = \{y | f_k(y) \in \mathbf{R}\}$. Assume that each value function f_k satisfies the following assumption:

(A) dom f_k is bounded and hereditary and has $\mathbf{0}$ as the minimum point, where $\mathbf{0}$ is the vector of all zeros and heredity means that for any $y, y' \in \mathbf{Z}^{E_{(k)}}$, $\mathbf{0} \le y' \le y \in$ dom f_k implies $y' \in$ dom f_k.

For example, the following value functions with $M = \{1\}$ and $W = \{2, 3\}$

$$f_1(x(1,2), x(1,3)) =$$
$$\begin{cases} 10(x(1,2) + x(1,3)) - x(1,2)^2 - x(1,3)^2 & \text{if } x(1,2), x(1,3) \ge 0 \\ & \text{and } x(1,2) + x(1,3) \le 3 \\ -\infty & \text{otherwise,} \end{cases}$$

$$f_j(x(1,j)) = \begin{cases} x(1,j) & \text{if } x(1,j) \in \{0,1,2,3\} (j = 2,3) \\ -\infty & \text{otherwise} \end{cases}$$

represent the case where (1) everyone wants to dance as many times, up to three, as possible and (2) man 1 wants to divide his dances between women 2 and 3 as equally as possible. Allocations $(x(1,2), x(1,3)) = (1,2)$ and (2,1) are stable in the sense below.

A vector $x \in \mathbf{Z}^E$ is called a *feasible allocation* if $x_{(k)} \in$ dom f_k for all $k \in M \cup W$. An allocation x is said to satisfy *incentive constraints* if each person has no incentive to unilaterally decrease the current units of x, that is, if it satisfies

$$f_k(x_{(k)}) = \max\{f_k(y) | y \le x_{(k)}\} \ (\forall k \in M \cup W). \quad (1)$$

An allocation x is called *unstable* if it does not satisfy incentive constraints or there exist $i \in M$, $j \in W$, $y' \in \mathbf{Z}^{E_{(i)}}$ and $y'' \in \mathbf{Z}^{E_{(j)}}$ such that

$$f_i(x_{(i)}) < f_i(y'), \quad (2)$$

$$y'(i, j') \le x(i, j') \quad (\forall j' \in W \backslash \{j\}), \quad (3)$$

$$f_j(x_{(j)}) < f_j(y''), \quad (4)$$

$$y''(i', j) \le x(i', j) \quad (\forall i' \in M \backslash \{i\}), \quad (5)$$

$$y'(i, j) = y''(i, j). \quad (6)$$

Conditions (2) and (3) say that man i can strictly increase his valuation by changing the current number of dances with j without increasing the numbers of dances with other women, and (4) and (5) describe a similar situation for women. Condition (6) requires that i and j agree on the number of dances between them. An allocation x is called *stable* if it is not unstable.

Problem 1 Given disjoint sets M and W and value functions $f_k : \mathbf{Z}^{E_{(k)}} \to \mathbf{R} \cup \{-\infty\}$ for $k \in M \cup W$ satisfying assumption (A), find a stable allocation x.

S

Remark 1 A time schedule for a given feasible allocation can be given by a famous result on graph coloring, namely, "any bipartite graph can be edge-colorable with the maximum degree colors."

Key Results

The work of Eguchi, Fujishige, and Tamura [3] gave a solution to Problem 1 in the case where each value function f_k is $\mathrm{M}^\natural$-concave.

Discrete Convex Analysis: $\mathrm{M}^\natural$-Concave Functions

Let V be a finite set. For each $S \subseteq V$, e_S denotes the characteristic vector of S defined by $e_S(v) = 1$ if $v \in S$ and $e_S(v) = 0$ otherwise. Also define e_0 as the zero vector in $\mathbf{Z}^V$. For a vector $x \in \mathbf{Z}^V$, its positive support $\mathrm{supp}^+(x)$ and negative support $\mathrm{supp}^-(x)$ are defined by $\mathrm{supp}^+(x) = \{u \in V | x(u) > 0\}$ and $\mathrm{supp}^-(x) = \{u \in V | x(u) < 0\}$. A function $f : \mathbf{Z}^V \to \mathbf{R} \cup \{-\infty\}$ is called *$M^\natural$-concave* if it satisfies the following condition $\forall\, x, y \in \mathrm{dom}\, f$, $\forall\, u \in \mathrm{supp}^+(x-y)$, $\exists v \in \mathrm{supp}^-(x-y) \cup \{0\}$:

$$f(x) + f(y) \leq f(x - e_u + e_v) + f(y + e_u - e_v).$$

The above condition says that the sum of the function values at two points does not decrease as the points symmetrically move one or two steps closer to each other on the set of integral lattice points of $\mathbf{Z}^V$. This is a discrete analogue of the fact that for an ordinary concave function, the sum of the function values at two points does not decrease as the points symmetrically move closer to each other on the straight line segment between the two points.

Example 1 A nonempty family $\mathcal{T}$ of subsets of V is called a *laminar family* if $X \cap Y = \emptyset$, $X \subseteq Y$ or $Y \subseteq X$ holds for every $X, Y \in \mathcal{T}$. For a laminar family $\mathcal{T}$ and a family of univariate concave functions $f_Y : \mathbf{R} \to \mathbf{R} \cup \{-\infty\}$ indexed by $Y \in \mathcal{T}$, the function $f : \mathbf{Z}^V \to \mathbf{R} \cup \{-\infty\}$ defined by

$$f(x) = \sum_{Y \in \mathcal{T}} f_Y \left(\sum_{v \in Y} x(v) \right) \quad (\forall\, x \in \mathbf{Z}^V)$$

is $\mathrm{M}^\natural$-concave. The stable marriage problem can be formulated as Problem 1 by using value functions of this type.

Example 2 For the independence family $\mathcal{I} \subseteq 2^V$ of a matroid on V and $w \in \mathbf{R}^V$, the function $f : \mathbf{Z}^V \to \mathbf{R} \cup \{-\infty\}$ defined by

$$f(x) = \begin{cases} \sum_{u \in X} w(u) & \text{if } x = e_X \text{ for some } X \in \mathcal{I} \\ -\infty & \text{otherwise} \end{cases} \quad (\forall\, x \in \mathbf{Z}^V)$$

is $\mathrm{M}^\natural$-concave. Fleiner [4] showed that there always exists a stable allocation for value functions of this type.

Theorem 1 ([6]) *Assume that the value functions $f_k (k \in M \cup W)$ are $\mathrm{M}^\natural$-concave satisfying (A). Then, a feasible allocation x is stable if and only if there exist $Z_M = (z_{(i)} | i \in M) \in (\mathbf{Z} \cup \{+\infty\})^E$ and $z_W = (z_{(j)} | j \in W) \in (\mathbf{Z} \cup \{+\infty\})^E$ such that*

$$x_{(i)} \in \arg\max\{f_i(y) | y \leq z_{(i)}\} \quad (\forall\, i \in M), \tag{7}$$

$$x_{(j)} \in \arg\max\{f_j(y) | y \leq z_{(j)}\} \quad (\forall\, j \in W), \tag{8}$$

$$z_M(e) = +\infty \text{ or } z_W(e) = +\infty \quad (\forall\, e \in E), \tag{9}$$

where $\arg\max\{f_i(y) | y \leq z_{(i)}\}$ denotes the set of all maximizers of f_i under the constraints $y \leq z_{(i)}$.

Theorem 2 ([3]) *Assume that the value functions $f_k (k \in M \cup W)$ are $\mathrm{M}^\natural$-concave satisfying (A). Then, there always exists a stable allocation.*

Eguchi, Fujishige, and Tamura [3] proved Theorem 2 by showing that the following algorithm finds a feasible allocation x, and z_M, z_W satisfying (7), (8), and (9).

Here, $z_W \vee x_M$ is defined by $(z_W \vee x_M)(e) = \max\{z_W(e), x_M(e)\}$ for all $e \in E$.

Algorithm EXTENDED-GS

Input: $M^\natural$-concave functions f_M, f_W with $f_M(x) = \sum_{i \in M} f_i(x_{(i)})$ and $f_W(x) = \sum_{j \in W} f_j(x_{(j)})$;
Output: (x, z_M, z_W) satisfying (7), (8), and (9);
$z_M := (+\infty, \cdots, +\infty)$, $z_W := x_W := \mathbf{0}$;
repeat{
let x_M be any element in
$\arg\max\{f_M(y) | x_W \le y \le z_M\}$;
let x_W be any element in
$\arg\max\{f_W(y) | y \le x_M\}$;
for each $e \in E$ with $x_M(e) > x_W(e)$ {
$z_M(e) := x_W(e)$;
$z_W(e) := +\infty$;
};
} **until** $x_M = x_W$;
return $(x_M, z_M, z_W \vee x_M)$.

Applications

Abraham, Irving, and Manlove [1] dealt with a student-project allocation problem which is a concrete example of models in [4] and [3] and discussed the structure of stable allocations.

Fleiner [5] generalized the stable marriage problem and its extension in [4] to a wide framework and showed the existence of a stable allocation by using a fixed point theorem.

Fujishige and Tamura [6] proposed a common generalization of the stable marriage problem and the assignment game defined by Shapley and Shubik [10] by utilizing $M^\natural$-concave functions and gave a constructive proof of the existence of a stable allocation.

Open Problems

Algorithm EXTENDED-GS solves the maximization problem of an $M^\natural$-concave function in each iteration. A maximization problem of an $M^\natural$-concave function f on E can be solved in polynomial time in $|E|$ and $\log L$, where $L = \max\{||x - y||_\infty | x, y \in \text{dom } f\}$, provided that the function value $f(x)$ can be calculated in constant time for each x [11, 12]. Eguchi, Fujishige, and Tamura [3] showed that EXTENDED-GS terminates after at most L iterations, where L is defined by $\{||x||_\infty | x \in \text{dom } f_M\}$ in this case, and there exist a series of instances in which EXTENDED-GS requires numbers of iterations proportional to L. On the other hand, Baïou and Balinski [2] gave a polynomial time algorithm in $|E|$ for the special case where f_M and f_W are linear on rectangular domains. Whether a stable allocation for the general case can be found in polynomial time in $|E|$ and $\log L$ or not is open.

Cross-References

- Assignment Problem
- Hospitals/Residents Problem
- Optimal Stable Marriage
- Stable Marriage
- Stable Marriage with Ties and Incomplete Lists

Recommended Reading

1. Abraham DJ, Irving RW, Manlove DF (2007) Two algorithms for the student-project allocation problem. J Discret Algorithms 5:73–90
2. Baïou M, Balinski, M (2002) Erratum: the stable allocation (or ordinal transportation) problem. Math Oper Res 27:662–680
3. Eguchi A, Fujishige S, Tamura A (2003) A generalized Gale-Shapley algorithm for a discrete-concave stable-marriage model. In: Ibaraki T, Katoh N, Ono H (eds) Algorithms and computation: 14th international symposium (ISAAC 2003), Kyoto. LNCS, vol 2906. Springer, Berlin, pp 495–504
4. Fleiner T (2001) A matroid generalization of the stable matching polytope. In: Gerards B, Aardal K (eds) Integer programming and combinatorial optimization: 8th international IPCO conference, Utrecht. LNCS, vol 2081. Springer, Berlin, pp 105–114
5. Fleiner T (2003) A fixed point approach to stable matchings and some applications. Math Oper Res 28:103–126
6. Fujishige S, Tamura A (2007) A two-sided discrete-concave market with bounded side payments: an approach by discrete convex analysis. Math Oper Res 32:136–155
7. Gale D, Shapley SL (1962) College admissions and the stability of marriage. Am Math Mon 69:9–15
8. Murota K (1998) Discrete convex analysis. Math Program 83:313–371
9. Murota K (2003) Discrete convex analysis. Society for Industrial and Applied Mathematics, Philadelphia
10. Shapley SL, Shubik M (1971) The assignment game I: the core. Int J Game Theor 1:111–130

11. Shioura A (2004) Fast scaling algorithms for M-convex function minimization with application to the resource allocation problem. Discret Appl Math 134:303–316
12. Tamura A (2005) Coordinatewise domain scaling algorithm for M-convex function minimization. Math Program 102:339–354

Stable Marriage with One-Sided Ties

Hiroki Yanagisawa
IBM Research – Tokyo, Tokyo, Japan

Keywords

Approximation algorithms; Incomplete lists; Integer programming; Linear programming relaxation; One-sided ties; Stable marriage problem

Years and Authors of Summarized Original Work

2007; Halldórsson, Iwama, Miyazaki, Yanagisawa
2014; Huang, Iwama, Miyazaki, Yanagisawa

Problem Definition

Over the last 50 years, the stable marriage problem has been extensively studied for many problem settings (see, e.g., [11]), and one of the most intensively studied problem settings is MAX SMTI (MAXimum Stable Marriage with Ties and Incomplete lists). An input for the stable marriage problem consists of n men, n women, and each person's preference list for the people of the opposite sex. In MAX SMTI, the preference list of each person can be incomplete, which means that each person is allowed to exclude unacceptable people from the preference list, and the preference list of each person is allowed to include *ties* to show indifference between two or more people.

The objective of MAX SMTI is to find the largest matching that satisfies a stability condition. Before describing the stability condition, we review some notation. A matching M is defined as a set of pairs of man m and woman w such that m and w are acceptable to each other. The *size* of a matching M is defined as the number of pairs in M. We say that a person p is *single* if p is not matched in M. When man m and woman w are matched in M, we write $M(m) = w$ and $M(w) = m$. We say that matching M is *stable* if it does not contain any pair of man and woman, each of whom prefers the other to the partner in M (if any). More precisely, a matching M is stable if there is no pair of man m' and woman w' that satisfy all three conditions (i)–(iii): (i) m' and w' are acceptable to each other but not matched in M, (ii) m' is single in M or m' strictly prefers w' to $M(m')$, and (iii) w' is single in M or w' strictly prefers m' to $M(w')$. MAX SMTI asks us to find a stable matching of the largest size, and this problem is known to be NP-hard [12]. Therefore, the approximability of this problem has been intensively studied.

In this entry, we show recent results for two major variants of MAX SMTI. One of the variants is MAX SMOTI (MAXimum Stable Marriage with One-Sided Ties and Incomplete lists), in which only women are allowed to include ties in their preference lists and the preference lists of men are strictly ordered. The other variant is MAX SSMTI (Special SMTI), which is an even more restricted variant of MAX SMOTI where the ties are only allowed at the ends of the women's preference lists. Note that these two variants are still known to be NP-hard [12].

Problem 1 (MAX SMOTI)
INPUT: n men, n women, and each person's preference list, where only women have ties
OUTPUT: A stable matching of maximum size

Problem 2 (MAX SSMTI)
INPUT: n men, n women, and each person's preference list, where ties are at the ends of the women's preference lists
OUTPUT: A stable matching of maximum size

Stable Marriage with One-Sided Ties, Table 1 Examples of instances for MAX SMOTI and MAX SSMTI

MAX SMOTI		MAX SSMTI	
$m_1 : w_2\ w_1$	$w_1 : m_1\ m_2$	$m_1 : w_1\ w_3$	$w_1 : (m_1\ m_2\ m_3)$
$m_2 : w_2\ w_3\ w_1$	$w_2 : (m_1\ m_2)\ m_3$	$m_2 : w_2\ w_3\ w_1$	$w_2 : m_2$
$m_3 : w_3\ w_2$	$w_3 : m_2\ m_3$	$m_3 : w_3\ w_1$	$w_3 : m_2\ (m_1\ m_3)$

Examples

Table 1 shows examples of instances for MAX SMOTI and MAX SSMTI. The instance for MAX SMOTI contains a set of men $\{m_1, m_2, m_3\}$ and a set of women $\{w_1, w_2, w_3\}$. The preference list of each person is described in decreasing order of preference, and tied people are enclosed in a pair of parenthesis. For example, woman w_2 is indifferent between m_1 and m_2 but prefers m_1 or m_2 over m_3. A matching $M = \{(m_2, w_1), (m_3, w_2)\}$ is not stable for this MAX SMOTI instance, because m_2 strictly prefers w_2 to w_1 $(= M(m_2))$ and w_2 strictly prefers m_2 to m_3 $(= M(w_2))$. An example of a stable matching for this instance is $M' = \{(m_1, w_2), (m_2, w_3)\}$, and we can find another larger stable matching $M^* = \{(m_1, w_1), (m_2, w_2), (m_3, w_3)\}$ of size 3.

Key Results

Here we review past research on MAX SMOTI and MAX SSMTI. We start by describing a simple proposal-based algorithm (often referred to as the Gale-Shapley algorithm or the deferred acceptance algorithm), which is guaranteed to find a *stable* matching. In this algorithm, all of the men and women are initially set to be single. We pick an arbitrary man m who is single, and let man m propose to woman w at the top of his preference list. When man m proposes to w, he deletes woman w from his preference list. Woman w always accepts any proposal if she is single, which makes a matching pair of m and w. We repeat this proposal procedure to find more and more matching pairs. When a woman w, who is already matched to a man m, receives another proposal from man m', woman w chooses the more highly ranked man based on her preference list. (That is, the matching partner of w is unchanged if w prefers m to m', and the matching partner of w is changed from m to m' and m becomes unmatched if w prefers m' to m.) If m and m' are tied in w's preference list, then w chooses an arbitrary man. The proposal procedure continues until we cannot find any man who can propose. (That is, this algorithm terminates when all of the men become matched or the preference lists of all single men become empty.) Any matching obtained by this algorithm can be proven to be stable. The size of the obtained stable matching depends mostly on the decisions by women when a woman receives two proposals from men who are tied in her preference list. In the worst case, the size of an obtained matching can be half of the optimum matching, and hence, the approximation ratio of this algorithm is 2. It was an open problem whether or not there exists an approximation algorithm whose approximation ratio is strictly better than 2. Iwama, Miyazaki, and Yamauchi [8] provided an affirmative answer for this open problem with a 1.875-approximation algorithm.

After this breakthrough, Király [10] developed a new simple 1.5-approximation algorithm for MAX SMOTI (which also applies to MAX SSMTI) by improving the decision strategy of the proposal-based algorithm when women receive multiple proposals from tied men. His algorithm proceeds in the same way as the proposal-based algorithm until one of the men's preference lists become empty. When the preference list of a man becomes empty, he enters into his second round. Specifically, he recovers his original preference list so that he can propose to the women in his original preference list again, but his status is changed to "promoted." A promoted man is not allowed to recover his original preference list when his preference list becomes empty the second time, and hence, no man can enter a

third round in Kiráry's algorithm. The decision strategy of the women is changed so that a woman is forced to choose a promoted man (if one exists) when she receives two proposals from men who are tied in her preference list. This improvement of the decision strategy is the key to achieve the 1.5-approximation.

Iwama, Miyazaki, and Yanagisawa [9] further improved the approximation ratio to 25/17 (< 1.4706) for MAX SMOTI with a new algorithm GSA-LP, which uses a more complex proposal sequence of the men and a more sophisticated decision strategy for the women. In GSA-LP, we compute an optimum solution for a linear programming relaxation of a natural integer programming formulation of the problem in advance and use it for the decision strategy of the women. In addition, the proposal sequence is changed so that a man can propose to a woman many times, and a man is allowed to recover his original preference list at most twice (in other words, a man is allowed to go into a third round). These changes yield an improved approximation ratio for MAX SMOTI. Very recently, GSA-LP was shown to achieve a 1.25-approximation for MAX SSMTI [5].

For MAX SMOTI, there are successive improvements over GSA-LP. Huang and Kavitha [4] developed another new algorithm that achieves a 22/15 (<1.4667)-approximation by using a maximum matching algorithm. Radnai [14] showed 41/28 (<1.4643)-approximation by using a more detailed analysis of this new algorithm and also showed that a lower bound of the approximation ratio of this algorithm is at least 13/9 (>1.4444). Dean and Jalasutram improved the analysis of GSA-LP and showed that the approximation ratio of GSA-LP is at most 19/13 (<1.4616) if we increase the number of rounds from three to four [1]. We also note that if the lengths of ties are restricted to two, then the approximation ratio of this restricted MAX SMOTI variant can be further improved. A randomized algorithm [2] achieves 10/7 (<1.4286)-approximation and Huang and Kavitha devised another deterministic algorithm [4] with the same approximation ratio.

For the negative side, both MAX SMOTI and MAX SSMTI are NP-hard to approximate within any constant factor better than 21/19 (>1.1052) and hard to approximate within any constant factor better than 5/4 (= 1.25) under the unique games conjecture [3, 15]. These lower bounds hold even if we restrict the lengths of the ties to two. Note that the approximation ratio of the GSA-LP algorithm for MAX SSMTI is 1.25, which matches the lower bound under the unique games conjecture.

Applications

MAX SSMTI was introduced by Irving and Manlove [6] based on an actual application of the Scottish Foundation Allocation Scheme, which allocates residents (medical students) to hospitals. In this scheme, each resident submits a strictly ordered preference list, while each hospital submits a preference list that may contain one tie of arbitrary length at the end of the list. The objective of this allocation scheme is to maximize the number of allocated residents, and it is easy to reformulate this many-to-one allocation scheme as a one-to-one matching problem (MAX SSMTI) using a cloning technique [11].

Open Problems

An obvious future goal is to narrow the gap between the upper and lower bounds of the approximability of MAX SMOTI. Assuming the unique games conjecture is true, we now know that the best possible approximation ratio is between 1.4616 and 1.25. Even if we restrict the lengths of ties to two, all we can do now is reduce the upper bound slightly down to 1.4286. Thus, there is still much room for improvement.

As for MAX SSMTI, the 1.25-approximation of the GSA-LP algorithm is the best possible if the unique games conjecture is true. A future project could investigate if we can construct a faster approximation algorithm, because the

GSA-LP algorithm uses a linear programming relaxation technique, which takes superlinear time in the worst case.

Experimental Results

Irving and Manlove [7] reported on experimental evaluations of some heuristic algorithms including the Király's algorithm on real-world and random instances for MAX SMOTI. Subsequently, Podhradský [13] conducted experimental evaluations on random instances for MAX SMOTI and MAX SSMTI using some other heuristic algorithms including GSA-LP.

Cross-References

- ▸ Simpler Approximation for Stable Marriage
- ▸ Stable Marriage
- ▸ Stable Marriage with Ties and Incomplete Lists

Recommended Reading

1. Dean BC, Jalasutram R (2015) Factor revealing LPs and stable matching with ties and incomplete lists. In Proceedings of the 3rd international workshop on matching under preferences, 2015 (to appear)
2. Halldórsson MM, Iwama K, Miyazaki S, Yanagisawa H (2004) Randomized approximation of the stable marriage problem. Theor Comput Sci 325(3):439–465
3. Halldórsson MM, Iwama K, Miyazaki S, Yanagisawa H (2007) Improved approximation results for the stable marriage problem. ACM Trans Algorithms 3(3):Article No. 30
4. Huang CC, Kavitha T (2014) An improved approximation algorithm for the stable marriage problem with one-sided ties. In: Proceedings of IPCO 2014, Bonn, pp 297–308
5. Huang CC, Iwama K, Miyazaki S, Yanagisawa H (2015) Approximability of finding largest stable matchings. Manuscript under submission
6. Irving RW, Manlove DF (2008) Approximation algorithms for hard variants of the stable marriage and hospital/residents problems. J Comb Optim 16(3):279–292
7. Irving RW, Manlove DF (2009) Finding large stable matchings. J Exp Algorithmics 14:Article No. 2
8. Iwama K, Miyazaki S, Yamauchi N (2007) A 1.875: approximation algorithm for the stable marriage problem. In: Proceedings of SODA 2007, New Orleans, pp 288–297
9. Iwama K, Miyazaki S, Yanagisawa H (2014) A 25/17-approximation algorithm for the stable marriage problem with one-sided ties. Algorithmica 68(3):758–775
10. Király Z (2011) Better and simpler approximation algorithms for the stable marriage problem. Algorithmica 60(1):3–20
11. Manlove DF (2013) Algorithmics of matching under preferences. World Scientific, Hackensack
12. Manlove DF, Irving RW, Iwama K, Miyazaki S, Morita Y (2002) Hard variants of stable marriage. Theor Comput Sci 276(1–2):261–279
13. Podhradský A (2010) Stable marriage problem algorithms. Master's thesis, Faculty of Informatics, Masaryk University
14. Radnai A (2014) Approximation algorithms for the stable matching problem. Master's thesis, Eötvös Loránd University
15. Yanagisawa H (2007) Approximation algorithms for stable marriage problems. Ph.D. thesis, Kyoto University

Stable Marriage with Ties and Incomplete Lists

Kazuo Iwama[1,2] and Shuichi Miyazaki[3]
[1]Computer Engineering, Kyoto University, Sakyo, Kyoto, Japan
[2]School of Informatics, Kyoto University, Sakyo, Kyoto, Japan
[3]Academic Center for Computing and Media Studies, Kyoto University, Kyoto, Japan

Synonyms

Stable matching problem

Keywords

Approximation algorithms; Incomplete lists; Stable marriage problem; Ties

Years and Authors of Summarized Original Work

2007; Iwama, Miyazaki, Yamauchi

Problem Definition

In the original setting of the stable marriage problem introduced by Gale and Shapley [2], each preference list has to include all members of the other party, and furthermore, each preference list must be totally ordered (see entry ▸ Stable Marriage also).

One natural extension of the problem is then to allow persons to include ties in preference lists. In this extension, there are three variants of the stability definition, super-stability, strong stability, and weak stability (see below for definitions). In the first two stability definitions, there are instances that admit no stable matching, but there is a polynomial-time algorithm in each case that determines if a given instance admits a stable matching and finds one if one exists [9]. On the other hand, in the case of weak stability, there always exists a stable matching, and one can be found in polynomial time.

Another possible extension is to allow persons to declare unacceptable partners, so that preference lists may be incomplete. In this case, every instance admits at least one stable matching, but a stable matching may not be a perfect matching. However, if there are two or more stable matchings for one instance, then all of them have the same size [3].

The problem treated in this entry allows both extensions simultaneously, which is denoted as SMTI (stable marriage with ties and incomplete lists).

Notations

An instance I of SMTI comprises n men, n women, and each person's preference list that may be incomplete and may include ties. If a man m includes a woman w in his list, w is *acceptable* to m. $w_i \succ_m w_j$ means that m strictly prefers w_i to w_j in I. $w_i =_m w_j$ means that w_i and w_j are tied in m's list (including the case $w_i = w_j$). The statement $w_i \succeq_m w_j$ is true if and only if $w_i \succ_m w_j$ or $w_i =_m w_j$. Similar notations are used for women's preference lists. A matching M is a set of pairs (m, w) such that m is acceptable to w, and vice versa, and each person appears at most once in M. If a man m is matched with a woman w in M, it is written as $M(m) = w$ and $M(w) = m$.

A man m and a woman w are said to form a *blocking pair for weak stability* for M if they are not matched together in M, but by matching them, both become better off, namely, (i) $M(m) \neq w$ but m and w are acceptable to each other, (ii) $w \succ_m M(m)$ or m is single in M, and (iii) $m \succ_w M(w)$ or w is single in M.

Two persons x and y are said to form a *blocking pair for strong stability* for M if they are not matched together in M, but by matching them, one becomes better off, and the other does not become worse off, namely, (i) $M(x) \neq y$ but x and y are acceptable to each other, (ii) $y \succ_x M(x)$ or x is single in M, and (iii) $x \succeq_y M(y)$ or y is single in M.

A man m and a woman w are said to form a *blocking pair for super-stability* for M if they are not matched together in M, but by matching them, neither becomes worse off, namely, (i) $M(m) \neq w$ but m and w are acceptable to each other, (ii) $w \succeq_m M(m)$ or m is single in M, and (iii) $m \succeq_w M(w)$ or w is single in M.

A matching M is called *weakly stable* (*strongly stable* and *super-stable*, respectively) if there is no blocking pair for weak (strong and super, respectively) stability for M.

Problem 1 (SMTI)

INPUT: n men, n women, and each person's preference list
OUTPUT: A stable matching

Problem 2 (MAX SMTI)

INPUT: n men, n women, and each person's preference list
OUTPUT: A stable matching of maximum size

The following problem is a restriction of MAX SMTI in terms of the length of preference lists:

Problem 3 ((p,q)-MAX SMTI)

INPUT: n men, n women, and each person's preference list, where each man's preference list includes at most p women and each woman's preference list includes at most q men
OUTPUT: A stable matching of maximum size

Definition of the Approximation Ratio

A goodness measure of an approximation algorithm T for a maximization problem is defined as follows: the *approximation ratio* of T is $\max\{opt(x)/T(x)\}$ over all instances x of size N, where $opt(x)$ and $T(x)$ are the sizes of the optimal and the algorithm's solutions, respectively.

Key Results

SMTI and MAX SMTI in Super-Stability and Strong Stability

Theorem 1 ([20]) *There is an $O(n^2)$-time algorithm that determines if a given SMTI instance admits a super-stable matching and finds one if one exists.*

Theorem 2 ([17]) *There is an $O(n^3)$-time algorithm that determines if a given SMTI instance admits a strongly stable matching and finds one if one exists.*

It is shown that all stable matchings for a fixed instance are of the same size [20]. Therefore, the above theorems imply that MAX SMTI can also be solved in the same time complexity.

SMTI and MAX SMTI in Weak Stability

In the case of weak stability, every instance admits at least one stable matching, but one instance can have stable matchings of different sizes. If the size is not important, a stable matching can be found in polynomial time by breaking ties arbitrarily and applying the Gale-Shapley algorithm.

Theorem 3 *There is an $O(n^2)$-time algorithm that finds a weakly stable matching for a given SMTI instance.*

However, if larger stable matchings are required, the problem becomes hard.

Theorem 4 ([5, 13, 21, 24]) *MAX SMTI is NP-hard and cannot be approximated within $33/29 - \epsilon$ for any positive constant ϵ unless P=NP. ($33/29 > 1.137$)*

The following approximation ratio is achieved by a local search type algorithm.

Theorem 5 ([14]) *There is a polynomial-time approximation algorithm for MAX SMTI whose approximation ratio is at most 15/8 (=1.875).*

There are a couple of approximation algorithms for restricted inputs.

Theorem 6 ([6]) *There is a polynomial-time randomized approximation algorithm for MAX SMTI whose expected approximation ratio is at most $10/7(\simeq 1.429)$ if, in a given instance, ties appear in one side only and the length of each tie is two.*

Theorem 7 ([6]) *There is a polynomial-time randomized approximation algorithm for MAX SMTI whose expected approximation ratio is at most $7/4(= 1.75)$ if, in a given instance, the length of each tie is two.*

Theorem 8 ([7]) *There is a polynomial-time approximation algorithm for MAX SMTI whose approximation ratio is at most $2/(1 + L^{-2})$ if, in a given instance, ties appear in one side only and the length of each tie is at most L.*

Theorem 9 ([7]) *There is a polynomial-time approximation algorithm for MAX SMTI whose approximation ratio is at most $13/7(\simeq 1.858)$ if, in a given instance, the length of each tie is two.*

(p, q)-MAX SMTI in Weak Stability

Irving et al. [12] show the boundary between P and NP-hardness in terms of the length of preference lists.

Theorem 10 ([12]) *$(2,\infty)$-MAX SMTI is solvable in time $O(n^{\frac{3}{2}} \log n)$.*

Theorem 11 ([12]) *$(3,3)$-MAX SMTI is NP-hard.*

Theorem 12 ([12]) *$(3,4)$-MAX SMTI is NP-hard and cannot be approximated within some constant $\delta(> 1)$ unless P=NP.*

Applications

One of the most famous applications of the stable marriage problem is a centralized assignment system between medical students (residents) and

hospitals. This is an extension of the stable marriage problem to a many-one variant: Each hospital declares the number of residents it can accept, which may be more than one, while each resident has to be assigned to at most one hospital. Actually, there are several applications in the world, known as NRMP in the USA [4], CaRMS in Canada [1], SFAS (previously known as SPA) in Scotland [10, 11], and JRMP in Japan [16]. One of the optimization criteria is clearly the number of matched residents. In a real-world application such as the above hospitals-residents matching, hospitals and residents tend to submit short preference lists that may include ties, in which case, the problem can be naturally considered as MAX SMTI.

Open Problems

An apparent open problem is to narrow the gap of approximability and inapproximability of MAX SMTI in weak stability.

Since the publication of the key result of this chapter (Theorem 5), there have been a lot of improvement. Király [18] presented a linear time 5/3-approximation algorithm (see ▸ Simpler Approximation for Stable Marriage). McDermid [22] then presented a 1.5-approximation algorithm (see ▸ Simpler Approximation for Stable Marriage), and Király [19] and Paluch [23] presented simpler algorithms with the same approximation ratio, which is the current best upper bound. The lower bound was improved by Yanagisawa [24], who showed that MAX SMTI is inapproximable to within a ratio smaller than $33/29(>1.137)$ unless P = NP. He also showed that MAX SMTI is inapproximable within a ratio smaller than $4/3(>1.333)$ under the Unique Games Conjecture (UGC).

As for the special case where ties can appear in one side only (see ▸ Stable Marriage with One-Sided Ties), Király [18] presented a 1.5-approximation algorithm. It was then improved to $25/17(<1.471)$ [15] and to $22/15(<1.467)$ [8], which is the current best upper bound. The current best lower bounds are $21/19(\simeq 1.105)$ under P$\neq$NP and 1.25 under UGC [7].

Cross-References

- ▸ Simpler Approximation for Stable Marriage
- ▸ Assignment Problem
- ▸ Hospitals/Residents Problem
- ▸ Hospitals/Residents Problems with Quota Lower Bounds
- ▸ Optimal Stable Marriage
- ▸ Ranked Matching
- ▸ Stable Marriage
- ▸ Stable Marriage and Discrete Convex Analysis
- ▸ Stable Partition Problem
- ▸ Simpler Approximation for Stable Marriage
- ▸ Stable Marriage with One-Sided Ties

Recommended Reading

1. Canadian Resident Matching Service (CaRMS), http://www.carms.ca/
2. Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69:9–15
3. Gale D, Sotomayor M (1985) Some remarks on the stable matching problem. Discret Appl Math 11:223–232
4. Gusfield D, Irving RW (1989) The stable marriage problem: structure and algorithms. MIT Press, Boston
5. Halldórsson MM, Irving RW, Iwama K, Manlove DF, Miyazaki S, Morita Y, Scott S (2003) Approximability results for stable marriage problems with ties. Theor Comput Sci 306:431–447
6. Halldórsson MM, Iwama Ka, Miyazaki S, Yanagisawa H (2004) Randomized approximation of the stable marriage problem. Theor Comput Sci 325(3):439–465
7. Halldórsson MM, Iwama Ka, Miyazaki S, Yanagisawa H (2007) Improved approximation results of the stable marriage problem. ACM Trans Algorithms 3(3):Article No. 30
8. Huang C-C, Kavitha T (2014) An improved approximation algorithm for the stable marriage problem with one-sided ties. In: Proceedings of the IPCO 2014, Bonn. LNCS, vol 8494, pp 297–308
9. Irving RW (1994) Stable marriage and indifference. Discret Appl Math 48:261–272
10. Irving RW (1998) Matching medical students to pairs of hospitals: a new variation on a well-known theme. In: Proceedings of the ESA 1998, Venice. LNCS, vol 1461, pp 381–392
11. Irving RW, Manlove DF, Scott S (2000) The hospitals/residents problem with ties. In: Proceedings of the SWAT 2000, Bergen. LNCS, vol 1851, pp 259–271

12. Irving RW, Manlove DF, O'Malley G (2009) Stable marriage with ties and bounded length preference lists. J Discret Algorithms 7(2):213–219
13. Iwama K, Manlove DF, Miyazaki S, Morita Y (1999) Stable marriage with incomplete lists and ties. In: Proceedings of the ICALP 1999, Prague. LNCS, vol 1644, pp 443–452
14. Iwama K, Miyazaki S, Yamauchi N (2007) A 1.875-approximation algorithm for the stable marriage problem. In: Proceedings of the SODA 2007, New Orleans, pp 288–297
15. K. Iwama, Miyazaki S, Yanagisawa H (2014) A 25/17-approximation algorithm for the stable marriage problem with one-sided ties. Algorithmica 68:758–775
16. Japanese Resident Matching Program (JRMP), http://www.jrmp.jp/
17. Kavitha T, Mehlhorn K, Michail D, Paluch KE (2007) Strongly stable matchings in time $O(nm)$ and extension to the hospitals-residents problem. ACM Trans Algorithms 3(2):Article No. 15
18. Király Z (2011) Better and simpler approximation algorithms for the stable marriage problem. Algorithmica 60(1):3–20
19. Király Z (2013) Linear time local approximation algorithm for maximum stable marriage. MDPI Algorithms 6(3):471–484
20. Manlove DF (1999) Stable marriage with ties and unacceptable partners. Technical Report no. TR-1999-29 of the Computing Science Department of Glasgow University
21. Manlove DF, Irving RW, Iwama K, Miyazaki S, Morita Y (2002) Hard variants of stable marriage. Theor Comput Sci 276(1–2):261–279
22. McDermid EJ (2009) A 3/2-approximation algorithm for general stable marriage. In: Proceedings of the ICALP, Rhodes. LNCS, vol 5555, pp 689–700
23. Paluch KE (2014) Faster and simpler approximation of stable matchings. Algorithms 7(2):189–202
24. Yanagisawa H (2007) Approximation algorithms for stable marriage problems. Ph.D. Thesis, Kyoto University

Stable Partition Problem

Katarína Cechlárová
Faculty of Science, Institute of Mathematics,
P. J. Šafárik University, Košice, Slovakia

Keywords

Coalition formation; Hedonic games; Stability

Years and Authors of Summarized Original Work

2002; Cechlárová, Hajduková
2004; Ballester

Problem Definition

Let N be a finite set of players; a nonempty subset of N is called a coalition. Each player $i \in N$ has a preference relation $\succeq_i$ (complete, reflexive, and transitive) over all the coalitions that contain i. Notation $S \succeq_i T$ means that player i *weakly prefers* coalition S to coalition T; if $S \succeq_i T$ and not $T \succeq_i S$, then player i *strictly prefers* S to T, denoted by $S \succ_i T$. If $S \succeq_i T$ and $T \succeq_i S$, then player i *is indifferent between* coalitions S and T (there is a *tie* in his preference list). Player i has strict preferences if her preference list contains no ties. There are several possible ways of representing preferences, but it is usually supposed that preference relations can be evaluated in polynomial time.

An instance I of the stable partition problem (or coalition formation game, or hedonic game) is given by the set of players and their preferences.

A partition Π is a collection of disjoint coalitions whose union equals N. It is supposed that each participant's appreciation of a coalition structure only depends on the coalition $\Pi(i)$ she is a member and not on the composition of other coalitions. Of interest are partitions that fulfill some kind of stability requirements.

We say that a coalition $S \subseteq N$ *strongly blocks* a partition Π, if each player $i \in S$ strictly prefers S to $\Pi(i)$, and a coalition $S \subseteq N$ *weakly blocks* a partition Π, if each player $i \in S$ weakly prefers S to $\Pi(i)$ and there exists at least one player $j \in S$ who strictly prefers S to $\Pi(j)$. Partition Π is:

- *Individually stable* if each player i weakly prefers $\Pi(i)$ to $\{i\}$;
- *Nash stable (NS)* if each player i weakly prefers $\Pi(i)$ to $X \cup \{i\}$ for each $X \in \Pi \cup \emptyset$;

- *Individually stable (IS)* if whenever a player i strictly prefers $X \cup \{i\}$ to $\Pi(i)$ for some $X \in \Pi$, then $X \succ_j X \cup \{i\}$ for at least one player $j \in X$;
- *Contractually individually stable (CIS)* if whenever a player i strictly prefers $X \cup \{i\}$ to $\Pi(i)$ for some $X \in \Pi$, then $X \succ_j X \cup \{i\}$ for at least one player $j \in X$ or $\Pi(i) \succ_j \Pi(i) \backslash \{i\}$ for at least one player $i \in \Pi(i)$;
- *Core stable* if it admits no blocking coalition;
- *Strictly core stable* if it admits no weakly blocking coalition;

Most of these definitions were introduced in [3] and [4] where also some sufficient conditions for the existence of stable partitions were formulated. An overview of the implications between these definitions can be found in [1]. The following problems have been studied algorithmically for various stability notions $\mathbb{S}$:

- $\mathbb{S}$-STABILITY-VERIFICATION: Given I and a partition Π, is Π a $\mathbb{S}$-stable partition?
- $\mathbb{S}$-STABILITY-EXISTENCE: Given I, does a $\mathbb{S}$-stable partition exist?
- $\mathbb{S}$-STABILITY-CONSTRUCTION: Given I, construct a $\mathbb{S}$-stable partition.
- $\mathbb{S}$-STABILITY-STRUCTURE: Describe the structure of $\mathbb{S}$-stable partitions for a given I.

The computational complexity of these problems depends on the specification of the preference relation in the input.

An Important special case of the stable partition problem arises when each coalition can contain at most two players. This is known under the name the *Stable Matching Problem* and is treated in detail in [14]; see also references in the entry Stable Marriage.

Key Results

Trivial Encoding

In the *trivial encoding*, each player lists all her individually rational coalitions (i.e., those that player i weakly prefers to coalition $\{i\}$).

Theorem 1 *Under the trivial encoding, the* STABILITY-VERIFICATION *problem is polynomially solvable for any stability definition.* STABILITY-EXISTENCE *is NP-complete for IR, NS, core, and strict core [2].* CORE-STABILITY-EXISTENCE *is NP-complete [2], even in the case when each player i has her preference list of the form $C_1(i) \succ_i C_2(i) \succ_i \{i\}$ and all acceptable coalitions have size three [11].*

As the trivial encoding may be of exponential size in the number of players, more succinct preference representations have been studied.

Anonymous Preferences

Players have anonymous preferences if all coalitions of the same size are tied, i.e., players do not care about the actual content of the coalitions, only about their sizes.

Theorem 2 *Under anonymous preferences, the* CORE-STABILITY-VERIFICATION *problem is polynomially solvable and* CORE-STABILITY-EXISTENCE *is NP-complete [2].*

Additive Preferences

In an additive hedonic game, each player i has a real-valued function $v_i : N \to \mathbb{R}$ and $S \succ_i T$ if and only if $\sum_{j \in S} v_i(j) > \sum_{j \in T} v_i(j)$.

Theorem 3 *In additive hedonic games,* STABILITY-VERIFICATION *is co-NP-complete in the strong sense for core and strict core [1, 17].* CORE-STABILITY-EXISTENCE *and* STRICT-CORE-STABILITY-EXISTENCE *are strongly NP-hard [18] even in the symmetric case [1].* INDIVIDUAL-STABILITY-EXISTENCE *and* NASH-STABILITY-EXISTENCE *are strongly NP-complete [1, 18]. Moreover,* CORE-STABILITY-EXISTENCE *is $\sum_2^p$-complete [19].*

Special cases of additive preferences arise if $v_i(j) \in \{-1, |N|\}$ for each $i, j \in N$ (friend-oriented case) or $v_i(j) \in \{1, -|N|\}$ for each $i, j \in N$ (enemy-oriented case). Under friend-oriented as well as under enemy-oriented preferences, a core-stable partition always exists [12], however, the following assertion holds.

Theorem 4 ([12]) *Under enemy-oriented preferences,* CORE-STABILITY-VERIFICATION *and* CORE-STABILITY-CONSTRUCTION *are strongly NP-complete and NP-hard, respectively.*

Preferences Derived from the Best and/or Worst Player

Suppose that each player i linearly orders only individual players or, more precisely, a subset of them – these are *acceptable* for i.

Preferences over players are extended to preferences over coalitions on the basis of the best or the worst player in the coalition as follows:

B-preferences – a player orders coalitions first on the basis of the most preferred member of the coalition, and if those are equal or tied, the coalition with smaller cardinality is preferred;

W-preferences – a player orders coalitions on the basis of the least preferred member of the coalition;

BW-preferences – a player orders coalitions first on the basis of the best member of the coalition, and if those are equal or tied, the coalition with a more preferred worst member is preferred.

In this case, preferences are considered *strict*, if the preferences over individuals are strict, and they are called *dichotomous* if all acceptable participants are tied in each preference list.

Theorem 5 *Under B-preferences,* STABILITY-VERIFICATION *is polynomial for core and strict core. A strict core and a core stable partition always exist if preferences over players are strict [9]. However, if preferences over players contain ties,* STABILITY-EXISTENCE *for core and strict core is NP-complete [6]. In the dichotomous case, a core stable partition can be constructed in polynomial time, but* STRICT-CORE-STABILITY-EXISTENCE *is NP-complete [5].*

Let us remark here that in the case of strict preferences, a strict core stable partition can be found by the famous Top Trading Cycles algorithm [9, 20].

The stable partition problem under W-preferences was studied in [7] and many features similar to the Stable Roommates Problem [14] were described. First, if a blocking coalition exists, then there is a blocking coalition of size at most 2. Hence, CORE-STABILITY-VERIFICATION is polynomial. CORE-STABILITY-EXISTENCE and CORE-STABILITY-CONSTRUCTION are polynomial in the strict preferences case, which can be shown using an extension of Irving's Stable Roommates Algorithm (discussed in detail in [14]). This algorithm can also be used to derive some results for CORE-STABILITY-STRUCTURE. In the case of ties, CORE-STABILITY-EXISTENCE is NP-complete.

Under BW preferences, in the strict preferences case, a core partition always exists and one can be obtained by the Top Trading Cycles algorithm, but STRICT-CORE-STABILITY-EXISTENCE is NP-hard. If preferences contain ties, CORE-STABILITY-EXISTENCE is NP-hard too [8]. CORE-STABILITY-VERIFICATION remains open.

Applications

Stable partitions arise in various economic and game theoretical models. They appear in the study of countries formation [10] and in multi-agent coordination scenarios and social networking services [13]. Stability is also desired in barter exchange economies with discrete commodities [20, 21], including exchange of kidneys for transplantations [5, 16]. Notice that in case when the cooperation of players consists in the exchange of some items within one partition set, the exchange cycle has also to be specified.

Open Problems

Due to the great number of variants, a lot of open problems exists. In almost all cases, STABILITY-STRUCTURE is not satisfactorily solved. For instances with no stable partition, one may seek one that minimizes the number of players who have an incentive to deviate. Parallel algorithms were also not studied.

Experimental Results

Stochastic local search algorithms for CORE-STABILITY-VERIFICATION in the additive preferences case were reported in [15].

Cross-References

▸ Stable Marriage

Recommended Reading

1. Aziz H, Brandt F, Seeding HG (2013) Computing desirable partitions in additively separable hedonic games. Artif Intell 195:316–334
2. Ballester C (2004) NP-completeness in hedonic games. Games Econ Behav 49(1):1–30
3. Banerjee S, Konishi H, Sönmez T (2001) Core in a simple coalition formation game. Soc Choice Welf 18:135–153
4. Bogomolnaia A, Jackson MO (2002) The stability of hedonic coalition structures. Games Econ Behav 38(2):201–230
5. Cechlárová K, Fleiner T, Manlove D (2005) The kidney exchange game. In: Zadnik-Stirn L, Drobne S (eds) Proceedings of SOR'05, Slovenia, pp 77–83
6. Cechlárová K, Hajduková J (2002) Computational complexity of stable partitions with B-preferences. Int J Game Theory 31(3):353–364
7. Cechlárová K, Hajduková J (2004) Stable partitions with W-preferences. Discret Appl Math 138(3):333–347
8. Cechlárová K, Hajduková J (2004) Stability of partitions under WB-preferences and BW-preferences. Int J Inf Technol Decis Making 3(4):605–614. Special Issue on Computational Finance and Economics
9. Cechlárová K, Romero-Medina A (2001) Stability in coalition formation games. Int J Game Theory 29:487–494
10. Cechlárová K, Dahm M, Lacko V (2001) Efficiency and stability in a discrete model of country formation. J Glob Optim 20(3–4):239–256
11. Deineko VG, Woeginger GJ (2013) Two hardness results for core stability in additive hedonic coalition formation games. Discret Appl Math 161:1837–1842
12. Dimitrov D, Borm P, Hendrickx R, Sung SCh (2006) Simple priorities and core stability in hedonic games. Soc Choice Welf 26(2):421–433
13. Elkind E, Wooldridge M (2009) Hedonic coalition nets. In: Proceedings of the 8th international conference on autonomous agents and multiagent systems (AAMAS 2009), Budapest, pp 417–424
14. Gusfield D, Irving RW (1989) The stable marriage problem. Structure and algorithms. MIT, Cambridge
15. Keinänen H (2010) Stochastic local search for core membership checking in hedonic games. LNCS 6220:56–70
16. Roth A, Sönmez T, Ünver U (2004) Kidney exchange. Q J Econ 119:457–488
17. Sung SCh, Dimitrov D (2007) On core membership testing for hedonic coalition formation games. Oper Res Lett 35:155–158
18. Sung SCh, Dimitrov D (2010) Computational complexity in additive hedonic games. Eur J Oper Res 203:635–639
19. Woeginger GJ (2013) A hardness result for core stability in additive hedonic games. Math Soc Sci 65:101-104
20. Shapley L, Scarf H (1974) On cores and indivisibility. J Math Econ 1:23–37
21. Yuan Y (1996) Residence exchange wanted: a stable residence exchange problem. Eur J Oper Res 90:536–546

Stackelberg Games: The Price of Optimum

Alexis Kaporis[1] and Paul (Pavlos) Spirakis[2,3,4]
[1]Department of Information and Communication Systems Engineering, University of the Aegean, Karlovasi, Samos, Greece
[2]Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece
[3]Computer Science, University of Liverpool, Liverpool, UK
[4]Computer Technology Institute (CTI), Patras, Greece

Keywords

Coordination ratio; Cournot game

Years and Authors of Summarized Original Work

2006; Kaporis, Spirakis

Problem Definition

Stackelberg games [15] may model the interplay among an authority and rational individuals that selfishly demand resources on a large-scale

network. In such a game, the authority *(Leader)* of the network is modeled by a distinguished player. The selfish users *(Followers)* are modeled by the remaining players.

It is well known that selfish behavior may yield a *Nash Equilibrium* with cost arbitrarily higher than the optimum one, yielding unbounded *Coordination Ratio* or *Price of Anarchy (PoA)* [7, 13]. Leader plays his strategy first assigning a portion of the total demand to some resources of the network. Followers observe and react selfishly assigning their demand to the most appealing resources. Leader aims to drive the system to an a posteriori Nash equilibrium with cost close to the overall optimum one [4, 6, 8, 10]. Leader may also be eager for his own rather than system's performance [2, 3].

A Stackelberg game can be seen as a special, and easy [6] to implement, case of *Mechanism Design*. It avoids the complexities of either computing taxes or assigning prices, or even designing the network at hand [9]. However, a central authority capable to control the overall demand on the resources of a network may be unrealistic in networks which evolute and operate under the effect of many and diversing economic entities. A realistic way [4] to act centrally even in large nets could be via *Virtual Private Networks (VPNs)* [1]. Another flexible way is to combine such strategies with *Tolls* [5, 14].

A dictator controlling the entire demand optimally on the resources surely yields PoA = 1. On the other hand, rational users do prefer a liberal world to live. Thus, it is important to compute the optimal Leader strategy which controls the *minimum* of the resources *(Price of Optimum)* and yields PoA $=$ 1. What is the complexity of computing the Price of Optimum? This is not trivial to answer, since the Price of Optimum depends crucially on computing an optimal Leader strategy. In particular, [6] proved that computing the optimal Leader strategy is hard.

The central result of this lemma is Theorem 5. It says that on nonatomic flows and arbitrary $s-t$ networks and latencies, computing the minimum portion of flow and Leader's optimal strategy sufficient to induce PoA $=$ 1 is easy [10].

Problem $(G(V,E),s,t \in V,r)$ INPUT: Graph $G, \forall e \in E$ latency ℓ_e, flow r, a source-destination pair (s,t) of vertices in V.
OUTPUT: (i) The minimum portion α_G of the total flow r sufficient for an optimal Stackelberg strategy to induce the optimum on G. (ii) The optimal Stackelberg strategy.

Models and Notations

Consider a graph $G(V,E)$ with parallel edges allowed. A number of rational and selfish users wish to route from a given source s to a destination node t an amount of flow r. Alternatively, consider a partition of users in k commodities, where user(s) in commodity i wish to route flow r_i through a source-destination pair (s_i,t_i), for each $i = 1,\ldots,k$. Each edge $e \in E$ is associated to a latency function $\ell_e()$, positive, differentiable, and strictly increasing on the flow traversing it.

Nonatomic Flows

There are infinitely many users, each routing his/her infinitesimally small amount of the total flow r_i from a given source s_i to a destination vertex t_i in graph $G(V,E)$. A flow f is an assignment of jobs f_e on each edge $e \in E$. The cost of the injected flow f_e (satisfying the standard constraints of the corresponding network-flow problem) that traverses edge $e \in E$ equals; $c_e(f_e) = f_e \times \ell_e(f_e)$. It is assumed that on each edge e the cost is convex with respect to the injected flow f_e. The overall system's cost is the sum $\sum_{e\in E} f_e \times \ell_e(f_e)$ of all edge costs in G. Let $f_{\mathcal{P}}$ the amount of flow traversing the $s_i - t_i$ path $\mathcal{P}$. The latency $\ell_{\mathcal{P}}(f)$ of $s_i - t_i$ path $\mathcal{P}$ is the sum $\sum_{e\in\mathcal{P}} \ell_e(f_e)$ of latencies per edge $e \in \mathcal{P}$. The cost $C_{\mathcal{P}}(f)$ of $s_i - t_i$ path $\mathcal{P}$ equals the flow $f_{\mathcal{P}}$ traversing it multiplied by path latency $\ell_{\mathcal{P}}(f)$. That is, $C_{\mathcal{P}}(f) = f_{\mathcal{P}} \times \sum_{e\in\mathcal{P}} \ell_e(f_e)$. In a Nash equilibrium, all $s_i - t_i$ paths traversed by nonatomic users in part i have a common latency, which is at most the latency of any untraversed

$s_i - t_i$ path. More formally, for any part i and any pair $\mathcal{P}_1$, $\mathcal{P}_2$ of $s_i - t_i$ paths, if $f_{\mathcal{P}_1} > 0$ then $\ell_{\mathcal{P}_1}(f) \leq \ell_{\mathcal{P}_2}(f)$. By the convexity of edge costs, the Nash equilibrium is unique and computable in polynomial time given a floating-point precision. Also computable is the unique *Optimum* assignment O of flow, assigning flow o_e on each $e \in E$ and minimizing the overall cost $\sum_{e \in E} o_e \ell_e(o_e)$. However, not all optimally traversed $s_i - t_i$ paths experience the same latency. In particular, users traversing paths with high latency have incentive to reroute toward more speedy paths. Therefore, the optimal assignment is unstable on selfish behavior.

A Leader dictates a *weak* Stackelberg strategy if on each commodity $i = 1, \ldots, k$ controls a fixed α portion of flow $r_i, \alpha \in [0, 1]$. A *strong* Stackelberg strategy is more flexible, since Leader may control $\alpha_i r_i$ flow in commodity i such that $\sum_{i=1}^{k} \alpha_i = \alpha$. Let a Leader dictating flow s_e on edge $e \in E$. The a posteriori latency $\tilde{\ell}_e(n_e)$ of edge e, with respect to the induced flow n_e by the selfish users, equals $\tilde{\ell}_e(n_e) = \ell_e(n_e + s_e)$. In the a posteriori Nash equilibrium, all $s_i - t_i$ paths traversed by the free selfish users in commodity i have a common latency, which is at most the latency of any selfishly untraversed path, and its cost is $\sum_{e \in E} (n_e + s_e) \times \tilde{\ell}_e(n_e)$.

Atomic Splittable Flows

There is a finite number of atomic users $1, \ldots, k$. Each user i is responsible for routing a non-negligible flow-amount r_i from a given source s_i to a destination vertex t_i in graph G. In turn, each flow-amount r_i consists of infinitesimally small jobs.

Let flow f assigning jobs f_e on each edge $e \in E$. Each edge flow f_e is the sum of partial flows $f_e^1, \ldots, f_e^k$ injected by the corresponding users $1, \ldots, k$. That is, $f_e = f_e^1 + \cdots + f_e^k$. As in the model above, the latency on a given $s_i - t_i$ path $\mathcal{P}$ is the sum $\sum_{e \in \mathcal{P}} \ell_e(f_e)$ of latencies per edge $e \in \mathcal{P}$. Let $f_{\mathcal{P}}^i$ be the flow that user i ships through an $s_i - t_i$ path $\mathcal{P}$. The cost of user i on a given $s_i - t_i$ path $\mathcal{P}$ is analogous to her path flow $f_{\mathcal{P}}^i$ routed via $\mathcal{P}$ times the total path latency $\sum_{e \in \mathcal{P}} \ell_e(f_e)$. That is, the path cost equals $f_{\mathcal{P}}^i \times \sum_{e \in \mathcal{P}} \ell_e(f_e)$. The overall cost $C_i(f)$ of user i is the sum of the corresponding path costs of all $s_i - t_i$ paths.

In a Nash equilibrium no user i can improve his cost $C_i(f)$ by rerouting, given that any user $j \neq i$ keeps his routing fixed. Since each atomic user minimizes its cost, if the game consists of only one user, then the cost of the Nash equilibrium coincides to the optimal one.

In a Stackelberg game, a distinguished atomic Leader player controls flow r_0 and plays first assigning flow s_e on edge $e \in E$. The a posteriori latency $\tilde{\ell}_e(x)$ of edge e on induced flow x equals $\tilde{\ell}_e(x) = \ell_e(x + s_e)$. Intuitively, after Leader's move, the induced selfish play of the k atomic users is equivalent to atomic splittable flows on a graph where each initial edge latency ℓ_e has been mapped to $\tilde{\ell}_e$. In game parlance, each atomic user $i \in \{1, \ldots, k\}$, having *fixed* Leader's strategy, computes his *best reply* against all other atomic users $\{1, \ldots, k\} \backslash \{i\}$. If n_e is the induced Nash flow on edge e, this yields total cost $\sum_{e \in E} (n_e + s_e) \times \tilde{\ell}_e(n_e)$.

Atomic Unsplittable Flows

The users are finite $1, \ldots, k$ and user i is allowed to send his non-negligible job r_i only on a *single* path. Despite this restriction, all definitions given in atomic splittable model remain the same.

Key Results

Let us see first the case of atomic splittable flows, on parallel M/M/1 links with different speeds connecting a given source-destination pair of vertices.

Theorem 1 (Korilis, Lazar, Orda [6]) *The Leader can enforce in polynomial time the network optimum if his/her controls flow* r_0 *exceeding a critical value* $\underline{r}^0$.

In the sequel, we focus on nonatomic flows on $s-t$ graphs with parallel links. In [6] primarily were studied cases that Leader's flow cannot induce network's optimum and was shown that an optimal Stackelberg strategy is easy to compute. In this vain, if $s-t$ parallel link instances are restricted to ones with linear latencies of equal slope, then an optimal strategy is easy [4].

Theorem 2 (Kaporis, Spirakis [4]) *The optimal Leader strategy can be computed in polynomial time on any instance (G, r, α) where G is an $s-t$ graph with parallel links and linear latencies of equal slope.*

Another positive result is that the optimal strategy can be approximated within $(1+\epsilon)$ in polynomial time, given that link latencies are polynomials with nonnegative coefficients.

Theorem 3 (Kumar, Marathe [8]) *There is a fully polynomial approximate Stackelberg scheme that runs in poly $\left(m, \frac{1}{\epsilon}\right)$ time and outputs a strategy with cost $(1+\epsilon)$ within the optimum strategy.*

For parallel link $s-t$ graphs with arbitrary latencies more can be achieved: in polynomial time a "threshold" value α_G is computed, sufficient for the Leader's portion to induce the optimum. The complexity of computing optimal strategies changes in a dramatic way around the critical value α_G from "hard" to "easy" (G, r, α) Stackelberg scheduling instances. Call α_G as the *Price of Optimum* for graph G.

Theorem 4 (Kaporis, Spirakis [4]) *On an input $s-t$ parallel link graph G with arbitrary strictly increasing latencies, the minimum portion α_G sufficient for a Leader to induce the optimum, as well as his/her optimal strategy, can be computed in polynomial time.*

As a conclusion, the Price of Optimum α_G essentially captures the hardness of instances (G, r, α). Since, for Stackelberg scheduling instances $(G, r, \alpha \geq \alpha_G)$, the optimal Leader strategy yields PoA $= 1$ and it is computed as hard as in P, while for $(G, r, \alpha < \alpha_G)$ the optimal strategy yields PoA < 1 and it is as easy as *NP* [10].

The results above are limited to parallel links connecting a given $s-t$ pair of vertices. Is it possible to efficiently compute the Price of Optimum for nonatomic flows on arbitrary graphs? This is not trivial to settle. Not only because it relies on computing an optimal Stackelberg strategy, which is hard to tackle [10], but also because Proposition B.3.1 in [11] ruled out previously known performance guarantees for Stackelberg strategies on general nets.

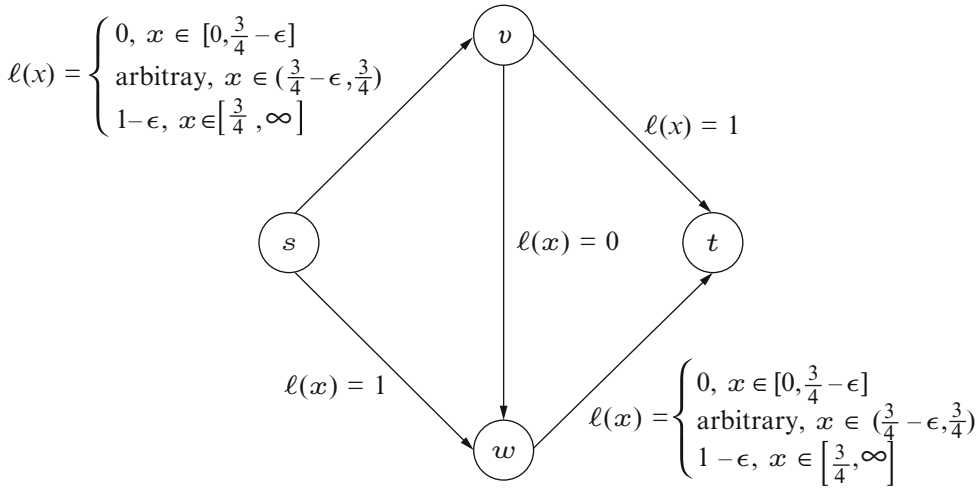
The central result of this lemma is presented below and completely resolves this question (extending Theorem 4).

Theorem 5 (Kaporis, Spirakis [4]) *On arbitrary $s-t$ graphs G with arbitrary latencies, the minimum portion α_G sufficient for a Leader to induce the optimum, as well as her optimal strategy, can be computed in polynomial time.*

Example

Consider the optimum assignment O of flow r that wishes to travel from source vertex s to sink t. O assigns flow o_e incurring latency $\ell_e(o_e)$ per edge $e \in G$. Let $\mathcal{P}_{s\to t}$ the set of all $s-t$ paths. The *shortest paths* in $\mathcal{P}_{s\to t}$ with respect to costs $\ell_e(o_e)$ per edge $e \in G$ can be computed in polynomial time. That is, the paths that given flow assignment O achieve path latency: $\min_{P\in\mathcal{P}_{s\to t}} \left(\sum_{e\in P} \ell_e(o_e)\right)$, i.e., minimize their path latency. It is crucial to observe that if we want the *induced* Nash assignment by the Stackelberg strategy to attain the optimum cost, then these shortest paths are *the only choice* for selfish users that are eager to travel from s to t. Furthermore, the uniqueness of the optimum assignment O determines the minimum part of flow which can be selfishly scheduled on these shortest paths. Observe that any flow assigned by O on a non-shortest $s-t$ path has incentive to opt for a shortest one. Then a Stackelberg strategy *must* freeze the flow on all non-shortest $s-t$ paths.

In particular, the idea sketched above achieves coordination ratio 1 on the graph in Fig. 1. On this graph Roughgarden proved that $\frac{1}{\alpha}\times$ (optimum



Stackelberg Games: The Price of Optimum, Fig. 1 A bad example for Stackelberg routing

cost) guarantee is *not* possible for general (s,t)-networks, Appendix B.3 in [11]. The optimal edge flows are ($r = 1$):

$$o_{s\to v} = \tfrac{3}{4} - \epsilon, o_{s\to w} = \tfrac{1}{4} + \epsilon, o_{v\to w} = \tfrac{1}{2} - 2\epsilon,$$
$$o_{v\to t} = \tfrac{1}{4} + \epsilon, o_{w\to t} = \tfrac{3}{4} - \epsilon$$

The shortest path $P_0 \in \mathcal{P}$ with respect to the optimum O is $P_0 = s \to v \to w \to t$ (see [11] pp. 143, 5th-3th lines before the end) and its flow is $f_{P_0} = \frac{1}{2} - 2\epsilon$. The non-shortest paths are $P_1 = s \to v \to t$ and $P_2 = s \to w \to t$ with corresponding optimal flows: $f_{P_1} = \frac{1}{4} + \epsilon$ and $f_{P_2} = \frac{1}{4} + \epsilon$. Thus, the Price of Optimum is

$$f_{P_1} + f_{P_2} = \frac{1}{2} + 2\epsilon = r - f_{P_0}$$

Applications

Stackelberg strategies are widely applicable in networking [6], see also Section 6.7 in [12].

Open Problems

It is important to extend the above results on atomic unsplittable flows.

Cross-References

- Algorithmic Mechanism Design
- Best Response Algorithms for Selfish Routing
- Facility Location
- Non-approximability of Bimatrix Nash Equilibria
- Price of Anarchy
- Selfish Unsplittable Flows: Algorithms for Pure Equilibria

Recommended Reading

1. Birman K (1997) Building secure and reliable network applications. Manning, Greenwich
2. Douligeris C, Mazumdar R (2006) Multilevel flow control of queues. In: Johns Hopkins conference on information sciences, Baltimore, 22–24 Mar 1989
3. Economides A, Silvester, J (1990) Priority load sharing: an approach using Stackelberg games. In: 28th annual Allerton conference on communications, control and computing, Monticello
4. Kaporis AC, Spirakis PG (2009) The price of optimum in Stackelberg games on arbitrary single commodity networks and latency functions. Theor Comput Sci **410**(8–10):745–755
5. Karakostas G, Kolliopoulos SG (2009) Stackelberg strategies for selfish routing in general multicommodity networks. Algorithmica **53**(1):132–153

6. Korilis YA, Lazar AA, Orda A (1997) Achieving network optima using stackelberg routing strategies. IEEE/ACM Trans Netw 5(1):161–173
7. Koutsoupias E, Papadimitriou CH (2009) Worst-case equilibria. Comput Sci Rev **3**(2):65–69
8. Kumar VSA, Marathe MV (2002) Improved results for Stackelberg scheduling strategies. In: 29th international colloquium, automata, languages and programming, Málaga. LNCS. Springer, pp 776–787
9. Roughgarden T (2001) Designing networks for selfish users is hard. In: 42nd IEEE annual symposium of foundations of computer science, Las Vegas, pp 472–481
10. Roughgarden T (2004) Stackelberg scheduling strategies. SIAM J Comput **33**(2):332–350
11. Roughgarden T 2002 Selfish routing. Dissertation, Cornell University. http://theory.stanford.edu/~tim/
12. Roughgarden T (2005) Selfish routing and the price of anarchy. MIT, Cambridge (2005)
13. Roughgarden T, Tardos É (2002) How bad is selfish routing? J ACM **49**(2):236–259
14. Swamy C (2007) The effectiveness of Stackelberg strategies and tolls for network congestion games. In: ACM-SIAM symposium on discrete algorithms, Philadelphia
15. von Stackelberg H (1934) Marktform und Gleichgewicht. Springer, Vienna

Staged Assembly

Andrew Winslow
Department of Computer Science, Tufts University, Medford, MA, USA

Keywords

Context-free grammars; DNA computing; Jigsaw; Natural computing; Polyomino; Shape decomposition

Years and Authors of Summarized Original Work

2008; Demaine, Demaine, Fekete, Ishaque, Rafalin, Schweller, Souvaine
2013; Demaine, Eisenstat, Ishaque, Winslow
2013; Winslow

Problem Definition

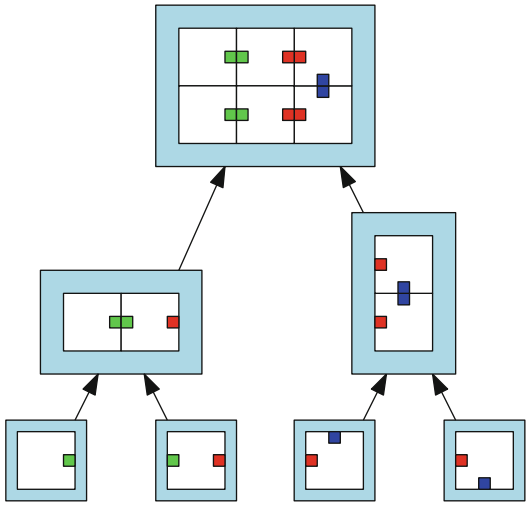
Algorithmic self-assembly is concerned with hands-off assembly of complex structures by mixing collections of simple particles that aggregate according to local rules. Staged self-assembly utilizes sequences of mixings to reduce the number of particle types used. The standard model of staged self-assembly builds on the abstract Tile Assembly Model (aTAM) of Winfree [7], where each particle is a non-rotatable unit square *tile* with a labeled *glue* on each side. Tiles attach to other tiles edgewise via glues of the same label, forming polyomino-shaped aggregates called *assemblies*.

In the simplest model, a pair of assemblies (of which single tiles are a special case) can attach via a single matching glue. In a more general model, a pair of assemblies can attach if they share a total of $\tau \in \mathbb{N}$ glues. The parameter τ is called the *temperature* of the system.

The self-assembly process is carried out by combining an infinite number of copies of a collection of *reagent assemblies* in a *bin*, where they attach in every possible way. The subset of the resulting assemblies that cannot attach to any other assemblies define the *product assemblies* of the mixing, i.e., the set of assemblies that remain once the assembly process is complete. A system consisting of a single bin with single-tile reagent assemblies is a *hierarchical* [2], *two-handed* [1], or *polyomino* [5] self-assembly system.

In a *staged self-assembly system* [3], the products of one bin can be used as the reagents of other bin (see Fig. 1). The directed acyclic graph describing the relationship mixings is called the *mix graph* of the system. An initial set of mixings each have a single tile as the only product assembly and no reagent assemblies.

Objectives In general, the goal is to design a system with a mixing containing a single product assembly of a desired polyomino shape while minimizing the size of some aspect of the system. Several aspects are considered, including the number of distinct tiles (*tile complexity*), number of edges of the mix graph (*mix graph complexity*),



Staged Assembly, Fig. 1 A staged self-assembly system. Each bin (*blue box*) contains the product assemblies of the bin. The reagent assemblies of a bin are the products of other bins (*incoming arrows*)

width of the mix graph (*bin complexity*), height of the mix graph (*stage complexity*), and temperature of the system. The computational complexity of finding an optimal system for an input shape under some measure of system complexity is also considered. In some cases, the desired polyomino shape also has each cell labeled, and the goal is to construct a given labeled shape using labeled tiles.

Problem 1 (Smallest Staged Self-Assembly System)

INPUT*: A labeled polyomino P.*
OUTPUT*: A staged self-assembly system containing a bin with a single product assembly with labeled shape P that is minimum in some measure.*

Key Results

We describe the results by increasing generality of the shapes assembled.

Lines

In the most constrained case, the input polyomino is an unlabeled $1 \times n$ polyomino (a *line*). Lines can be assembled by $\tau = 1$ systems using $O(1)$ tile types, $O(1)$ bins, and $O(\log n)$ stages and mix graph edges. The idea is to repeatedly double the length of a line assembly as proved by Demaine et al. [3].

Demaine, Eisenstat, Ishaque, and Winslow [4] prove that the case of labeled lines is roughly equivalent to the problem of finding the smallest context-free grammar that is a single string consisting of the labels of the line read from left to right. In particular, any context-free grammar $\mathcal{G}$ with $|\mathcal{G}|$ rules and deriving a single string σ can be converted into a $\tau = 1$ staged self-assembly system $\mathcal{S}$ assembling a line with left-to-right label string σ, where the number of edges in the mix graph of $\mathcal{S}$ is $O(|\mathcal{G}|)$. The complexity of the smallest staged self-assembly system where the input polyomino is a labeled line and the system has an upper limit on the number of glue types appearing on the tiles was proven to be NP-hard [4].

Squares

Demaine et al. [3] prove that unlabeled $n \times n$ squares are possible with a $\tau = 1$ staged systems containing $O(1)$ tile types and bins, $O(\log n)$ stages, and thus a mix graph with $O(\log n)$ stages. The system uses an idea similar to that for assembling lines but in two steps: first assemble $n \times 1$ columns and then combine them to form $n \times 2$, then $n \times 4$, etc., rectangles. This construction uses a *jigsaw* technique to ensure attaching rectangles cannot assemble askew.

Demaine et al. also prove that unlabeled $n \times n$ squares can be assembled using $\tau = 2$ staged systems using $O(1)$ tile types, $O(\sqrt{\log n})$ bins, and $O(\log \log n)$ stages. The approach is to *simulate* known $\tau = 2$ single-bin systems that efficiently assemble squares by constructing *macrotiles*: large assemblies that simulate the behavior of distinct tile types by encoding glue types in geometry on their surfaces. Such macrotiles allow staged systems to trade off tile types for stages by replacing many distinct tile types with an initial sequence of stages that assemble macrotile versions of the tiles.

General Shapes

For general shapes, several different results highlight the trade-offs in complexity enabled by staged assembly. Demaine et al. [3] prove that any unlabeled shape can be assembled by a $\tau = 1$ staged system using $O(1)$ tile types, $O(\log n)$ bins, and a number of stages proportional to the diameter of the dual grid graph of the shape. If the shape is monotone, then a similar system with $O(n)$ bins and $O(\log n)$ stages (increased bins but decreased stages) exists.

If the system is permitted to assemble a scaled version of the input shape, then the system of Soloveichik and Winfree [6] can be simulated with macrotiles, resulting in a $\tau = 2$ staged system with $O(1)$ tile types, $O(K/\log K)$ bins, and $O(\log \log K)$ stages, where K is the Kolmogorov complexity of the shape. For labeled shapes, Winslow [8] proves that any polyomino context-free grammar $\mathcal{G}$ (a generalization of context-free grammars to two dimensions) with $|\mathcal{G}|$ rules deriving a single labeled polyomino P can be converted into a staged system $\mathcal{S}$ assembling a scaled version of P consisting of labeled macrotiles where the number of edges in the mix graph of $\mathcal{S}$ is $O(|\mathcal{G}|)$.

Applications

The theory of algorithmic self-assembly is rooted in the design of nanoscale particle systems, particularly DNA-based systems. For staged self-assembly in particular, the capability of assembling complex shapes using only $O(1)$ tile types is highly desirable in practice, as engineering many tile types with desired glues is often far more challenging than carrying out a sequence of mixings.

Open Problems

The complexity of the smallest staged self-assembly problem where the number of glue types used is unconstrained remains open, both for the case of lines and general shapes. For lines, the problem is known to be in NP and when the number of glues is constrained is NP-complete (both proved in [4]). For general shapes, the problem is only known to be in PSPACE (proved in [9]) and NP-hard when the number of glues is constrained, following from the special case of lines. The complexity of verifying that a staged assembly system produces a given shape also remains open and is only known to lie in PSPACE.

Cross-References

- Combinatorial Optimization and Verification in Self-Assembly
- Experimental Implementation of Tile Assembly
- Patterned Self-Assembly Tile Set Synthesis
- Self-Assembly at Temperature 1
- Self-Assembly of Squares and Scaled Shapes

Recommended Reading

1. Cannon S, Demaine ED, Demaine ML, Eisenstat S, Patitz MJ, Schweller RT, Summers SM, Winslow A (2013) Two hands are better than one (up to constant factors): self-assembly in the 2HAM vs. aTAM. In: STACS 2013, Kiel, LIPIcs, vol 20. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp 172–184
2. Chen H, Doty D (2012) Parallelism and time in hierarchical self-assembly. In: Proceedings of the 23rd annual ACM-SIAM symposium on discrete algorithms (SODA), Kyoto, pp 1163–1182
3. Demaine ED, Demaine ML, Fekete SP, Ishaque M, Rafalin E, Schweller RT, Souvaine DL (2008) Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. Nat Comput 7(3):347–370
4. Demaine ED, Eisenstat S, Ishaque M, Winslow A (2013) One-dimensional staged self-assembly. Nat Comput 12(2):247–258
5. Luhrs C (2010) Polyomino-safe DNA self-assembly via block replacement. Nat Comput 9(1):97–109
6. Soloveichik D, Winfree E (2007) Complexity of self-assembled shapes. SIAM J Comput 36(6):1544–1569
7. Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, Caltech
8. Winslow A (2013a) Staged self-assembly and polyomino context-free grammars. In: Soloveichik D, Yurke B (eds) DNA 19, Tempe
9. Winslow A (2013b) Staged self-assembly and polyomino context-free grammars. PhD thesis, Tufts University

Statistical Multiple Alignment

István Miklós
Department of Plant Taxonomy and Ecology, Eötvös Loránd University, Budapest, Hungary

Keywords

Multiple HMM; Statistical alignment; Stochastic modeling of insertions and deletions; Time-continuous Markov models

Years and Authors of Summarized Original Work

2003; Hein, Jensen, Pedersen

Problem Definition

The three main types of mutations modifying biological sequences are insertions, deletions, and substitutions. The simplest model involving these three types of mutations is the so-called Thorne-Kishino-Felsenstein model [16]. In this model, the characters of a sequence evolve independently. Each character in the sequence can be substituted with another character according to a prescribed reversible time-continuous Markov model on the possible characters. Insertion-deletions are modeled as a birth-death process. Insertions can happen at the beginning of the sequence, at the end of the sequence, and between any two characters. It is possible to insert a character into the empty sequence. The time span between two insertions is exponentially distributed with parameter λ, and this parameter does not depend on the context of the position. The newborn character is drawn from the equilibrium distribution of the substitution process. Each character is deleted after an exponentially distributed waiting time with parameter μ, and its two positions where insertions can happen are joined.

The multiple statistical alignment problem is to calculate the likelihood of a set of sequences, namely, what is the probability of observing a set of sequences, given all the necessary parameters that describe the evolution of sequences. Hein, Jensen, and Pedersen were the first who gave an algorithm to calculate this probability [5]. Their algorithm has $O(5^n L^n)$ running time, where n is the number of sequences, and L is the geometric mean of the sequences. The running time has been improved to $O(2^n L^n)$ by Lunter et al. [9].

Notations

Substitutions

A time-continuous Markov model for a substitution process on an alphabet Σ is given by a $k \times k$ rate matrix Q, with constraints

$$q_{i,j} \geq 0 \qquad \forall i \neq æ \tag{1}$$

$$\sum_i q_{i,j} = 0 \qquad \forall i \tag{2}$$

where k is the size of the alphabet. The probability that a character a_i will be character a_j after time t can be calculated with the exponentiation of the rate matrix:

$$P_t(a_j|a_i) = p_{i,j} \qquad \text{where} \tag{3}$$

$$P = e^{Qt} \tag{4}$$

The exponentiated matrix can be easily calculated if the rate matrix is diagonalized, namely, if $Q = W\Lambda W^{-1}$, where Λ is a diagonal matrix, then

$$e^{Qt} = We^{\Lambda t}W^{-1} \tag{5}$$

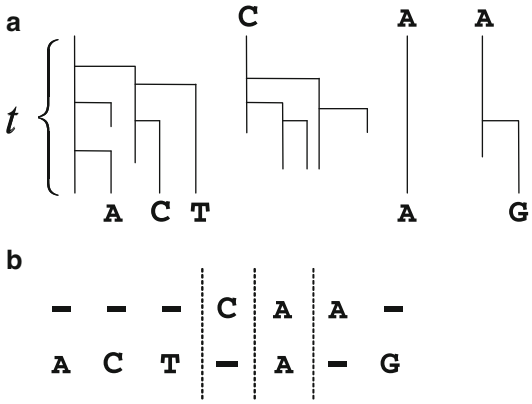
$e^{\Lambda t}$ can be easily calculated, since it is a diagonal matrix containing $e^{\lambda_i t}$ in the ith position of the diagonal.

Insertions and Deletions

A Galton-Watson tree is a rooted, edge-weighted binary tree that describes a birth-death process for a time span t. The process starts at the root of the tree, and a split represents a birth. Edge weights represent times, and leaves having a distance from the root smaller than t represent death events. Leaves being t time fare from the root are the individuals that live at time point t.

Insertion-deletion events transforming one sequence into another can be described with Galton-Watson forests: births represent insertions, and deaths represent deletions. Each character of the ancestral sequence has a tree, and there is an additional tree at the beginning of the sequence associated to an imaginary character. This imaginary character cannot die. Roots of the trees are the characters of the ancestral sequence, and each character of the descendant sequence is a leaf of one of the trees, being t time fare from the root. There might be additional leaves that are not associated with characters of the descendant sequences; these are the died out lineages. The forest is aligned such that edges do not cross each other while the characters of the two sequences keep their original order. Each Galton-Watson forest indicates an alignment of the two sequences; see Fig. 1. Given a birth and death process, the probability density of a Galton-Watson tree can be calculated easily. Assuming independence, the probability of a Galton-Watson forest is the product of the probabilities of its trees. The probability of an alignment is the integral of the probabilities of the forests that represent it. Due to independence, it is enough to tell the probability of alignment patterns that might arise as an image of a Galton-Watson tree (see Fig. 1b); the probability of an alignment is the product of the probabilities of its patterns.

In the Thorne-Kishino-Felsenstein model (TKF91 model) [16], both the birth and the death processes are Poisson processes with parameters λ and μ, respectively. The probability of the possible patterns can be found on Fig. 2.



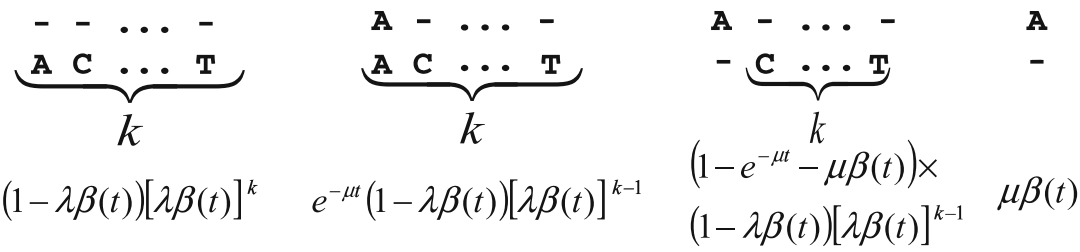
Statistical Multiple Alignment, Fig. 1 (**a**) A Galton-Watson forest representing insertion-deletion events. The first tree starts with an immortal element that is responsible to the insertions at the beginning of the sequence. (**b**) The alignment indicated by the Galton-Watson forest above. Each tree makes a pattern of the alignment; patterns are separated with *dashed lines*

Evolutionary Trees

An evolutionary tree is a leaf-labeled, edge-weighted, rooted binary tree. Labels are the species related by the evolutionary tree, and weights are evolutionary distances. It might happen that the evolutionary changes had different speed at different lineages, and hence the tree is not necessary ultrametric, namely, the root not necessary has the same distance to all leaves. The nodes of an evolutionary tree can be partially ordered such that two nodes are comparable if there is a path from the root to any of the leaves containing the two nodes in question, and in this case the smaller node is the one that is closer to the root on the path. Each node v of an evolutionary tree indicates a subtree that contains v and all the nodes that are greater than v. Hereafter we consider only these subtrees.

Given a set S of l-long sequences over alphabet Σ, a substitution model M on Σ and an evolutionary tree T are labeled by the sequences. The likelihood of the tree is the probability of observing the sequences at the leaves of the tree, given that the substitution process starts at the root of the tree with the equilibrium distribution. This likelihood is denoted by $P(S|T, M)$. The substitution likelihood problem is to calculate the likelihood of the tree.

Let Σ be a finite alphabet and let $S_1 = s_{1,1}s_{1,2}\ldots s_{1,L_1}$, $S_2 = s_{2,1}s_{2,2}\ldots s_{2,L_2}$, $\ldots$ $S_n = s_{n,1}s_{n,2}\ldots s_{n,L_n}$ be sequences over this alphabet. Let a TKF91 model $TKF91$ be given with its parameters: substitution model M, insertion rate λ, and deletion rate μ. Let T be an evolutionary tree labeled by S_1, S_2, $\ldots$ S_n. The multiple statistical alignment problem is to calculate the likelihood of the tree, $P(S_1, S_2, \ldots S_n|T, TKF91)$, given that



Statistical Multiple Alignment, Fig. 2 The probabilities of alignment patterns. From *left* to *right*: k insertions at the beginning of the alignment, a match followed by $k-1$ insertions, a deletion followed by k insertions, and a deletion not followed by insertions. $\beta = \frac{1-e^{(\lambda-\mu)t}}{\mu-\lambda e^{(\lambda-\mu)t}}$

the TKF91 process starts at the root with the equilibrium distribution.

Multiple Hidden Markov Models

It will turn out that the TKF91 model can be transformed to a multiple Hidden Markov Model; therefore we formally define it here. A multiple Hidden Markov Model (multiple HMM) is a directed graph with distinguished start and end states, (the in degree of the start and the out degree of the end state are both 0), together with the following described transition and emission distributions. Each vertex has a transition distribution over its out edges. The vertexes can be divided into two classes, the emitting and silent states. Each emitting state emits one-one random character to a prescribed set of sequences; it is possible that a state emits only one character to one sequence. For each state, an emission distribution over the alphabet and the set of sequences gives the probabilities which characters will be emitted to which sequences. The Markov process is a random walk from the start to the end, following the transition distribution on the out edges. When the walk is in an emitting state, characters are emitted according to the emission distribution of the state. The process is hidden since the observer sees only the emitted sequences, and the observer does not observe which character is emitted by which state, even the observer does not see which characters are co-emitted. The multiple HMM problem is to calculate the emission probability of a set of sequences for a multiple HMM. This probability can be calculated with the forward algorithm that has $O(V^2L^n)$ running time, where V is the number of emitting states in the multiple HMM, L is the geometric mean of the sequences, and n is the number of sequences [3].

Key Results

Substitutions have been modeled with time-continuous Markov models since the late 1960s [8], and an efficient algorithm for likelihood calculations was published in 1980 [4]. The running time of this efficient algorithm grows linearly both with the number of sequences and with the length of the sequences being analyzed, and it grows squarely with the size of the alphabet. The algorithm belongs to the class of dynamic programming algorithms. For each character, subtree, and position x, the algorithm calculates what would be the likelihood of the characters in position x in the sequences belonging to the subtree if the substitution process started in the root of the subtree with the given character. These probabilities are called conditional likelihoods. It is easy to show that

$$L_p(\alpha, x) = \left(\sum_{\alpha_1} P_{t_1}(\alpha_1|\alpha) L_{d_1}(\alpha_1, x)\right) \left(\sum_{\alpha_2} P_{t_2}(\alpha_2|\alpha) L_{d_2}(\alpha_2, x)\right) \quad (6)$$

where d_1 and d_2 are the descendant nodes of the parent node p and t_1 and t_2 are the length of the edges connecting p with d_1 and d_2, respectively. The likelihood of the tree can be calculated from the conditional likelihoods of the tree. Recall that

$P(S|T, M)$ is the likelihood of observing a set of sequences S on the leaves of an evolutionary tree T under the substitution model M:

$$P(S|T, M) = \prod_x \sum_\alpha L_{root}(\alpha, x)\pi_\alpha \quad (7)$$

Thorne, Kishino, and Felsenstein gave an $O(nm)$ running time algorithm for calculating the likelihood of an n-long and an m-long sequence under their model [16]. It was not clear for long time how to extend this algorithm to more than two sequences. In 2001, several researchers [7,12] realized that the TKF91 model for two sequences is equivalent with a pair Hidden Markov Model (pair HMM) in the sense that the transition and emission probabilities of the pair HMM can be parameterized with λ, μ and the transition and equilibrium probabilities of the substitution model; moreover there is a bijection between the paths emitting the two sequences and alignments such that the probability of a path in the pair HMM equals to the probability of the corresponding alignment of the two sequences. Hence the likelihood of two sequences can be calculated with the forward algorithm of the pair HMM.

After this discovery, it was relatively easy to develop an algorithm for multiple statistical alignment [5]. The key observation is that a multiple HMM can be created as a composition of pair HMMs along the evolutionary tree. This technique was already known in the speech recognition literature [14], and was also rediscovered by Ian Holmes [6], who named this technique as transducer composition. The number of states in the so-created multiple HMM is $O(5^{\frac{n}{2}})$, where n is the number of leaves of the tree. The emission probabilities are the substitution likelihoods on the tree, which can be efficiently calculated as shown above. The running time of the forward algorithm is $5^n L^n$, where L is the geometric mean of the sequence lengths.

Lunter et al. [9] introduced an algorithm that does not need a multiple HMM description of the TKF91 model to calculate the likelihood of a tree. Using a logical sieve algorithm, they were able to reduce the running time to $O(2^n L^n)$. They called their algorithm the "one-state recursion" since their dynamic programming algorithm does not need different state of a multiple HMM to calculate the likelihood correctly.

Applications

Since the running time of the best known algorithm for multiple statistical alignment grows exponentially with the number of sequences, on its own it is not useful in practice. However, Lunter et al. also showed that there is a one-state recursion to calculate the likelihood of the tree given an alignment [10]. The running time of this algorithm grows only linearly with both the alignment length and the number of sequences. Since the number of states in a multiple HMM that can emit the same multiple alignment column might grow exponentially, this version of the one-state recursion is a significant improvement. The one-state recursion for multiple alignments is used in a Bayesian Markov chain Monte Carlo where the state space is the Descartes product of the possible multiple alignments and evolutionary trees. The one-state recursion provides an efficient likelihood calculation for a point in the state space [11].

Csűrös and Miklós introduced a model for gene content evolution that is equivalent with the multiple statistical alignment problem for alphabet size 1 [2]. They gave a polynomial running time algorithm that calculates the likelihood of the tree. The running time is $O(n + hL^2)$, where n is the number of sequences, h is the height of the evolutionary tree, and L is the sum of the sequence lengths.

Thorne, Kishino, and Felsenstein also introduced a fragment model, also called the TKF92 model, in which multiple insertions and deletions are allowed [17]. The birth process is still a Poisson process, but instead of single characters, fragments of characters are inserted with a geometrically distributed length. The fragments are unbreakable, and the death process is going on the fragments. The TKF92 model for a pair of sequences also can be described into a pair HMM and the TKF92 model on a tree can be

transformed to a multiple HMM. Such multiple HMM is used in the StatAlign software package [13]. The software package has been extended to predict the common structure of sequences (e.g., slowly quickly evolving regions, RNA secondary structures) by combining this multiple HMM with other stochastic models describing the structure of sequences [1, 15].

Open Problems

It is conjectured that the multiple statistical alignment problem cannot be solved in polynomial time for any nontrivial alphabet size. One also can ask what the most likely multiple alignment is or, equivalently, what the most probable path in the multiple HMM is that emits the given sequences. For a set of sequences, a TKF91 model, and an evolutionary tree, the decision problem "Is there a multiple alignment that is more probable than p" is conjectured to be NP-complete.

It is conjectured that there is no one-state recursion for the TKF92 model.

Recommended Reading

1. Arunapuram P, Edvardsson I, Golden M, Anderson JW, Novák Á, Sükösd Z, Hein J (2013) StatAlign 2.0: combining statistical alignment with RNA secondary structure prediction. Bioinformatics 29(5):654–655
2. Csűrös M, Miklós I (2006) A probabilistic model for gene content evolution with duplication, loss, and horizontal transfer. In: Proceedings of RECOMB2006. Lecture notes in bioinformatics, Springer Verlag, vol 3909, pp 206–220
3. Durbin R, Eddy S, Krogh A, Mitchison G (1998). Biological sequence analysis. Cambridge University Press, Cambridge
4. Felsenstein J (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. J Mol Evol 17:368–376
5. Hein JJ, Jensen JL, Pedersen CNS (2003) Recursions for statistical multiple alignment. PNAS 100:14960–14965
6. Holmes I (2003) Using guide trees to construct multiple-sequence evolutionary hmms. Bioinformatics 19:i147–i157
7. Holmes I, Bruno WJ (2001) Evolutionary HMMs: a Bayesian approach to multiple alignment. Bioinformatics 17(9):803–820
8. Jukes TH, Cantor CR (1969) Evolution of protein molecules. In: Munro HN (ed) Mammalian protein metabolism. Academic, New York, pp 21–132
9. Lunter GA, Miklós I, Song YS, Hein J (2003) An efficient algorithm for statistical multiple alignment on arbitrary phylogenetic trees. J Comput Biol 10(6):869–889
10. Lunter GA, Miklós I, Drummond AJ, Jensen JL, Hein JJ (2003) Bayesian phylogenetic inference under a statistical indel model. In: Proceedings of WABI2003. Lecture notes in bioinformatics, Springer Verlag, vol 2812, pp 228–244
11. Lunter GA, Miklós I, Drummond AJ, Jensen JL, Hein JJ (2005) Bayesian coestimation of phylogeny and sequence alignment. BMC Bioinformatics 6:83
12. Metzler D, Fleißner R, Wakolbringer A, von Haeseler A (2001) Assessing variability by joint sampling of alignments and mutation rates. J Mol Evol 53:660–669
13. Novák A, Miklós I, Lyngsø R, Hein J (2008) StatAlign: an extendable software package for joint bayesian estimation of alignments and evolutionary trees. Bioinformatics 24(20):2403–2404
14. Pereira F, Riley M (1997) Speech recognition by composition of weighted finite automata. In: Finite-state language processing. MIT, Cambridge, pp 149–173
15. Satija R, Novák A, Miklós I, Lyngsø R, Hein J (2009) BigFoot: Bayesian alignment and phylogenetic footprinting with MCMC. BMC Evol Biol 9:217
16. Thorne JL, Kishino H, Felsenstein J (1991) An evolutionary model for maximum likelihood alignment of DNA sequences. J Mol Evol 33:114–124
17. Thorne JL, Kishino H, Felsenstein J (1992) Inching toward reality: an improved likelihood model of sequence evolution. J Mol Evol 34:3–16

Statistical Query Learning

Vitaly Feldman
IBM Research – Almaden, San Jose, CA, USA

Keywords

Classification noise; Noise-tolerant learning; PAC learning; SQ dimension; Statistical query

Years and Authors of Summarized Original Work

1998; Kearns

Problem Definition

The problem deals with learning to classify from random labeled examples in Valiant's PAC model [30]. In the *random classification noise* model of Angluin and Laird [1], the label of each example given to the learning algorithm is flipped randomly and independently with some fixed probability η called the *noise rate*. Robustness to such benign form of noise is an important goal in the design of learning algorithms. Kearns defined a powerful and convenient framework for constructing noise-tolerant algorithms based on *statistical queries*. Statistical query (SQ) learning is a natural restriction of PAC learning that models algorithms that use statistical properties of a data set, rather than individual examples. Kearns demonstrated that any learning algorithm that is based on statistical queries can be automatically converted to a learning algorithm in the presence of random classification noise of arbitrary rate smaller than the information-theoretic barrier of $1/2$. This result was used to give the first noise-tolerant algorithm for a number of important learning problems. In fact, virtually all known noise-tolerant PAC algorithms were either obtained from SQ algorithms or can be easily cast into the SQ model.

In subsequent work, the model of Kearns has been extended to other settings and found a number of additional applications in machine learning and theoretical computer science.

Definitions and Notation

Let $\mathcal{C}$ be a class of $\{-1, +1\}$-valued functions (also called *concepts*) over an input space X. In the basic PAC model, a learning algorithm is given examples of an unknown function f from $\mathcal{C}$ on points randomly chosen from some unknown distribution $\mathcal{D}$ over X and should produce a hypothesis h that approximates f. More formally, an *example oracle* $\mathrm{EX}(f, \mathcal{D})$ is an oracle that upon being invoked returns an example $\langle x, f(x)\rangle$, where x is chosen randomly with respect to $\mathcal{D}$, independently of any previous examples. A learning algorithm for $\mathcal{C}$ is an algorithm that for every $\epsilon > 0$, $\delta > 0$, $f \in \mathcal{C}$, and distribution $\mathcal{D}$ over X, given ϵ, δ, and access to $\mathrm{EX}(f, \mathcal{D})$ outputs, with probability at least $1 - \delta$, a hypothesis h that ϵ-approximates f with respect to $\mathcal{D}$ (i.e., $\mathbf{Pr}_{\mathcal{D}}[f(x) \neq h(x)] \leq \epsilon$). Efficient learning algorithms are algorithms that run in time polynomial in $1/\epsilon$, $1/\delta$ and the size of the learning problem s. The size of a learning problem is determined by the description length of f under some fixed representation scheme for functions in $\mathcal{C}$ and the description length of an element in X (often proportional to the dimension n of the input space).

A number of variants of this basic framework are commonly considered. The basic PAC model is also referred to as *distribution-independent* learning to distinguish it from *distribution-specific* PAC learning in which the learning algorithm is required to learn with respect to a single distribution $\mathcal{D}$ known in advance. A *weak* learning algorithm is a learning algorithm that can produce a hypothesis whose error on the target concept is noticeably less than $1/2$ (and not necessarily any $\epsilon > 0$). More precisely, a weak learning algorithm produces a hypothesis h such that $\mathbf{Pr}_{\mathcal{D}}[f(x) \neq h(x)] \leq 1/2 - 1/p(s)$ for some fixed polynomial p. The basic PAC model is often referred to as *strong* learning in this context.

In the random classification noise model $\mathrm{EX}(f, \mathcal{D})$ is replaced by a faulty oracle $\mathrm{EX}^{\eta}(f, \mathcal{D})$, where η is the noise rate. When queried, this oracle returns a noisy example $\langle x, b\rangle$ where $b = f(x)$ with probability $1 - \eta$ and $\neg f(x)$ with probability η independently of previous examples. When η approaches $1/2$ the label of the corrupted example approaches the result of a random coin flip, and therefore, the running time of learning algorithms in this model is allowed to depend on $\frac{1}{1-2\eta}$ (the dependence must be polynomial for the algorithm to be considered efficient). For simplicity, one usually assumes that η is known to the learning algorithm. This assumption can be removed using a simple technique due to Laird [26].

To formalize the idea of learning from statistical properties of a large number of examples, Kearns introduced a new oracle $\mathrm{STAT}(f, \mathcal{D})$ that replaces $\mathrm{EX}(f, \mathcal{D})$. The oracle $\mathrm{STAT}(f, \mathcal{D})$ takes

as input a *statistical query* (SQ) of the form (χ, τ), where χ is a $\{-1, +1\}$-valued function on labeled examples and $\tau \in [0, 1]$ is the *tolerance* parameter. Given such a query, the oracle responds with an estimate v of $\mathbf{Pr}_{\mathcal{D}}[\chi(x, f(x)) = 1]$ that is accurate to within an additive $\pm\tau$.

Note that the oracle does not guarantee anything else on the value v beyond $|v - \mathbf{Pr}_{\mathcal{D}}[\chi(x, f(x)) = 1]| \leq \tau$ and an SQ learning algorithm needs to work with any possible implementation of the oracle. Yang proposed a stronger, *honest* version of the oracle which to a call with function χ returns the value of $\chi(x, f(x))$, where x is chosen randomly and independently according to $\mathcal{D}$ [32]. This version was shown to be equivalent to the original model up to polynomial factors [17].

Chernoff bounds easily imply that $\mathrm{STAT}(f, \mathcal{D})$ can, with high probability, be simulated using $\mathrm{EX}(f, \mathcal{D})$ by estimating $\mathbf{Pr}_{\mathcal{D}}[\chi(x, f(x)) = 1]$ on $O(\tau^{-2})$ examples. Therefore, the SQ model is a restriction of the PAC model. Efficient SQ algorithms allow only efficiently evaluatable χ's and impose an inverse polynomial lower bound on the tolerance parameter over all oracle calls. Kearns also observes that in order to simulate all the statistical queries used by an algorithm, one does not necessarily need new examples for each estimation. Instead, assuming that the set of possible queries of the algorithm has Vapnik-Chervonenkis dimension d, all its statistical queries can be simulated using $\tilde{O}(d\,\tau^{-2}(1-2\eta)^{-2}\log(1/\delta))$ examples [24].

Key Results

Statistical Queries and Noise-Tolerance

The main result given by Kearns is a way to simulate statistical queries using noisy examples.

Lemma 1 ([24]) *Let (χ, τ) be a statistical query such that χ can be evaluated on any input in time T and let $\mathrm{EX}^{\eta}(f, \mathcal{D})$ be a noisy oracle. The value $\mathbf{Pr}_{\mathcal{D}}[\chi(x, f(x)) = 1]$ can, with probability at least $1-\delta$, be estimated within τ using $O(\tau^{-2}(1-2\eta)^{-2}\log(1/\delta))$ examples from $\mathrm{EX}^{\eta}(f, \mathcal{D})$ and time $O(\tau^{-2}(1-2\eta)^{-2}\log(1/\delta)\cdot T)$.*

This simulation is based on estimating several probabilities using examples from the noisy oracle and then offsetting the effect of noise. The lemma implies that any efficient SQ algorithm for a concept class $\mathcal{C}$ can be converted to an efficient learning algorithm for $\mathcal{C}$ tolerating random classification noise of any rate $\eta < 1/2$.

Theorem 1 ([24]) *Let $\mathcal{C}$ be a concept class efficiently* PAC *learnable from statistical queries. Then $\mathcal{C}$ is efficiently* PAC *learnable in the presence of random classification noise of rate η for any $\eta < 1/2$.*

Balcan and Feldman describe more general conditions on noise under which a specific SQ algorithm can be simulated in the presence of noise [3].

Statistical Query Algorithms

Kearns showed that, despite the major restriction on the way an SQ algorithm accesses the examples, many PAC learning algorithms known at the time can be modified to use statistical queries instead of random examples [24]. Examples of learning algorithms for which he described an SQ analogue and thereby obtained a noise-tolerant learning algorithm include:

- Learning decision trees of constant rank.
- *Attribute-efficient* algorithms for learning conjunctions.
- Learning axis-aligned rectangles over $\mathbb{R}^n$.
- Learning AC^0 (constant-depth unbounded fan-in) Boolean circuits over $\{0,1\}^n$ with respect to the uniform distribution in quasipolynomial time.

Subsequent works have provided numerous additional examples of algorithms used in theory and practice of machine learning that can either be implemented using statistical queries or can be replaced by an alternative SQ-based algorithm of similar complexity. For example, the Perceptron algorithm and learning of linear threshold functions [6, 12], boosting [2], attribute-efficient learning via the Winnow algorithm (cf. [16]), k-means clustering [5] and convex optimization-

based methods [20]. We note that many learning algorithms rely only on evaluations of functions on random examples and therefore can be seen as using access to the honest statistical query oracle. In such cases the SQ implementation follows immediately from the equivalence of the Kearns' SQ oracle and the honest one [17].

The only known example of a technique for which there is no SQ analogue is Gaussian elimination for solving linear equations over a finite field. This technique can be used to learn parity functions that are not learnable using SQs (as we discuss below). As a result, with the exception of the parity learning problem, known bounds on the complexity of learning from random examples are, up to polynomial factors, the same as known bound for learning with statistical queries.

Statistical Query Dimension

The restricted way in which SQ algorithms use examples makes it simpler to understand the limitations of efficient learning in this model. A long-standing open problem in learning theory is learning of the concept class of all parity functions over $\{0,1\}^n$ with noise (a parity function is a XOR of some subset of n Boolean inputs). Kearns has demonstrated that parities cannot be efficiently learned using statistical queries even under the uniform distribution over $\{0,1\}^n$ [24]. This hardness result is unconditional in the sense that it does not rely on any unproven complexity assumptions.

The technique of Kearns was generalized by Blum et al. who proved that efficient SQ learnability of a concept class $\mathcal{C}$ is characterized by a relatively simple combinatorial parameter of $\mathcal{C}$ called the *statistical query dimension* [7]. The quantity they defined, measures the maximum number of "nearly uncorrelated" functions in a concept class. (The definition and the results were simplified and strengthened in subsequent works [17, 29] and we use the improved statements here.) More formally,

Definition 1 For a concept class $\mathcal{C}$ and distribution $\mathcal{D}$, the *statistical query dimension* of $\mathcal{C}$ with respect to $\mathcal{D}$, denoted SQ-DIM$(\mathcal{C},\mathcal{D})$, is the largest number d such that $\mathcal{C}$ contains d functions $f_1, f_2, \ldots, f_d$ such that for all $i \neq j$, $|\mathbf{E}_{\mathcal{D}}[f_i f_j]| \leq \frac{1}{d}$.

Blum et al. relate the SQ dimension to learning in the SQ model as follows.

Theorem 2 ([7, 17]) *Let $\mathcal{C}$ be a concept class and $\mathcal{D}$ be a distribution such that* SQ-DIM$(\mathcal{C},\mathcal{D}) = d$.

- *If all queries are made with tolerance of at least $1/d^{1/3}$, then at least $d^{1/3}-2$ queries are required to learn $\mathcal{C}$ with error $1/2 - 1/(2d^3)$ in the SQ model.*
- *There exists an algorithm for learning $\mathcal{C}$ with respect to $\mathcal{D}$ that makes d fixed queries, each of tolerance $1/(4d)$, and finds a hypothesis with error at most $1/2 - 1/(2d)$.*

Thus SQ-DIM characterizes weak SQ learnability relative to a fixed distribution $\mathcal{D}$ up to a polynomial factor. Parity functions are uncorrelated with respect to the uniform distribution and therefore, any concept class that contains a superpolynomial number of parity functions cannot be learned by statistical queries with respect to the uniform distribution. This, for example, includes such important concept classes as *k-juntas* over $\{0,1\}^n$ (or functions that depend on at most k input variables) for $k = \omega(1)$ and *decision trees* of superconstant size.

Simon showed that (strong) PAC learning relative to a fixed distribution $\mathcal{D}$ using SQs can also be characterized by a more general and involved dimension [28]. Simpler and tighter characterizations of distribution-specific PAC learning using SQs have been demonstrated by Feldman [15] and Szörényi [29]. Feldman also extended the characterization to the agnostic learning model.

Despite characterizing the number of queries of certain tolerance, the SQ-DIM and its generalizations capture surprisingly well the computational complexity of SQ learning of most concept classes. One reason for this is that if a concept class has polynomial SQ-DIM then it can be learned by a polynomial-time algorithm with advice also referred to as a "non-uniform"

algorithm (cf. [18]). However it was shown by Feldman and Kanade that for strong PAC learning there exist artificial problems whose computational complexity is larger than their statistical query complexity [18].

Applications of these characterizations to proving lower bounds on SQ algorithms can be found in [11, 15, 19, 25]. Relationships of SQ-DIM to other notions of complexity of concept classes were investigated in [22, 27].

Applications

The ideas behind the use of statistical queries to produce noise-tolerant algorithms were adapted to learning using *membership queries* (or ability to ask for the value of the unknown function at any point) and used to give a noise-tolerant algorithm for learning DNF with respect to the uniform distribution [9, 21]. The SQ model of learning was generalized to active learning (or learning where labels are requested only for some of the points) and used to obtain new efficient noise-tolerant active learning algorithms [3].

The restricted way in which an SQ algorithm uses data implies it can be used to obtain learning algorithms with additional useful properties. Blum et al. [5] show that an SQ algorithm can be used to obtain a *differentially-private* [13] algorithm for the problem. In fact, SQ algorithms are equivalent to *local* (or *randomized-response*) differentially-private algorithms [23]. Chu et al. [10] show that SQ algorithms can be automatically parallelized on multicore architectures and give many examples of popular machine learning algorithms that can be sped up using this approach.

The SQ learning model has also been instrumental in understanding Valiant's model of evolution as learning [31]. Feldman showed that the model is equivalent to learning with a restricted form of SQs referred to as correlational SQs [14]. A correlational SQ is a query of the form $\chi(x, \ell) = g(x) \cdot \ell$ for some $g : X \to [-1, 1]$. Such queries were first studied by Ben-David et al. [4] (remarkably, before the introduction of the SQ model itself) and distribution-specific learning with such queries is equivalent to learning with (unrestricted) SQs.

Statistical query-based access can naturally be defined for any problem where the input is a set of i.i.d. samples from a distribution. Feldman et al. show that lower bounds based on SQ-DIM can be extended to this more general setting and give examples of applications [17, 20].

Open Problems

The main questions related to learning with random classification noise are still open. Is every concept class efficiently learnable in the PAC model also learnable in the presence of random classification noise? Is every concept class efficiently learnable in the presence of random classification noise of arbitrarily high rate (less than $1/2$) also efficiently learnable using statistical queries? A partial answer to this question was provided by Blum et al. who show that Gaussian elimination can be used in low dimension to obtain a class learnable with random classification noise of constant rate $\eta < 1/2$ but not learnable using SQs [8]. For both questions a central issue seems to be obtaining a better understanding of the complexity of learning parities with noise.

The complexity of learning from statistical queries remains an active area of research with many open problems. For example, there is currently an exponential gap between known lower and upper bounds on the complexity of distribution-independent SQ learning of polynomial-size DNF formulae and AC^0 circuits (cf. [27]). Several additional open problems on complexity of SQ learning can be found in [16, 19, 22].

Cross-References

- Attribute-Efficient Learning
- Learning Constant-Depth Circuits
- Learning DNF Formulas
- Learning Heavy Fourier Coefficients of Boolean Functions
- Learning with Malicious Noise
- PAC Learning

Recommended Reading

1. Angluin D, Laird P (1988) Learning from noisy examples. Mach Learn 2:343–370
2. Aslam J, Decatur S (1998) General bounds on statistical query learning and pac learning with noise via hypothesis boosting. Inf Comput 141(2):85–118
3. Balcan M-F, Feldman V (2013) Statistical active learning algorithms. In: NIPS, Lake Tahoe, pp 1295–1303
4. Ben-David S, Itai A, Kushilevitz E (1990) Learning by distances. In: Proceedings of COLT, Rochester, pp 232–245
5. Blum A, Dwork C, McSherry F, Nissim K (2005) Practical privacy: the SuLQ framework. In: Proceedings of PODS, Baltimore, pp 128–138
6. Blum A, Frieze A, Kannan R, Vempala S (1997) A polynomial time algorithm for learning noisy linear threshold functions. Algorithmica 22(1/2):35–52
7. Blum A, Furst M, Jackson J, Kearns M, Mansour Y, Rudich S (1994) Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In: Proceedings of STOC, Montréal, pp 253–262
8. Blum A, Kalai A, Wasserman H (2003) Noise-tolerant learning, the parity problem, and the statistical query model. J ACM 50(4):506–519
9. Bshouty N, Feldman V (2002) On using extended statistical queries to avoid membership queries. J Mach Learn Res 2:359–395
10. Chu C, Kim S, Lin Y, Yu Y, Bradski G, Ng A, Olukotun K (2006) Map-reduce for machine learning on multicore. In: Proceedings of NIPS, Vancouver, pp 281–288
11. Dachman-Soled D, Feldman V, Tan L-Y, Wan A, Wimmer K (2014) Approximate resilience, monotonicity, and the complexity of agnostic learning. arXiv, CoRR, abs/1405.5268
12. Dunagan J, Vempala S (2004) A simple polynomial-time rescaling algorithm for solving linear programs. In: Proceedings of STOC, Chicago, pp 315–320
13. Dwork C, McSherry F, Nissim K, Smith A (2006) Calibrating noise to sensitivity in private data analysis. In: TCC, New York, pp 265–284
14. Feldman V (2008) Evolvability from learning algorithms. In: Proceedings of STOC, Victoria, pp 619–628
15. Feldman V (2012) A complete characterization of statistical query learning with applications to evolvability. J Comput Syst Sci 78(5):1444–1459
16. Feldman V (2014) Open problem: the statistical query complexity of learning sparse halfspaces. In: COLT, Barcelona, pp 1283–1289
17. Feldman V, Grigorescu E, Reyzin L, Vempala S, Xiao Y (2013) Statistical algorithms and a lower bound for planted clique. In: STOC, Palo Alto. ACM, pp 655–664
18. Feldman V, Kanade V (2012) Computational bounds on statistical query learning. In: COLT, Edinburgh, pp 16.1–16.22
19. Feldman V, Lee H, Servedio R (2011) Lower bounds and hardness amplification for learning shallow monotone formulas. In: COLT, Budapest, vol 19, pp 273–292
20. Feldman V, Perkins W, Vempala S (2013) On the complexity of random satisfiability problems with planted solutions. In: CoRR, abs/1311.4821
21. Jackson J, Shamir E, Shwartzman C (1997) Learning with queries corrupted by classification noise. In: Proceedings of the fifth Israel symposium on the theory of computing systems, Ramat-Gan, pp 45–53
22. Kallweit M, Simon H (2011) A close look to margin complexity and related parameters. In: COLT, Budapest, pp 437–456
23. Kasiviswanathan SP, Lee HK, Nissim K, Raskhodnikova S, Smith A (2011) What can we learn privately? SIAM J Comput 40(3):793–826
24. Kearns M (1998) Efficient noise-tolerant learning from statistical queries. J ACM 45(6): 983–1006
25. Klivans A, Sherstov A (2007) Unconditional lower bounds for learning intersections of halfspaces. Mach Learn 69(2–3):97–114
26. Laird P (1988) Learning from good and bad data. Kluwer Academic, Boston
27. Sherstov AA (2008) Halfspace matrices. Comput Complex 17(2):149–178
28. Simon H (2007) A characterization of strong learnability in the statistical query model. In: Proceedings of symposium on theoretical aspects of computer science, Aachen, pp 393–404
29. Szörényi B (2009) Characterizing statistical query learning: simplified notions and proofs. In: Proceedings of ALT, Porto, pp 186–200
30. Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142
31. Valiant LG (2009) Evolvability. J ACM 56(1):3.1–3.21. Earlier version in ECCC, 2006
32. Yang K (2005) New lower bounds for statistical query learning. J Comput Syst Sci 70(4):485–509

Statistical Timing Analysis

Sachin S. Sapatnekar
Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA

Keywords

Delay uncertainty; Electronic design automation; Principal component analysis; Process variations; Static timing analysis; Timing closure

Years and Authors of Summarized Original Work

2003; Chang, Sapatnekar
2005; Chang, Sapatnekar

Problem Definition

The timing behavior of integrated systems is strongly affected by the characteristics of transistors and wires in the system. Variations in the manufacturing process can cause drifts in these characteristics from one manufactured part to another. The traditional approach to addressing these variations was to choose a worst-case value for each process parameter, but this has become unsustainable in the face of current-day variations. Statistical timing analysis provides a computationally efficient way to translate the probability density function of the underlying process parameter spread to the distribution of circuit timing.

A key underlying structure for timing analysis is a graph $G(V, E)$ of a combinational circuit, where the vertex set V corresponds to the gates, primary inputs, and primary outputs of the circuit, and each connection between these gates corresponds to an edge in E. The delay of each gate corresponds to a probability distribution that is a function of the distributions of the underlying (possibly correlated) process parameters, and the task of combinational statistical timing analysis is to obtain the distribution of the maximum (or minimum) delay of the circuit, over all primary outputs. The extension of this problem to general edge-triggered sequential circuits is straightforward. Such circuits can be decomposed into independent combinational blocks, and the maximum (or minimum) operator acts on the delay distribution at all primary outputs of all combinational blocks of the sequential circuit.

Key Results

The framework that is used for statistical timing analysis is based on graph-based topological traversals that maintain a closed-form structure for the delay from the primary inputs of the circuit to the output of each vertex (referred to as the *arrival time*). The computation under this paradigm scales linearly with $|E|$. While it is certainly possible to perform statistical timing analysis through Monte Carlo simulations based on samples of the process parameter space, such an approach is uncompetitive compared to graph traversal algorithms. The traversal approach consists of three key steps [1, 2]:

- Translating the underlying process parameter variations to an orthogonal set of random variables
- Representing gate delay variations in terms of this orthogonal set
- Performing a topological traversal of G and computing the arrival time at each node and maximum delay of the circuit

Orthogonalizing Process Parameter Distributions

A common assumption is that the underlying process parameters, such as the transistor width W and effective length L_{eff} of devices, gate oxide thickness (T_{ox}), and device threshold voltage (V_{t}) due to random dopant fluctuations, show a Gaussian distribution. Each individual device is separately represented by such a parameter. The distributions of T_{ox} and V_{t} are largely uncorrelated across devices. In contrast, the dimension-based parameters, W and L_{eff}, show strong spatial correlations, whereby the distributions of nearby devices are strongly correlated, and this correlation falls off as a function of distance.

The existence of correlations can significantly complicate the task of statistical timing analysis, since all pairwise combinations of random variables must be considered during the optimization, potentially leading to quadratic complexity in $|V|$. To overcome this, an initial principal component analysis (PCA) [7] step is carried out that orthogonalizes the underlying Gaussians, enabling linear-time analysis. PCA is a one-time operation for a given process (which is used for numerous designs). Therefore, although its worst-case com-

plexity is cubic in $|V|$, the expense is practically manageable as it is amortized over numerous designs. Furthermore, sparsity properties of the correlation matrix realistically imply that in practice, the cost of PCA scales considerably slower than this cubic rate.

For cases where the underlying process parameters may be a mix of Gaussians or non-Gaussians, it is possible to orthogonalize the Gaussian parameters using PCA and non-Gaussian parameters using independent component analysis (ICA) [4]. The approach in [8] extends the graph-based approach presented here and shows how statistical timing analysis can be performed for case where some or all process parameters are non-Gaussian.

Gate Delay Distribution

To build a model for the gate delay that captures the underlying variations in process parameters, we observe that the delay function $d = f(\mathbf{P})$, where $\mathbf{P}$ is a set of process parameters, can be approximated d linearly using a first-order Taylor expansion:

$$d = d_0 + \sum_{\forall \text{ parameters } p_i} \left[\frac{\partial f}{\partial p_i}\right]_0 \Delta p_i \quad (1)$$

where d_0 is the nominal value of d, calculated at the nominal values of parameters in the set $\mathbf{P}$; $\left[\frac{\partial f}{\partial p_i}\right]_0$ is computed at the nominal values of p_i; $\Delta p_i = p_i - \mu_{p_i}$ is a normally distributed random variable; and $\Delta p_i \sim N(0, \sigma_{p_i})$. The delay function here can be arbitrarily complex.

If all parameters in $\mathbf{P}$ can be modeled by Gaussian distributions, this approximation implies that d is a linear combination of Gaussians, which is therefore Gaussian. Its mean μ_d and variance σ_d^2 are

$$\mu_d = d_0 \quad (2)$$

$$\sigma_d^2 = \sum_{\forall i} \left[\frac{\partial f}{\partial p_i}\right]_0^2 \sigma_{p_i}^2 + 2 \sum_{\forall i \neq j} \left[\frac{\partial f}{\partial p_i}\right]_0 \left[\frac{\partial f}{\partial p_j}\right]_0 \text{cov}\left(p_i, p_j\right) \quad (3)$$

where $\text{cov}\left(p_i, p_j\right)$ is the covariance of p_i and p_j.

This approximation is valid when Δp_i has relatively small variations, in which domain the first-order Taylor expansion is adequate and the approximation is acceptable with little loss of accuracy. This is generally true of the impact of within-die variations on delay, where the process parameter variations are relatively small in comparison with the nominal values, and the function changes by a small amount under this perturbation. Hence, the delays, as functions of the process parameters, can be approximated as normal distributions when the parameter variations are assumed to be normal. Higher-order expansions based on quadratics have also be explored to cover cases where the variations are larger [6, 11].

Circuit Delay Distribution

A PCA-based approach maintains the invariant that the output arrival time at each gate is a Gaussian variable represented as

$$a_i(p_1, \ldots, p_n) = a_i^0 + \sum_{i=1}^{n} k_i p_i' + k_{n+1} p_{n+1}' \quad (4)$$

Here, the primed variables correspond to the principal components of the unprimed variables and maintain the form of the arrival time after each sum and max operation. Gate delays, as represented in Eq. 1, can be translated into a similar representation based on principal components as a one-time step during gate library characterization. Under orthogonalization, many operations become much simpler since the covariance terms disappear: for example, Eq. 3 can

be evaluated in linear time instead of quadratic time.

The task of statistical timing analysis is to translate these gate delay distributions to circuit delay probabilities while performing a topological traversal. The operations performed at each node encountered during this traversal in STA are of two types [5]:

- A gate (vertex) is being processed in STA when the arrival times of all inputs are known, at which time the candidate delay values at the output are computed using the "sum" operation that adds the delay at each input with the input-to-output pin delay.
- Once these candidate delays have been found, the "max" operation is applied to determine the maximum arrival time at the output.

Since the gate delays are Gaussian, the "sum" operation is merely an addition of Gaussians, which is well known to be a Gaussian. The computation of the max function, however, poses greater problems. The set of candidate delays are all Gaussian, so that this function must find the maximum of Gaussians. Such a maximum may be reasonably approximated using a Gaussian [3]. A detailed description of how the invariant representation is maintained under the max operation is presented in [1, 2].

The cost of this method corresponds to running a bounded number of deterministic STAs, and it is demonstrated to be accurate, given the statistics of **P**.

Applications

Statistical timing analysis has been extensively used in industry [10] and has seeded a large amount of academic research. Integrated circuit manufacturing foundries have promoted the use of statistical timing analysis by providing PCA-like information with their process parameter models, thus enabling design flows that are statistically based.

The ideas of statistical analysis have also motivated simpler and more approximate methods, used in industry today, based on on-chip variation (OCV) derating factors. In its most elementary form, OCV adds margins to each timing path to account for possible variation. More involved versions of OCV, such as advanced OCV (AOCV), capture the essence of spatial correlation by using derating factors that depend on factors such as spatial distance and logical depth of a path [9].

Experimental Results

Statistical timing analysis based on orthogonalization brings down the computational cost from quadratic to linear in the number of variables and can be applied to large circuit instances. The method is capable of considering both spatial correlations and structural correlations, i.e., correlations between paths that share gates, since such correlations are embedded into the invariant representation. This makes the approach accurate and computationally practical, as described in [1, 2, 10] and the large body of follow-on work.

URLs to Code and Data Sets

The MinnSSTA statistical static timing analyzer is available at http://www.ece.umn.edu/~sachin/software/MinnSSTA/index.html.

Recommended Reading

1. Chang H, Sapatnekar SS (2003) Statistical timing analysis considering spatial correlations using a single PERT-like traversal. In: Proceedings of the IEEE/ACM international conference on computer-aided design, San Jose, pp 621–625

2. Chang H, Sapatnekar SS (2005) Statistical timing analysis under spatial correlations. IEEE Trans Comput-Aided Des Integr Circuits Syst 24(9):1467–1482
3. Clark CE (1961) The greatest of a finite set of random variables. Oper Res 9:85–91
4. Hyvärinen A, Oja E (2000) Independent component analysis: algorithms and applications. Neural Netw 13:411–430
5. Jacobs E, Berkelaar MRCM (2000) Gate sizing using a statistical delay model. In: Proceedings of design and test in Europe, Paris, pp 283–290
6. Li X, Le J, Gopalakrishnan P, Pileggi LT (2007) Asymptotic probability extraction for nonnormal performance distributions. IEEE Trans Comput-Aided Des Integr Circuits Syst 26(1):16–37
7. Morrison DF (1976) Multivariate statistical methods. McGraw-Hill, New York
8. Singh J, Sapatnekar SS (2008) A scalable statistical static timing analyzer incorporating correlated non-Gaussian and Gaussian parameter variations. IEEE Trans Comput-Aided Des Integr Circuits Syst 27(1):160–173
9. Synopsys Inc (2009) PrimeTime® Advanced OCV Technology. www.synopsys.com/Tools/Implementation/SignOff/CapsuleModule/PrimeTime_AdvancedOCV_WP.pdf
10. Visweswariah C, Ravindran K, Kalafala K, Walker SG, Narayan S, Beece DK, Piaget J, Venkateswaran N, Hemmett JG (2006) First-order incremental block-based statistical timing analysis. IEEE Trans Comput-Aided Des Integr Circuits Syst 25(10):2170–2180
11. Zhan Y, Strojwas AJ, Li X, Pileggi LT, Newmark D, Sharma M (2005) Correlation-aware statistical timing analysis with non-Gaussian delay distributions. In: Proceedings of the ACM/IEEE design automation conference, San Jose, pp 77–82

Steiner Forest

Guido Schäfer
Institute for Mathematics and Computer Science, Technical University of Berlin, Berlin, Germany

Keywords

Requirement join; *R*-join

Years and Authors of Summarized Original Work

1995; Agrawal, Klein, Ravi

Problem Definition

The *Steiner forest problem* is a fundamental problem in network design. Informally, the goal is to establish connections between pairs of vertices in a given network at minimum cost. The problem generalizes the well-known *Steiner tree problem*. As an example, assume that a telecommunication company receives communication requests from their customers. Each customer asks for a connection between two vertices in a given network. The company's goal is to build a minimum cost network infrastructure such that all communication requests are satisfied.

Formal Definition and Notation

More formally, an instance $I = (G, c, R)$ of the Steiner forest problem is given by an undirected graph $G = (V, E)$ with vertex set V and edge set E, a non-negative cost function $c: E \to \mathbb{Q}^+$, and a set of vertex pairs $R = \{(s_1, t_1), \ldots, (s_k, t_k)\} \subseteq V \times V$. The pairs in R are called *terminal pairs*. A feasible solution is a subset $F \subseteq E$ of the edges of G such that for every terminal pair $(s_i, t_i) \in R$ there is a path between s_i and t_i in the subgraph $G[F]$ induced by F. Let the cost $c(F)$ of F be defined as the total cost of all edges in F, i.e., $c(F) = \sum_{e \in F} c(e)$. The goal is to find a feasible solution F of minimum cost $c(F)$. It is easy to see that there exists an optimum solution which is a forest.

The Steiner forest problem may alternatively be defined by a set of *terminal groups* $R = \{g_1, \ldots, g_k\}$ with $g_i \subseteq V$ instead of terminal pairs. The objective is to compute a minimum cost subgraph such that all terminals belonging to the same group are connected. This definition is equivalent to the one given above.

Related Problems

A special case of the Steiner forest problem is the *Steiner tree problem* (see also the entry ▸ Steiner Trees). Here, all terminal pairs share a common root vertex $r \in V$, i.e., $r \in \{s_i, t_i\}$ for all terminal pairs $(s_i, t_i) \in R$. In other words, the problem consists of a set of terminal vertices $R \subseteq V$ and a root vertex $r \in V$ and the goal is to connect the

terminals in R to r in the cheapest possible way. A minimum cost solution is a tree.

The *generalized Steiner network problem* (see the entry ▸ Generalized Steiner Network), also known as the *survivable network design problem*, is a generalization of the Steiner forest problem. Here, a *connectivity requirement* function $r: V \times V \to \mathbb{N}$ specifies the number of edge disjoint paths that need to be established between every pair of vertices. That is, the goal is to find a minimum cost multi-subset H of the edges of G (H may contain the same edge several times) such that for every pair of vertices $(x, y) \in V$ there are $r(x, y)$ edge disjoint paths from x to y in $G[H]$. The goal is to find a set H of minimum cost. Clearly, if $r(x, y) \in \{0, 1\}$ for all $(x, y) \in V \times V$, this problem reduces to the Steiner forest problem.

Key Results

Agrawal, Klein and Ravi [1, 2] give an approximation algorithm for the Steiner forest problem that achieves an approximation ratio of 2. More precisely, the authors prove the following theorem.

Theorem 1 *There exists an approximation algorithm that for every instance $I = (G, c, R)$ of the Steiner forest problem, computes a feasible forest F such that*

$$c(F) \leq \left(2 - \frac{1}{k}\right) \cdot \mathsf{OPT}(I),$$

where k is the number of terminal pairs in R and $\mathsf{OPT}(I)$ is the cost of an optimal Steiner forest for I.

Related Work

The Steiner tree problem is NP-hard [10] and APX-complete [4, 8]. The current best lower bound on the achievable approximation ratio for the Steiner tree problem is 1.0074 [21]. Goemans and Williamson [11] generalized the results obtained by Agrawal, Klein and Ravi to a larger class of connectivity problems, which they term *constrained forest problems*. For the Steiner forest problem, their algorithm achieves the same approximation ratio of $(2 - 1/k)$. The algorithms of Agrawal, Klein and Ravi [2] and Goemans and Williamson [11] are both based on the classical *undirected cut formulation* for the Steiner forest problem [3]. The integrality gap of this relaxation is known to be $(2 - 1/k)$ and the results in [2, 11] are therefore tight. Jain [15] presents a 2-approximation algorithm for the generalized Steiner network problem.

Primal-Dual Algorithm

The main ideas of the algorithm by Agrawal, Klein and Ravi [2] are sketched below; subsequently, AKR is used to refer to this algorithm. The description given here differs from the one in [2]; the interested reader is referred to [2] for more details.

The algorithm is based on the following integer programming formulation for the Steiner forest problem. Let $I = (G, c, R)$ be an instance of the Steiner forest problem. Associate an indicator variable $x_e \in \{0, 1\}$ with every edge $e \in E$. The value of x_e is 1 if e is part of the forest F and 0 otherwise. A subset $S \subseteq V$ of the vertices is called a *Steiner cut* if there exists at least one terminal pair $(s_i, t_i) \in R$ such that $|\{s_i, t_i\} \cap S| = 1$; S is said to *separate* terminal pair (s_i, t_i). Let $\mathcal{S}$ be the set of all Steiner cuts. For a subset $S \subseteq V$, define $\delta(S)$ as the the set of all edges in E that have exactly one endpoint in S. Given a Steiner cut $S \in \mathcal{S}$, any feasible solution F of I must contain at least one edge that *crosses* the cut S, i.e., $\sum_{e \in \delta(S)} x_e \geq 1$. This gives rise to the following *undirected cut formulation*:

$$\text{minimize} \quad \sum_{e \in E} c(e) x_e \qquad \text{(IP)}$$

$$\text{subject to} \quad \sum_{e \in \delta(S)} x_e \geq 1 \quad \forall S \in \mathcal{S} \qquad (1)$$

$$x_e \in \{0, 1\} \quad \forall e \in E. \qquad (2)$$

The dual of the linear programming relaxation of (IP) has a variable y_S for every Steiner cut $S \in \mathcal{S}$. There is a constraint for every edge $e \in E$ that requires that the total dual assigned to sets $S \in \mathcal{S}$ that contain exactly one endpoint of e is at most the cost $c(e)$ of the edge:

$$\text{maximize} \quad \sum_{S \in \mathcal{S}} y_S \qquad \text{(D)}$$

$$\text{subject to} \quad \sum_{S \in \mathcal{S}: e \in \delta(S)} y_S \leq c(e) \quad \forall e \in E \quad (3)$$

$$y_S \geq 0 \quad \forall S \in \mathcal{S}. \quad (4)$$

Algorithm AKR is based on the *primal-dual schema* (see, e.g., [22]). That is, the algorithm constructs both a feasible primal solution for (IP) and a feasible dual solution for (D). The algorithm starts with an infeasible primal solution and reduces its degree of infeasibility as it progresses. At the same time, it creates a feasible dual packing of subsets of large total value by raising dual variables of Steiner cuts.

One can think of an execution of AKR as a process over time. Let x^τ and y^τ, respectively, be the primal incidence vector and feasible dual solution at time τ. Initially, let $x_e^0 = 0$ for all $e \in E$ and $y_S^0 = 0$ for all $S \in \mathcal{S}$. Let F^τ denote the forest corresponding to the set of edges with $x_e^\tau = 1$. A tree T in F^τ is called *active* at time τ if it contains a terminal that is separated from its mate; otherwise it is *inactive*. Intuitively, AKR grows trees in F^τ that are active. At the same time, the algorithm raises dual values of Steiner cuts that correspond to active trees. If two active trees collide, they are merged. The process terminates if all trees are inactive and thus there are no unconnected terminal pairs. The interplay of the primal (growing trees) and the dual process (raising duals) is somewhat subtle and outlined next.

An edge $e \in E$ is *tight* if the corresponding constraint (3) holds with equality; a path is tight if all its edges are tight. Let H^τ be the subgraph of G that is induced by the tight edges for dual y^τ. The connected components of H^τ induce a partition $\mathcal{C}^\tau$ on the vertex set V. Let $\mathcal{S}^\tau$ be the set of all Steiner cuts contained in $\mathcal{C}^\tau$, i.e., $\mathcal{S}^\tau = \mathcal{C}^\tau \cap \mathcal{S}$. AKR raises the dual values y_S for all sets $S \in \mathcal{S}^\tau$ uniformly at all times $\tau \geq 0$. Note that y^τ is dual feasible. The algorithm maintains the invariant that F^τ is a subgraph of H^τ at all times. Consider the event that a path P between two trees T_1 and T_2 of F^τ becomes tight. The missing edges of P are then added to F^τ and the process continues. Eventually, all trees in F^τ are inactive and the process halts.

Applications

The computation of (approximate) solutions for the Steiner forest problem has various applications both in theory and practice; only a few recent developments are mentioned here.

Algorithms for more complex network design problems often rely on good approximation algorithms for the Steiner forest problem. For example, the recent approximation algorithms [6, 9, 12] for the *multi-commodity rent-or-buy problem* (MRoB) are based on the random sampling framework by Gupta et al. [12, 13]. The framework uses a Steiner forest approximation algorithm that satisfies a certain *strictness* property as a subroutine. Fleischer et al. [9] show that AKR meets this strictness requirement, which leads to the current best 5-approximation algorithm for MRoB. The strictness property also plays a crucial role in the boosted sampling framework by Gupta et al. [14] for two-stage stochastic optimization problems with recourse.

Online versions of Steiner tree and forest problems have been studied by by Awerbuch et al. [5] and Berman and Coulston [7]. In the area of algorithmic game theory, the development of *group-strategyproof cost sharing mechanisms* for network design problems such as the Steiner tree problem has recently received a lot of attention; see e.g., [16, 17, 19, 20]. An adaptation of AKR yields such a cost sharing mechanism for the Steiner forest problem [18].

Cross-References

- Generalized Steiner Network
- Steiner Trees

Recommended Reading

The interested reader is referred in particular to the articles [2, 11] for a more detailed description of primal-dual approximation algorithms for general network design problems.

1. Agrawal A, Klein P, Ravi R (1991) When trees collide: an approximation algorithm for the generalized Steiner problem on networks. In: Proceedings of the 23rd annual ACM symposium on theory of computing. Association for Computing Machinery, New York, pp 134–144
2. Agrawal A, Klein P, Ravi R (1995) When trees collide: an approximation algorithm for the generalized Steiner problem in networks. SIAM J Comput 24(3):445–456
3. Aneja YP (1980) An integer linear programming approach to the Steiner problem in graphs. Networks 10(2):167–178
4. Arora S, Lund C, Motwani R, Sudan M, Szegedy M (1998) Proof verification and the hardness of approximation problems. J ACM 45(3):501–555
5. Awerbuch B, Azar Y, Bartal Y (1996) On-line generalized Steiner problem. In: Proceedings of the 7th annual ACM-SIAM symposium on discrete algorithms, 2005. Society for Industrial and Applied Mathematics, Philadelphia, pp 68–74
6. Becchetti L, Könemann J, Leonardi S, Pál M (2005) Sharing the cost more efficiently: improved approximation for multicommodity rent-or-buy. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, pp 375–384
7. Berman P, Coulston C (1997) On-line algorithms for Steiner tree problems. In: Proceedings of the 29th annual ACM symposium on theory of computing. Association for Computing Machinery, New York, pp 344–353
8. Bern M, Plassmann P (1989) The Steiner problem with edge lengths 1 and 2. Inf Process Lett 32(4):171–176
9. Fleischer L, Könemann J, Leonardi S, Schäfer G (2006) Simple cost sharing schemes for multicommodity rent-or-buy and stochastic Steiner tree. In: Proceedings of the 38th annual ACM symposium on theory of computing. Association for Computing Machinery, New York, pp 663–670
10. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco
11. Goemans MX, Williamson DP (1995) A general approximation technique for constrained forest problems. SIAM J Comput 24(2):296–317
12. Gupta A, Kumar A, Pál M, Roughgarden T (2003) Approximation via cost-sharing: a simple approximation algorithm for the multicommodity rent-or-buy problem. In: Proceedings of the 44th annual IEEE symposium on foundations of computer science. IEEE Computer Society, Washington, pp 606–617
13. Gupta A, Kumar A, Pál M, Roughgarden T (2007) Approximation via cost-sharing: simpler and better approximation algorithms for network design. J ACM 54(3):Article 11
14. Gupta A, Pál M, Ravi R, Sinha A (2004) Boosted sampling: approximation algorithms for stochastic optimization. In: Proceedings of the 36th annual ACM symposium on theory of computing. Association for Computing Machinery, New York, pp 417–426
15. Jain K (2001) A factor 2 approximation for the generalized Steiner network problem. Combinatorica 21(1):39–60
16. Jain K, Vazirani VV (2001) Applications of approximation algorithms to cooperative games. In: Proceedings of the 33rd annual ACM symposium on theory of computing. Association for Computing Machinery, New York, pp 364–372
17. Kent K, Skorin-Kapov D (1996) Population monotonic cost allocation on mst's. In: Proceedings of the 6th international conference on operational research. Croatian Operational Research Society, Zagreb, pp 43–48
18. Könemann J, Leonardi S, Schäfer G (2005) A group-strategyproof mechanism for Steiner forests. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, pp 612–619
19. Megiddo N (1978) Cost allocation for Steiner trees. Networks 8(1):1–6
20. Moulin H, Shenker S (2001) Strategyproof sharing of submodular costs: budget balance versus efficiency. Econ Theory 18(3):511–533
21. Thimm M (2003) On the approximability of the Steiner tree problem. Theor Comput Sci 295(1–3):387–402
22. Vazirani VV (2001) Approximation algorithms. Springer, Berlin

Steiner Trees

Weili Wu[1,2,3] and Yaocun Huang[3]
[1]College of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi Province, China
[2]Department of Computer Science, California State University, Los Angeles, CA, USA
[3]Department of Computer Science, The University of Texas at Dallas, Richardson, TX, USA

Keywords

Approximation algorithm design

Years and Authors of Summarized Original Work

2006; Du, Graham, Pardalos, Wan, Wu, Zhao

Definition

Given a set of points, called *terminals*, in a metric space, the problem is to find the shortest tree interconnecting all terminals. There are three important metric spaces for Steiner trees, the Euclidean plane, the rectilinear plane, and the edge-weighted network. The Steiner tree problems in those metric spaces are called the *Euclidean Steiner tree (EST)*, the *rectilinear Steiner tree (RST)*, and the *network Steiner tree (NST)*, respectively. EST and RST have been found to have polynomial-time approximation schemes (PTAS) by using adaptive partition. However, for NST, there exists a positive number r such that computing r-approximation is NP-hard. So far, the best performance ratio of polynomial-time approximation for NST is achieved by k-restricted Steiner trees. However, in practice, the iterated 1-Steiner tree is used very often. Actually, the iterated 1-Steiner was proposed as a candidate of good approximation for Steiner minimum trees a long time ago. It has a very good record in computer experiments, but no correct analysis was given showing the iterated 1-Steiner tree having a performance ratio better than that of the minimum spanning tree until the recent work by Du et al. [9]. There is minimal difference in construction of the 3-restricted Steiner tree and the iterated 1-Steiner tree, which makes a big difference in analysis of those two types of trees. Why does the difficulty of analysis make so much difference? This will be explained in this article.

History and Background

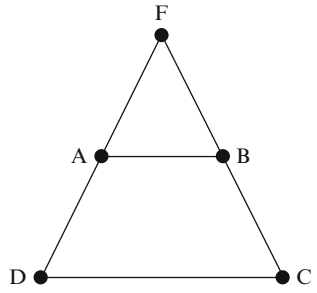
The Steiner tree problem was proposed by Gauss in 1835 as a generalization of the Fermat problem. Given three points A, B, and C in the Euclidean plane, Fermat studied the problem of finding a point S to minimize $|SA| + |SB| + |SC|$. He determined that when all three inner angles of triangle ABC are less than 120°, the optimal S should be at the position that $\angle ASB = \angle BSC = \angle CSA = 120°$.

The generalization of the Fermat problem has two directions:

1. Given n points in the Euclidean plane, find a point S to minimize the total distance from S to n given points. This is still called the Fermat problem.
2. Given n points in the Euclidean plane, find the shortest network interconnecting all given points.

Gauss found the second generalization through communication with Schumacher. On March 19, 1836, Schumacher wrote a letter to Gauss and mentioned a paradox about Fermat's problem: Consider a convex quadrilateral $ABCD$. It is known that the solution of Fermat's problem for four points A, B, C, and D is the intersection E of diagonals AC and BD. Suppose extending DA and CB can obtain an intersection F. Now, move A and B to F. Then E will also be moved to F. However, when the angle at F is less than 120°, the point F cannot be the solution of Fermat's problem for three given points F, D, and C. What happens? (Fig. 1.)

On March 21, 1836, Gauss wrote a letter replying to Schumacher in which he explained that the mistake of Schumacher's paradox occurs at the place where Fermat's problem for four points A, B, C, and D is changed to Fermat's problem for three points F, C, and D. When A and B are identical to F, the total distance from E to four points A, B, C, and D equals $2|EF| + |EC| + |ED|$, not $|EF| + |EC| + |ED|$. Thus,



Steiner Trees, Fig. 1 Convex quadrilateral $ABCD$, Fermat's problem

the point E may not be the solution of Fermat's problem for F, C, and D. More importantly, Gauss proposed a new problem. He said that it is more interesting to find the shortest network rather than a point. Gauss also presented several possible connections of the shortest network for four given points.

It was unfortunate that Gauss' letter was not seen by researchers of Steiner trees at an earlier stage. Especially, R. Courant and H. Robbins who in their popular book *What is mathematics?* (published in 1941) [6] called Gauss' problem the Steiner tree so that "Steiner tree" became a popular name for the problem.

The Steiner tree became an important research topic in mathematics and computer science due to its applications in telecommunication and computer networks. Starting with Gilbert and Pollak's work published in 1968, many publications on Steiner trees have been generated to solve various problems concerning it.

One well-known problem is the *Gilbert-Pollak conjecture* on the Steiner ratio, which is the least ratio of lengths between the Steiner minimum tree and the minimum spanning tree on the same set of given points. Gilbert and Pollak in 1968 conjectured that the Steiner ratio in the Euclidean plane is $\sqrt{3}/2$ which is achieved by three vertices of an equilateral triangle. A great deal of research effort has been put into the conjecture and it was finally proved by Du and Hwang [7].

Another important problem is called the *better approximation*. For a long time no approximation could be proved to have a performance ratio smaller than the inverse of the Steiner ratio. Zelikovsky [14] made the first breakthrough. He found a polynomial-time 11/6-approximation for NST which beats 1/2, the inverse of the Steiner ratio in the edge-weighted network. Later, Berman and Ramaiye [2] gave a polynomial-time 92/72-approximation for RST, and Du, Zhang, and Feng [8] closed the story by showing that in any metric space, there exists a polynomial-time approximation with a performance ratio better than the inverse of the Steiner ratio provided that for any set of a fixed number of points, the Steiner minimum tree is polynomial-time computable.

All the above better approximations came from the family of k-restricted Steiner trees. By improving some detail of construction, the constant performance ratio was decreasing, but the improvements were also becoming smaller. In 1996, Arora [1] made significant progress for EST and RST. He showed the existence of PTAS for EST and RST. Therefore, the theoretical researchers now pay more attention to NST. Bern and [3] showed that NST is MAX SNP-complete. This means that there exists a positive number r; computing the r-approximation for NST is NP-hard. The best-known performance for NST was given by Robin and Zelikovsky [12]. They also gave a very simple analysis to a well-known heuristic, the iterated 1-Steiner tree for pseudo-bipartite graphs.

Analysis of the iterated 1-Steiner tree is another long-standing open problem. Since Chang [4,5] proposed that the iterated 1-Steiner tree approximates the Steiner minimum tree in 1972, its performance has been claimed to be very good through computer experiments [10, 13], but no theoretical analysis supported this claim. Actually, both the k-restricted Steiner tree and the iterated 1-Steiner tree are obtained by greedy algorithms, but with different types of potential functions. For the iterated 1-Steiner tree, the potential function is non-submodular, but for the k-restricted Steiner tree, it is submodular; a property that holds for k-restricted Steiner trees may not hold for iterated 1-Steiner trees. Actually, the submodularity of potential function is very important in analysis of greedy approximations [11]. Du et al. [9] gave a correct analysis for the iterated 1-Steiner tree with a general technique to deal with non-submodular potential function.

Key Results

Consider input edge-weighted graph $G = (V, E)$ of NST. Assume that G is a complete graph and the edge weight satisfies the triangular inequality; otherwise, consider the complete graph on V with each edge (u, v) having a weight equal to the length of the shortest path between u and v in G. Given a set P of terminals, a *Steiner tree* is a

tree interconnecting all given terminals such that every leaf is a terminal.

In a Steiner tree, a terminal may have degree more than one. The Steiner tree can be decomposed, at those terminals with degree more than one, into smaller trees in which every terminal is a leaf. In such a decomposition, each resulting small tree is called a *full component*. The *size* of a full component is the number of terminals in it. A Steiner tree is *k-restricted* if every full component of it has a size at most k. The shortest k-restricted Steiner tree is also called the *k-restricted Steiner minimum tree*. Its length is denoted by $smt_k(P)$. Clearly, $smt_2(P)$ is the length of the minimum spanning tree on P, which is also denoted by $mst(P)$. Let $smt(P)$ denote the length of the Steiner minimum tree on P. If $smt_3(P)$ can be computed in polynomial time, then it is better than $mst(P)$ for an approximation of $smt(P)$. However, so far no polynomial-time approximation has been found for $smt_3(P)$. Therefore, Zelikovsky [14] used a greedy approximation of $smt_3(P)$ to approximate $smt(P)$. Actually, Chang [4, 5] used a similar greedy algorithm to compute an iterated 1-Steiner tree. Let $\mathcal{F}$ be a family of subgraphs of input edge-weighted graph G. For any connected subgraph H, denote by $mst(H)$ the length of the minimum spanning tree of H, and for any subgraph H, denote by $mst(H)$ the sum of $mst(H')$ for H' over all connected components of H. Define

$$gain(H) = mst(P) - mst(P : H) - mst(H),$$

where $mst(P : H)$ is the length of the minimum spanning tree interconnecting all unconnected terminals in P after every edge of H shrinks into a point.

Greedy Algorithm $H \leftarrow \emptyset$;
while P has not been interconnected by H **do** choose $F \in \mathcal{F}$ to maximize $gain(H \cup F)$;
output $mst(H)$.

When $\mathcal{F}$ consists of all full components of size at most three, this greedy algorithm gives the 3-restricted Steiner tree of Zelikovsky [14]. When $\mathcal{F}$ consists of all 3-stars and all edges where a 3-star is a tree with three leaves and a central vertex, this greedy algorithm produces the iterated 1-Steiner tree. An interesting fact pointed out by Du et al. [9] is that the function $gain(\cdot)$ is submodular over all full components of size at most three, but not submodular over all 3-stars and edges.

Let us consider a base set E and a function f from all subsets of E to real numbers. f is *submodular* if for any two subsets A, B of E,

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B).$$

For $x \in E$ and $A \subseteq E$, denote $\Delta_x f(A) = f(A \cup \{x\}) - f(A)$.

Lemma 1 *f is submodular if and only if for any $A \subset E$ and distinct $x, y \in E - A$,*

$$\Delta_x \Delta_y f(A) \leq 0. \tag{1}$$

Proof Suppose f is submodular. Set $B = A \cup \{x\}$ and $C = A \cup \{y\}$. Then $B \cup C = A \cup A \cup \{x, y\}$ and $B \cap C = A$. Therefore, one has

$$f(A \cup \{x, y\}) - f(A \cup \{x\}) - f(A \cup \{y\}) + f(A) \leq 0,$$

that is, (1) holds.

Conversely, suppose (1) holds for any $A \subset E$ and distinct $x, y \in E - A$. Consider two subsets A, B of E. If $A \subseteq B$ or $B \subseteq A$, it is trivial to have

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B).$$

Therefore, one may assume that $A \backslash B \neq \emptyset$ and $B \backslash A \neq \emptyset$. Write $A \backslash B = \{x_1, \ldots, x_k\}$ and $B \backslash A = \{y_1, \ldots, y_h\}$. Then

$$\begin{aligned} & f(A \cup B) - f(A) - f(B) + f(A \cap B) \\ &= \sum_{i=1}^{k} \sum_{j=1}^{h} \Delta_{x_i} \Delta_{y_j} f(A \cup \{x_1, \ldots, x_{i-1}\} \cup \{y_1, \ldots, y_{j-1}\}) \\ &\leq 0, \end{aligned}$$

where $\{x_1, \ldots x_{i-1}\} = \emptyset$ for $i = 1$ and $\{y_1, \ldots, y_{j-1}\} = \emptyset$ for $j = 1$. □

Lemma 2 *Define $f(H) = -mst(P : H)$. Then f is submodular over edge set E.*

Proof Note that for any two distinct edges x and y not in subgraph H,

$$\begin{aligned}
&\Delta_x \Delta f(H) \\
&\quad = -mst(P : H \cup x \cup y) + mst(P : H \cup x) \\
&\qquad + mst(P : H \cup y) - mst(P : H) \\
&\quad = (mst(P : H) - mst(P : H \cup x \cup y)) \\
&\qquad - (mst(P : H) - mst(P : H \cup x)) \\
&\qquad + (mst(P : H) - mst(P : H \cup y)).
\end{aligned}$$

Let T be a minimum spanning tree for unconnected terminals after every edge of H shrinks into a point. T contains a path P_x connecting two endpoints of x and also a path P_y connecting two endpoints of y. Let e_x (e_y) be a longest edge in P_x (P_y). Then

$$\begin{aligned}
mst(P : H) - mst(P : H \cup x) &= length(e_x), \\
mst(P : H) - mst(P : H \cup y) &= length(e_y).
\end{aligned}$$

$mst(P : H) - mst(P : H \cup x \cup y)$ can be computed as follows: Choose a longest edge $e\prime$ from $P_x \cup P_y$. Note that $T \cup x \cup y - e'$ contains a unique cycle Q. Choose a longest edge $e\prime\prime$ from $(P_x \cup P_y) \cap Q$. Then

$$mst(P : H) - mst(P : H \cup x \cup y) = length(e'')$$

Now, to show the submodularity of f, it suffices to prove

$$length(e_x) + length(e_y) \geq length(e'') \quad (2)$$

Case 1. $e_x P_x \cap P_y$ and $e_y P_x \cap P_y$. Without loss of generality, assume $length(e_x) \geq length(e_y)$. Then one may choose $e' = e_x$ so that $(P_x \cup P_y) \cap Q = P_y$. Hence, one can choose $e'' = e_y$. Therefore, the equality holds for (2).

Case 2. $e_x P_x \cap P_y$ and $e_y \in P_x \cap P_y$. Clearly, $length(e_x) \geq length(e_y)$. Hence, one may choose $e' = e_x$ so that $(P_x \cup P_y) \cap Q = P_y$. Hence, one can choose $e'' = e_y$. Therefore, the equality holds for (2).

Case 3. $e_x \in P_x \cap P_y$ and $e_y P_x \cap P_y$. Similar to Case 2.

Case 4. $e_x \in P_x \cap P_y$ and $e_y \in P_x \cap P_y$. In this case, $length(e_x) = length(e_y) = length(e')$. Hence, (2) holds. □

The following explains that the submodularity of $gain(\cdot)$ holds for a k-restricted Steiner tree.

Theorem 1 *Let ε be the set of all full components of a Steiner tree. Then $gain(\cdot)$ as a function on the power set of ε is submodular.*

Proof Note that for any $\mathcal{H} \subset \mathcal{E}$ and $x, y \in \mathcal{E} - \mathcal{H}$,

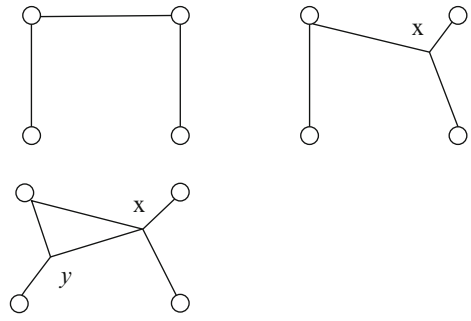
$$\Delta_x \Delta_y mst(H) = 0,$$

where $H = \cup_{z \in \mathcal{H}} z$. Thus, this theorem follows from Lemma 2.

Let $\mathcal{F}$ be the set of 3-stars and edges chosen in the greedy algorithm to produce an iterated 1-Steiner tree. Then $gain(\cdot)$ may not be submodular on $\mathcal{F}$. To see this fact, consider two 3-stars x and y in Fig. 2. Note that $gain(x \cup y) > gain(x), gain(y) \leq 0$, and $gain(\emptyset) = 0$. One has

$$gain(x \cup y) - gain(x) - gain(y) + gain(\emptyset) > 0.$$

□



Steiner Trees, Fig. 2 An example for the proof of Theorem 1

Applications

The Steiner tree problem is a classic NP-hard problem with many applications in the design of computer circuits, long-distance telephone lines, multicast routing in communication networks, etc. There exist many heuristics of the greedy type for Steiner trees in the literature. Most of them have a good performance in computer experiments, without support from theoretical analysis. The approach given in this work may apply to them.

Open Problems

It is still open whether computing the 3-restricted Steiner minimum tree is NP-hard or not. For $k \geq 4$, it is known that computing the k-restricted Steiner minimum tree is NP-hard.

Cross-References

- ▸ Greedy Approximation Algorithms
- ▸ Minimum Spanning Trees
- ▸ Rectilinear Steiner Tree

Recommended Reading

1. Arora S (1996) Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. In: Proceedings of the 37th IEEE symposium on foundations of computer science, pp 2–12
2. Berman P, Ramaiyer V (1994) Improved approximations for the Steiner tree problem. J Algorithms 17:381–408
3. Bern M, Plassmann P (1989) The Steiner problem with edge lengths 1 and 2. Inf Proc Lett 32:171–176
4. Chang SK (1972) The generation of minimal trees with a Steiner topology. J ACM 19:699–711
5. Chang SK (1972) The design of network configurations with linear or piecewise linear cost functions. In: Symposium on computer-communications, networks, and teletraffic. IEEE Computer Society Press, Los Alamitos, pp 363–369
6. Crourant R, Robbins H (1941) What is mathematics? Oxford University Press, New York
7. Du DZ, Hwang FK (1990) The Steiner ratio conjecture of Gilbert-Pollak is true. Proc Natl Acad Sci USA 87:9464–9466
8. Du DZ, Zhang Y, Feng Q (1991) On better heuristic for Euclidean Steiner minimum trees. In: Proceedings of the 32nd FOCS. IEEE Computer Society Press, Los Alamitos
9. Du DZ, Graham RL, Pardalos PM, Wan PJ, Wu W, Zhao W (2008) Analysis of greedy approximations with nonsubmodular potential functions. In: Proceedings of 19th ACM-SIAM symposium on discrete algorithms (SODA). ACM, New York, pp 167–175
10. Kahng A, Robins G (1990) A new family of Steiner tree heuristics with good performance: the iterated 1-Steiner approach. In: Proceedings of IEEE international conference on computer-aided design, Santa Clara, pp 428–431
11. Wolsey LA (1982) An analysis of the greedy algorithm for the submodular set covering problem. Combinatorica 2:385–393
12. Robin G, Zelikovsky A (2000) Improved Steiner trees approximation in graphs. In: SIAM-ACM symposium on discrete algorithms (SODA), San Francisco, pp 770–779
13. Smith JM, Lee DT, Liebman JS (1981) An $O(N \log N)$ heuristic for Steiner minimal tree problems in the Euclidean metric. Networks 11:23–39
14. Zelikovsky AZ (1993) The 11/6-approximation algorithm for the Steiner problem on networks. Algorithmica 9:463–470

Stochastic Knapsack

Viswanath Nagarajan
University of Michigan, Ann Arbor, MI, USA

Keywords

Adaptive; Packing constraints; Stochastic optimization

Years and Authors of Summarized Original Work

2004; Dean, Goemans, Vondrák
2011; Bhalgat, Goel, Khanna

Problem Definition

This problem deals with packing a maximum reward set of items into a knapsack of given capacity, when the item sizes are random. The input is a collection of n items, where each item $i \in [n] := \{1, \ldots, n\}$ has reward $r_i \geq 0$ and

size $S_i \geq 0$, and a knapsack capacity $B \geq 0$. In the *stochastic knapsack* problem, all rewards are deterministic but the sizes are random. The random variables S_is are independent with known, arbitrary distributions. The actual size of an item is known only when it is placed into the knapsack. The objective is to add items sequentially (one by one) into the knapsack so as to maximize the expected reward of the items that fit into the knapsack. As usual, a subset T of items is said to fit into the knapsack if the total size $\sum_{i \in T} S_i$ is at most the knapsack capacity B.

A feasible solution (or policy) to the stochastic knapsack problem is represented by a decision tree. Nodes in this decision tree denote the current "state" of the solution (i.e., previously added items and the residual knapsack capacity) as well as the new item to place into the knapsack at this state. Branches in the decision tree denote the random size instantiations of items placed into the knapsack. Such solutions are called *adaptive policies*, to emphasize the fact that the items being placed may depend on previously observed outcomes. More formally, an adaptive policy is given by a mapping $\pi : 2^{[n]} \times [0, B] \to [n]$, where $\pi(T, C)$ denotes the next item to place into the knapsack when some subset $T \subseteq [n]$ of items has already been added, and $C = B - \sum_{i \in T} S_i$ is the residual knapsack capacity. The policy ends when the knapsack overflows (i.e., the total size of items added exceeds the knapsack capacity); we use the convention that no reward is obtained from the last overflowing item.

Notice that an arbitrary adaptive policy may require exponential space to even store. This motivates a special class of solutions, called *nonadaptive policies*. A nonadaptive policy is just specified by a fixed ordering of the n items, and the solution adds items into the knapsack in that order (irrespective of the actual size instantiations) until the knapsack overflows. Again, there is no reward obtained from the last overflowing item. While it may be easier to obtain a good nonadaptive policy, the obvious drawback is that nonadaptive policies may perform much worse than adaptive policies. The benefit of being adaptive is quantified by a measure called the *adaptivity gap*, which is the maximum ratio (over all instances) of the expected reward of an optimal adaptive policy to the expected reward of an optimal nonadaptive policy.

In both the adaptive and nonadaptive settings, the stochastic knapsack problem is at least NP-hard, since it generalizes the deterministic knapsack problem. Moreover, certain questions regarding adaptive policies are PSPACE-hard [4].

Notation We assume that the item size distributions are given explicitly. For any item $i \in [n]$ define its effective reward $w_i = r_i \cdot \Pr[S_i \leq B]$ and its mean truncated size $\mu_i = \mathbb{E}[\min\{S_i, B\}]$. Note that the expected reward obtained by placing the single item i into the knapsack is exactly w_i.

Key Results

Dean, Goemans, and Vondrák introduced the stochastic knapsack problem and the notion of adaptivity gaps. They proved the following.

Theorem 1 ([4]) *There is a polynomial time algorithm for the stochastic knapsack problem that computes a nonadaptive policy having expected reward at least $\frac{1}{4}$ that of an optimal adaptive policy.*

As a consequence, the adaptivity gap of the stochastic knapsack problem is also upper bounded by four. Dean, Goemans, and Vondrák [4] also showed an instance of stochastic knapsack that lower bounds the adaptivity gap by $\frac{5}{4}$.

The algorithm in Theorem 1 uses a natural greedy approach. It outputs the better of the following two nonadaptive policies:

- Place the single item $i^* = \arg\max_{i \in [n]} w_i$.
- Place items in nonincreasing order of w_i / μ_i.

In terms of adaptive policies, Bhalgat, Goel, and Khanna proved the following.

Theorem 2 ([2, 3]) *For any constant $\epsilon > 0$, there is polynomial time algorithm for the stochastic knapsack problem that computes an*

adaptive policy having expected reward at least $\frac{1}{2+\epsilon}$ *that of an optimal adaptive policy.*

The algorithm in Theorem 2 relies on an intricate transformation of general size distributions to certain canonical distributions and an algorithm for computing a near-optimal adaptive policy under canonical size distributions.

Extensions

Several variants of the stochastic knapsack problem have been studied, and good algorithms have been obtained for them.

Correlated Stochastic Knapsack

This is a generalization of the stochastic knapsack problem, where each item's reward is also random and possibly correlated with its size. The distributions across items are still independent: so the correlations are only between the size and reward of a single item. Gupta, Krishnaswamy, Molinaro, and Ravi [6] gave an algorithm for this problem that computes a nonadaptive policy having expected reward within factor 8 of the optimal adaptive policy. Recently, Ma [8] gave an algorithm that for any constant $\epsilon > 0$ computes an adaptive policy having expected reward within factor $2 + \epsilon$ of the optimal adaptive policy; this algorithm requires item sizes and the capacity B to be specified in unary.

Budgeted Multi-armed Bandit

The input to this problem consists of a bound B and n "arms" (each arm is a Markov chain with rewards at its states and a specified starting state). A feasible policy consists of B steps. In each step, the policy can select one arm $i \in [n]$: upon selecting arm i, it gets the reward at the current state of arm i and the arm transitions to its next state according to its Markov chain. The objective is to maximize the expected total reward over B steps of the policy. Again, we are interested in adaptive policies, whose actions may depend on past outcomes. Guha and Munagala [5] introduced this problem and gave a $(2 + \epsilon)$-approximation algorithm, under the assumption that the rewards of each arm satisfy a "Martingale" condition (which is natural in many settings). Gupta, Krishnaswamy, Molinaro, and Ravi [6] gave the first constant-factor approximation algorithm for this problem without the Martingale reward assumption. The constant factor in the latter result was improved to 6.75 by Ma [8].

Stochastic Orienteering

This problem is defined on a finite metric space (V, d) with vertex set V and distance function $d : V \times V \to \mathbb{R}_+$ that satisfies (i) symmetry $d(u, v) = d(v, u)$ for all $u, v \in V$ and (ii) triangle inequality $d(u, w) \leq d(u, v) + d(v, w)$ for all $u, v, w \in V$. The distances between vertices denote travel times. Each vertex $i \in V$ corresponds to a job having deterministic reward $r_i \geq 0$ and random processing time $S_i \geq 0$. The random variables S_is are independent with known, arbitrary distributions. Given a start-vertex $\rho \in V$ and bound B, the goal is to compute a policy, which describes a (possibly adaptive) path originating from ρ that visits vertices and runs the respective jobs. The actual processing time of a job is known only when it completes. The policy ends when the total time (for travel plus processing) exceeds B. The objective is to maximize the expected total reward; there is no reward obtained from a partially completed job (which may occur at the end of the policy). As before, an optimal policy may be adaptive and choose the next job to run based on previously observed outcomes. Gupta, Krishnaswamy, Nagarajan, and Ravi [7] gave an $O(\log \log B)$-approximation algorithm for the stochastic orienteering problem; this result requires the bound B, distances, and processing times to be integer valued. As a corollary, [7] also upper bounded the adaptivity gap by $O(\log \log B)$. Recently, Bansal and Nagarajan [1] gave an $\Omega\left(\sqrt{\log \log B}\right)$ lower bound on the adaptivity gap.

Applications

The stochastic knapsack problem and its variants model various applications in advertising, logistics, medical diagnosis, and robotics.

Open Problems

It is not known if the stochastic knapsack problem is any harder to approximate than the usual (deterministic) knapsack problem. In particular, is there a PTAS for stochastic knapsack? Determining a tight bound on its adaptivity gap is also an interesting open question.

Recommended Reading

1. Bansal N, Nagarajan V (2014) On the adaptivity gap of stochastic orienteering. In: IPCO, Bonn, pp 114–125
2. Bhalgat A (2011) A $(2+\epsilon)$-approximation algorithm for the stochastic knapsack problem. Unpublished manuscript
3. Bhalgat A, Goel A, Khanna S (2011) Improved approximation results for stochastic knapsack problems. In: SODA, San Francisco, pp 1647–1665
4. Dean BC, Goemans, MX, Vondrák J (2008) Approximating the stochastic knapsack problem: the benefit of adaptivity. Math Oper Res 33(4):945–964
5. Guha S, Munagala K (2013) Approximation algorithms for Bayesian multi-armed bandit problems. CoRR abs/1306.3525
6. Gupta A, Krishnaswamy R, Molinaro M, Ravi R (2011) Approximation algorithms for correlated knapsacks and non-martingale bandits. In: FOCS, Palm Springs, pp 827–836
7. Gupta A, Krishnaswamy R, Nagarajan V, Ravi R (2012) Approximation algorithms for stochastic orienteering. In: SODA, Kyoto, pp 1522–1538
8. Ma W (2014) Improvements and generalizations of stochastic knapsack and multi-armed bandit approximation algorithms: extended abstract. In: SODA, Portland, pp 1154–1163

Stochastic Scheduling

Jay Sethuraman
Industrial Engineering and Operations Research, Columbia University, New York, NY, USA

Keywords

Queueing; Sequencing

Years and Authors of Summarized Original Work

2001; Glazebrook, Nino-Mora

Problem Definition

Scheduling is concerned with the allocation of scarce resources (such as machines or servers) to competing activities (such as jobs or customers) over time. The distinguishing feature of a *stochastic* scheduling problem is that some of the relevant data are modeled as *random variables*, whose distributions are known, but whose actual realizations are not. Stochastic scheduling problems inherit several characteristics of their deterministic counterparts. In particular, there are virtually an unlimited number of problem types depending on the machine environment (single machine, parallel machines, job shops, flow shops), processing characteristics (preemptive versus nonpreemptive, batch scheduling versus allowing jobs to arrive "over time," due dates, deadlines), and objectives (makespan, weighted completion time, weighted flow time, weighted tardiness). Furthermore, stochastic scheduling models have some new, interesting features (or difficulties!):

- The scheduler may be able to make inferences about the remaining processing time of a job by using information about its elapsed processing time; whether the scheduler is allowed to make use of this information or not is a question for the modeler.
- Many scheduling algorithms make decisions by comparing the processing times of jobs. If jobs have deterministic processing times, this poses no problems as there is only one way to compare them. If the processing times are random variables, comparing processing times is a subtle issue. There are many ways to compare pairs of random variables, and some are only *partial* orders. Thus, any algorithm that operates by comparing processing times must now specify the particular ordering used to

compare random variables (and to determine what to do if two random variables are not comparable under the specified ordering).

These considerations lead to the notion of a scheduling *policy*, which specifies how the scarce resources have to be allocated to the competing activities as a function of the *state* of the system at any point in time. The state of the system includes information such as prior job completions, the elapsed time of jobs currently in service, the realizations of the random release dates and due dates (if any), and any other information that can be inferred based on the history observed so far. A policy that is allowed to make use of all this information is said to be *dynamic*, whereas a policy that is not allowed to use any state information is *static*.

Given any policy, the objective function for a stochastic scheduling model operating under that policy is typically a random variable. Thus, comparison of two policies entails the comparison of the associated random variables, so the *sense* in which these random variables are compared must be specified. A common approach is to find a solution that optimizes the *expected value* of the objective function (which has the advantage that it is a total ordering); less commonly, other orderings such as the stochastic ordering or the likelihood ratio ordering are used.

Key Results

Consider a single machine that processes n jobs, with the (random) processing time of job i given by a distribution $F_i(\cdot)$ whose mean is p_i. The Weighted Shortest Expected Processing Time first (WSEPT) rule sequences the jobs in decreasing order of w_i / p_i. Smith [13] proved that the WSEPT rule minimizes the sum of weighted completion times when the processing times are deterministic. Rothkopf [11] generalized this result and proved the following:

Theorem 1 *The WSEPT rule minimizes the expected sum of the weighted completion times in the class of all nonpreemptive dynamic policies (and hence also in the class of all nonpreemptive static policies).*

If preemption is allowed, the WSEPT rule is not optimal. Nevertheless, Sevcik [12] showed how to assign an "index" to each job at each point in time such that scheduling a job with the largest index at each point in time is optimal. Such policies are called index policies and have been investigated extensively because they are (relatively) simple to implement and analyze. Often, the optimality of index policies can be proved under some assumptions on the processing time distributions. For instance, Weber, Varaiya, and Walrand [14] proved the following result for scheduling n jobs on m identical parallel machines:

Theorem 2 *The SEPT rule minimizes the expected sum of completion times in the class of all nonpreemptive dynamic polices, if the processing time distributions of the jobs are stochastically ordered.*

For the same problem but with the makespan objective, Bruno, Downey, and Frederickson [3] proved the optimality of the Longest Expected Processing Time first rule provided all the jobs have exponentially distributed processing times.

One of the most significant achievements in stochastic scheduling is the proof of optimality of index policies for the multiarmed bandit problem and its many variants, due originally to Gittins and Jones [5, 6]. In an instance of the bandit problem, there are N projects, each of which is in any one of a possibly finite number of states. At each (discrete) time, any one of the projects can be attempted, resulting in a random reward; the attempted project undergoes a (Markovian) state transition, whereas the other projects remain frozen and do not change state. The goal of the decision maker is to determine an optimal way to attempt the projects so as to maximize the total discounted reward. Of course one can solve this problem as a large, stochastic dynamic program, but such an approach does not reveal any structure and is moreover computationally impractical

except for very small problems. (Also, if the state space of any project is countable or infinite, it is not clear how one can solve the resulting DP exactly!) The remarkable result of Gittins and Jones [6] is the optimality of index policies: to each state of each project, one can associate an index so that attempting a project with the largest index at any point in time is optimal. The original proof of Gittins and Jones [6] has subsequently been simplified by many authors; moreover, several alternative proofs based on different techniques have appeared, leading to a much better understanding of the class of problems for which index policies are optimal [2, 4, 5, 10, 17].

While index policies are easy to implement and analyze, they are often not optimal in many problems. It is therefore natural to investigate the gap between an optimal index policy (or a natural heuristic) and an optimal policy. For example, the WSEPT rule is a natural heuristic for the problem of scheduling jobs on identical parallel machines to minimize the expected sum of the weighted completion times. However, the WSEPT rule is not necessarily optimal. Weiss [16] showed that, under mild and reasonable assumptions, the expected number of times that the WSEPT rule differs from the optimal decision is bounded above by a *constant*, independent of the number of jobs. Thus, the WSEPT rule is asymptotically optimal. As another example of a similar result, Whittle [18] generalized the multiarmed bandit model to allow for state transitions in projects that are *not* activated, giving rise to the "restless bandit" model. For this model, Whittle [18] proposed an index policy whose asymptotic optimality was established by Weber and Weiss [15].

A number of stochastic scheduling models allow for jobs to arrive over time according to a stochastic process. A commonly used model in this setting is that of a multiclass queueing network. Multiclass queueing networks serve as useful models for problems in which several types of *activities* compete for a limited number of shared *resources*. They generalize deterministic job-shop problems in two ways: jobs arrive *over time* and each job has a *random* processing time at each stage. The optimal control problem in a multiclass queueing network is to find an optimal allocation of the available resources to activities over time. Not surprisingly, index policies are optimal only for restricted versions of this general model. An important example is scheduling a multiclass single-server system with feedback: there are N types of jobs; type i jobs arrive according to a Poisson process with rate λ_i, require service according to a service-time distribution $F_i(\cdot)$ with mean processing time s_i, and incur holding costs at rate c_i per unit time. A type i job after undergoing processing becomes a type j job with probability p_{ij} or exits the system with probability $1-\sum_j p_{ij}$ isn't in document. The objective is to find a scheduling policy that minimizes the expected holding cost rate in steady state. Klimov [9] proved the optimality of index policies for this model, as well as for the objective in which the total discounted holding cost is to be minimized. While the optimality result does not hold when there are many parallel machines, Glazebrook and Niño-Mora [7] showed that this rule is asymptotically optimal. For more general models, the prevailing approach is to use approximations such as fluid approximations [1] or diffusion approximations [8].

Applications

Stochastic scheduling models are applicable in many settings, most prominently in computer and communication networks, call centers, logistics and transportation, and manufacturing systems [4, 10].

Cross-References

- List Scheduling
- Minimum Weighted Completion Time

Recommended Reading

1. Avram F, Bertsimas D, Ricard M (1995) Fluid models of sequencing problems in open queueing networks: an optimal control approach. In: Kelly FP, Williams RJ (eds) Stochastic networks. Proceedings

of the international mathematics association, vol 71. Springer, New York, pp 199–234
2. Bertsimas D, Niño-Mora J (1996) Conservation laws, extended polymatroids and multiarmed bandit problems: polyhedral approaches to indexable systems. Math Oper Res 21(2):257–306
3. Bruno J, Downey P, Frederickson GN (1981) Sequencing tasks with exponential service times to minimize the expected flow time or makespan. J ACM 28:100–113
4. Dacre M, Glazebrook K, Nino-Mora J (1999) The achievable region approach to the optimal control of stochastic systems. J R Stat Soc Ser B 61(4):747–791
5. Gittins JC (1979) Bandit processes and dynamic allocation indices. J R Stat Soc Ser B 41(2):148–177
6. Gittins JC, Jones DM (1974) A dynamic allocation index for the sequential design experiments. In: Gani J, Sarkadu K, Vince I (eds) Progress in statistics. European meeting of statisticians I. North Holland, Amsterdam, pp 241–266
7. Glazebrook K, Niño-Mora J (2001) Parallel scheduling of multiclass M/M/m queues: approximate and heavy-traffic optimization of achievable performance. Oper Res 49(4):609–623
8. Harrison JM (1988) Brownian models of queueing networks with heterogenous customer populations. In: Fleming W, Lions PL (eds) Stochastic differential systems, stochastic control theory and applications. Proceedings of the international mathematics association. Springer, New York, pp 147–186
9. Klimov GP (1974) Time-sharing service systems I. Theory Probab Appl 19:532–551
10. Pinedo M (2002) Scheduling: theory, algorithms and systems, 2nd edn. Prentice Hall, Englewood Cliffs
11. Rothkopf M (1966) Scheduling with random service times. Manag Sci 12:707–713
12. Sevcik KC (1974) Scheduling for minimum total loss using service time distributions. J ACM 21:66–75
13. Smith WE (1956) Various optimizers for single-stage production. Nav Res Logist Q 3:59–66
14. Weber RR, Varaiya P, Walrand J (1986) Scheduling jobs with stochastically ordered processing times on parallel machines to minimize expected flow time. J Appl Probab 23:841–847
15. Weber RR, Weiss G (1990) On an index policy for restless bandits. J Appl Probab 27:637–648
16. Weiss G (1992) Turnpike optimality of Smith's rule in parallel machine stochastic scheduling. Math Oper Res 17:255–270
17. Whittle P (1980) Multiarmed bandit and the Gittins index. J R Stat Soc Ser B 42:143–149
18. Whittle P (1988) Restless bandits: activity allocation in a changing world. In: Gani J (ed) A celebration of applied probability. Journal of applied probability, vol 25A. Applied Probability Trust, Sheffield, pp 287–298

String Matching

Maxime Crochemore[1,2,3] and Thierry Lecroq[4]
[1]Department of Computer Science, King's College London, London, UK
[2]Laboratory of Computer Science, University of Paris-East, Paris, France
[3]Université de Marne-la-Vallée, Champs-sur-Marne, France
[4]Computer Science Department and LITIS Faculty of Science, Université de Rouen, Rouen, France

Keywords

Border; Pattern matching; Period; Prefix; Shift function; String matching; Suffix

Years and Authors of Summarized Original Work

1977; Knuth, Morris, Pratt
1977; Boyer, Moore
1994; Crochemore, Czumaj, Gąsieniec, Jarominek, Lecroq, Plandowski, Rytter

Problem Definition

Given a *pattern string* $P = p_1 p_2 \dots p_m$ and a *text string* $T = t_1 t_2 \dots t_n$, both being sequences over an alphabet Σ of size σ, the *exact string-matching (ESM)* problem is to find one or, more generally, all the text positions where P occurs in T, that is, compute the set $\{j \mid 1 \le j \le n-m+1 \text{ and } P = t_j t_{j+1} \dots t_{j+m-1}\}$.

Both worst- and average-case complexities are considered. For the latter one assumes that pattern and text are randomly generated by choosing each character uniformly and independently from Σ. For simplicity and practicality the assumption $m = o(n)$ is made in this entry.

Key Results

Most algorithms that solve the ESM problem proceed in two steps: a preprocessing phase of the pattern P followed by a searching phase over the

text T. The preprocessing phase serves to collect information on the pattern in order to speed up the searching phase.

The searching phase of string-matching algorithms works as follows: they first align the left ends of the pattern and the text, then compare the aligned symbols of the text and the pattern – this specific work is called an attempt or a scan – and after a whole match of the pattern or after a mismatch, they shift the pattern to the right. They repeat the same procedure again until the right end of the pattern goes beyond the right end of the text. The scanning part can be viewed as operating on the text through a window, which size is most often the length of the pattern. This processing manner is called the scan and shift mechanism. Different scanning strategies of the window lead to algorithms having specific properties and advantages.

The brute force algorithm for the ESM problem consists in checking if P occurs at each position j on T, with $1 \leq j \leq n - m + 1$. It does not need any preprocessing phase. It runs in quadratic time $O(mn)$ with constant extra space and performs $O(n)$ character comparisons on average. This is to be compared with the following bounds.

Theorem 1 (Cole et al. [6]) *The minimum number of character comparisons to solve the ESM problem in the worst case is* $n + \frac{9}{4m}(n - m)$, *and there exists an algorithm performing at most* $n + \frac{8}{3(m+1)}(n-m)$ *character comparisons in the worst case.*

Theorem 2 (Yao [26]) *The ESM problem needs* $\Omega\left(\frac{\log_\sigma m}{m} \times n\right)$ *time in expectation.*

Online Text Parsing

The first linear ESM algorithm appears in the 1970s. The preprocessing phase consists in computing the periods of the pattern prefixes, or equivalently the length of the longest border for all the prefixes of the pattern. A border of a string is both a prefix and a suffix of it distinct from the string itself. Let $next[i]$ be the length of the longest border of $p_1 \dots p_{i-1}$. Consider an attempt at position j, when the pattern $p_1 \dots p_m$ is aligned with the segment $t_j \dots t_{j+m-1}$ of the text. Assume that the first mismatch (during a left to right scan) occurs between symbols p_i and t_{i+j} for $1 \leq i \leq m$. Then, $p_1 \dots p_{i-1} = t_j \dots t_{i+j-1} = u$ and $a = p_i \neq t_{i+j} = b$. A prefix v of the pattern may match a suffix of the portion u of the text. By the definition of table *next*, a shift that aligns $p_{next[i]}$ with t_{i+j} cannot miss any occurrence of P in T, and thus backtracking in the text is not necessary. There exist two variants [18, 19], depending on whether $p_{next[i]}$ has to be different from p_i or not. The second is slightly more efficient.

Theorem 3 (Knuth, Morris, and Pratt [18]) *The text searching can be done in time* $O(n)$ *and space* $O(m)$. *Preprocessing the pattern can be done in time* $O(m)$.

The search can also be realized using an implementation with successor by default of the deterministic automaton $\mathcal{D}(P)$ recognizing the language $\Sigma^* P$. The size of the implementation is $O(m)$ independent of the alphabet size, due to the fact that $\mathcal{D}(P)$ possesses $m + 1$ states, m forward arcs, and at most m backward arcs. Using the automaton for searching a text leads to an algorithm having an efficient delay (maximum time for processing a character of the text).

Theorem 4 (Hancart [15]) *Searching for the pattern* P *can be done with a delay of* $O(min\{\sigma, \log_2 m)\})$ *letter comparisons.*

Note that for most algorithms the pattern preprocessing is not necessarily done before the text parsing, as it can be performed on the fly during the parsing.

Algorithms Sublinear on the Average

The Boyer-Moore algorithm [3] is among the most efficient ESM algorithms. A simplified version of it, or the entire algorithm, is often implemented in text editors for the search and substitute commands.

The algorithm scans the characters of the window from right to left beginning with its rightmost symbol. In case of a mismatch (or a complete match of the pattern), it uses two precomputed functions to shift the pattern to the right.

These two shift functions are called the *bad-character shift* and the *good-suffix shift*. They are based on the following observations. Assume that a mismatch occurs between character $p_i = a$ of the pattern and character $t_{i+j} = b$ of the text during an attempt at position j. Then, $p_{i+1} \dots p_m = t_{i+j+1} \dots t_{j+m} = u$ and $p_i \neq t_{i+j}$. The good-suffix shift consists in aligning the segment $t_{i+j+1} \dots t_{j+m}$ with its rightmost occurrence in P that is preceded by a character different from p_i. Another variant called the *best-suffix shift* consists in aligning the segment $t_{i+j} \dots t_{j+m}$ with its rightmost occurrence in P. Both variants can be computed in time and space $O(m)$ independent of the alphabet size. If there exists no such segment, the shift consists in aligning the longest suffix v of $t_{i+j+1} \dots t_{j+m}$ with a matching prefix of x. The bad-character shift consists in aligning the text character t_{i+j} with its rightmost occurrence in $p_1 \dots p_{m-1}$. If t_{i+j} does not appear in the pattern, no occurrence of P in T can overlap the symbol t_{i+j}, then the left end of the pattern is aligned with the character at position $i + j + 1$. The search can then be done in $O(n/m)$ in the best case.

Theorem 5 (Cole [5]) *During the search for a nonperiodic pattern P of length m (such that the length of the longest border of P is less than $m/2$) in a text T of length n, the Boyer-Moore algorithm performs at most $3n$ comparisons between letters of P and of T.*

In practice, when scanning the window from right to left during an attempt, it is sometimes more efficient to only use the bad-character shift. This was first done by the Horspool algorithm [16]. Other practical efficient algorithms are the Quick Search by Sunday [24] and the Tuned Boyer-Moore by Hume and Sunday [17].

Yao's bound can be reached using an indexing structure giving access to all the factors of the reverse pattern. This is done by the Reverse Factor algorithm also called BDM (for Backward Dawg Matching).

Theorem 6 (Crochemore et al. [9]) *The search can be done in optimal expected time $O\left(\frac{\log_\sigma m}{m} \times n\right)$ using the suffix automaton or the suffix tree of the reverse pattern.*

A factor oracle can be used instead of an index structure. A factor oracle is an automaton simpler than the suffix automaton that may recognize some additional strings of length smaller than m. The only string of length m accepted by the factor oracle of a string w of length m is w itself. Then it can be used for solving the ESM problem. This is done by the Backward Oracle Matching (BOM) algorithm of Allauzen, Crochemore, and Raffinot [1]. Its behavior in practice is similar to the one of the BDM algorithm.

Time-Space Optimal Algorithms

Algorithms of this type run in linear time (for both preprocessing and searching) and need only constant space in addition to the inputs.

Theorem 7 (Galil and Seiferas [13]) *The search can be done optimally in time $O(n)$ and constant extra space.*

After Galil and Seiferas' first solution, other solutions are by Crochemore-Perrin [8] and Rytter [22]. These algorithms rely on a partition of the pattern in two parts; they first search for the right part of the pattern from left to right, and then, if no mismatch occurs, they search for the left part. The partition can be the perfect factorization [13], the critical factorization [8], or based on the lexicographically maximum suffix of the pattern [22]. Another solution by Crochemore [7] is a variant of KMP [18]: it computes lower bounds of pattern prefixes periods on the fly and requires no preprocessing.

Bit-Parallel Solution

It is possible to use the bit-parallelism technique for ESM.

Theorem 8 (Baeza-Yates and Gonnet [2]; Wu and Manber [25]) *If the length m of the string P is smaller than the number of bits of a machine word, the preprocessing phase can be done in time and space $O(\sigma)$ and the searching phase in time $O(n)$.*

It is even possible to use this bit-parallelism technique to simulate the BDM algorithm. This is realized by the BNDM (Backward Nondeterministic Dawg Matching) algorithm [20].

There exists another method that uses the bit-parallelism technique that is optimal on the average. It considers sparse q-grams and thus avoids to scan a lot of text positions. It is due to Fredriksson and Grabowski [12].

Applications

The methods that are described here apply to the treatment of the natural language, of genetic and musical sequences, the problems of safety related to data flows like virus detection, and the management of the textual databases, to quote only some immediate applications.

Open Problems

There remain only a few open problems on this question. It is still unknown if it is possible to design an average optimal time constant space string-matching algorithm. The exact size of the Boyer-Moore automaton is still unknown [3]. The Boyer-Moore automaton was first introduced by Knuth [18]. Its states encode all the possible situations when searching the pattern with the Boyer-Moore algorithm and remember every text character already matched in the window.

Experimental Results

The book of G. Navarro and M. Raffinot [21] is a good introduction and presents an experimental map of ESM algorithms for different alphabet sizes and pattern lengths. Basically, the Shift-Or algorithm is efficient for small alphabets and short patterns, the BNDM algorithm is efficient for medium-sized alphabets and medium-length patterns, the Horspool algorithm is efficient for large alphabets, and the BOM algorithm is efficient for long patterns. The article of S. Faro and T. Lecroq [11] updates the experimental map with the most recent results.

URLs to Code and Data Sets

The site monge.univ-mlv.fr/~lecroq/string presents a large number of ESM algorithms (see also [4]). Each algorithm is implemented in C code and a Java applet is given. The site www.dmi.unict.it/~faro/smart presents SMART, a string-matching research tool, which contains the C code of a great number of exact string-matching algorithms and some corpora (natural language, musical, biological, and random texts). The user can easily plug its own algorithm to compare it against some selected algorithms.

Cross-References

- Approximate String Matching is the version where errors are permitted;
- Multiple String Matching is the version where a finite set of patterns is searched for in a text;
- Regular Expression Matching is the more complex case where P can be a regular expression;
- Suffix Trees and Arrays refers to the case where the text is preprocessed.

Further information can be found in the three following books: [10, 14] and [23].

Recommended Reading

1. Allauzen C, Crochemore M, Raffinot M (1999) Factor oracle: a new structure for pattern matching. In: SOFSEM'99, Milovy. LNCS, vol 1725, pp 291–306
2. Baeza-Yates RA, Gonnet GH (1992) A new approach to text searching. C ACM 35(10):74–82
3. Boyer RS, Moore JS (1977) A fast string searching algorithm. C ACM 20(10):762–772
4. Charras C, Lecroq T (2004) Handbook of exact string matching algorithms. King's College, London
5. Cole R (1994) Tight bounds on the complexity of the Boyer-Moore string matching algorithm. SIAM J Comput 23(5):1075–1091
6. Cole R, Hariharan R, Paterson M, Zwick U (1995) Tighter lower bounds on the exact complexity of string matching. SIAM J Comput 24(1):30–45

7. Crochemore M (1992) String-matching on ordered alphabets. Theor Comput Sci 92(1):33–47
8. Crochemore M, Perrin D (1991) Two-way string matching. J ACM 38(3):651–675
9. Crochemore M, Czumaj A, Gąsieniec L, Jarominek S, Lecroq T, Plandowski W, Rytter W (1994) Speeding up two string matching algorithms. Algorithmica 12(4/5):247–267
10. Crochemore M, Hancart C, Lecroq T (2007) Algorithms on strings. Cambridge University Press, New York
11. Faro S, Lecroq T (2013) The exact online string matching problem: a review of the most recent results. C ACM, Harlow, 45(2):13
12. Fredriksson K, Grabowski S (2005) Practical and optimal string matching. In: Proceedings of SPIRE'2005, Buenos Aires. LNCS, vol 3772, pp 374–385
13. Galil Z, Seiferas J (1983) Time-space optimal string matching. J Comput Syst Sci 26(3):280–294
14. Gusfield D (1997) Algorithms on strings, trees and sequences. Cambridge University Press, New York
15. Hancart C (1993) On Simon's string searching algorithm. Inf Process Lett 47(2):95–99
16. Horspool RN (1980) Practical fast searching in strings. Softw Pract Exp 10(6):501–506
17. Hume A, Sunday DM (1991) Fast string searching. Softw Pract Exp 21(11):1221–1248
18. Knuth DE, Morris JH Jr, Pratt VR (1977) Fast pattern matching in strings. SIAM J Comput 6(1):323–350
19. Morris JH Jr, Pratt VR (1970) A linear pattern-matching algorithm. Report 40, University of California, Berkeley
20. Navarro G, Raffinot M (1998) A bit-parallel approach to suffix automata: fast extended string matching. In: Farach-Colton M (ed) Proceedings of the 9th annual symposium on combinatorial pattern matching, Piscataway. Lecture notes in computer science, vol 1448. Springer, Berlin, Piscataway, New Jersey, USA, pp 14–33
21. Navarro G, Raffinot M (2002) Flexible pattern matching in strings – practical on-line search algorithms for texts and biological sequences. Cambridge University Press, Cambridge
22. Rytter W (2003) On maximal suffixes and constant-space linear-time versions of KMP algorithm. Theor Comput Sci 299(1–3):763–774
23. Smyth WF (2002) Computing patterns in strings. Addison Wesley Longman, Harlow
24. Sunday DM (1990) A very fast substring search algorithm. C ACM 33(8):132–142
25. Wu S, Manber U (1992) Fast text searching allowing errors. C ACM 35(10):83–91
26. Yao A (1979) The complexity of pattern matching for a random string. SIAM J Comput 8:368–387

String Sorting

Rolf Fagerberg
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

Keywords

Sorting of multidimensional keys; Vector sorting

Years and Authors of Summarized Original Work

1997; Bentley, Sedgewick

Problem Definition

The problem is to sort a set of strings into lexicographical order. More formally: A *string* over an *alphabet* Σ is a finite sequence $x_1x_2x_3\ldots x_k$ where $x_i \in \Sigma$ for $i = 1,\ldots,k$. The x_is are called the *characters* of the string, and k is the *length* of the string. If the alphabet Σ is ordered, the *lexicographical order* on the set of strings over Σ is defined by declaring a string $x = x_1x_2x_3\ldots x_k$ smaller than a string $y = y_1y_2y_3\ldots y_l$ if either there exists a $j \geq 1$ such that $x_i = y_i$ for $1 \leq i < j$ and $x_j < y_j$ or if $k < l$ and $x_i = y_i$ for $1 \leq i \leq k$. Given a set S of strings over some ordered alphabet, the problem is to sort S according to lexicographical order.

The input to the string sorting problem consists of an array of pointers to the strings to be sorted. The output is a permutation of the array of pointers, such that traversing the array will point to the strings in nondecreasing lexicographical order.

The complexity of string sorting depends on the alphabet as well as the machine model. The main solution [15] described in this entry works for alphabets of unbounded size (i. e., comparisons are the only operations on characters of Σ) and can be implemented on a pointer

machine. See below for more information on the asymptotic complexity of string sorting in various settings.

Key Results

This section is structured as follows: first, the key result appearing in the title of this entry [15] is described; then an overview of other relevant results in the area of string sorting is given.

The string sorting algorithm proposed by Bentley and Sedgewick in 1997 [15] is called three-way radix quicksort [5]. It works for unbounded alphabets, for which it achieves optimal performance.

Theorem 1 *The algorithm three-way radix quicksort sorts K strings of total length N in time $O(K \log K + N)$.*

This time complexity is optimal, which follows by considering strings of the form `bbb...bx`, where all xs are different: Sorting the strings can be no faster than sorting the xs, and all `b`s must be read (else an adversary could change one unread `b` to `a` or `c`, making the returned order incorrect). A more precise version of the bounds above (upper as well as lower) is $K \log K + D$, where D is the sum of the lengths of the *distinguishing prefixes* of the strings. The distinguishing prefix d_s of a string s in a set S is the shortest prefix of s which is not a prefix of another string in S (or is s itself, if s is a prefix of another string). Clearly, $K \leq D \leq N$.

The three-way radix quicksort of Bentley and Sedgewick is not the first algorithm to achieve this complexity; however, it is a very simple and elegant way of doing it. As demonstrated in [3, 15], it is also very fast in practice. Although various elements of the algorithm had been noted earlier, their practical usefulness for string sorting was overlooked until the work in [15].

Three-way radix quicksort is shown in pseudocode in Fig. 1 (adapted from [5]), where S is a list of strings to be sorted and d is an integer. To sort S, an initial call SORT(S, 1) is made. The value s_d denotes the dth character of the string s, and + denotes concatenation. The presentation in Fig. 1 assumes that all strings end in a special end-of-string (EOS) character (such as the null character in C). In an actual implementation, S will be an array of pointers to strings, and the sort will be in-place (using an in-place method from standard quicksort for three-way partitioning of the array into segments holding $S_<$, $S_=$, and $S_>$), rendering concatenation superfluous.

```
SORT(S, d)
  IF |S| ≤ 1:
    RETURN
  Choose a partitioning character v ∈ {s_d | s ∈ S}
  S< = {s ∈ S | s_d < v}
  S= = {s ∈ S | s_d = v}
  S> = {s ∈ S | s_d > v}
  SORT(S<, d)
  IF v ≠ EOS:
    SORT(S=, d + 1)
  SORT(S>, d)
  S = S< + S= + S>
```

String Sorting, Fig. 1 Three-way radix quicksort (assuming each string ends in a special EOS character)

Correctness follows from the following invariant being maintained by the algorithm: At the start of a call SORT(S, d), all strings in S agree on the first $d - 1$ characters.

Time complexity depends on how the partitioning character v is chosen. One particular choice is the median of all the dth characters (including doublets) of the strings in S. Partitioning and median finding can be done in time $O(|S|)$, which is $O(1)$ time per string partitioned. Hence, the total running time of the algorithm is the sum over all strings of the number of partitionings they take part in. For each string, let a partitioning be of type I if the string ends up in $S_<$ or $S_>$ and of type II if it ends up in $S_=$. For a string s, type II can only occur $|d_s|$ times and type I can only occur $\log K$ times. Hence, the running time is $O(K \log K + D)$.

Like for standard quicksort, median finding impairs the constant factors of the algorithm, and more practical choices of partitioning character include selecting a random element among all the dth characters of the strings in S and selecting the median of three elements in this set. The worst-case bound is lost, but the result is a fast, randomized algorithm.

Note that the ternary recursion tree of three-way radix quicksort is equivalent to a trie over the input strings where each trie node is implemented by a binary search tree whose node elements are the child edges (in the trie) of the trie node. In more detail, a node in a binary tree contains the character of a trie edge and a pointer to the root of the binary tree implementing the corresponding trie child. The search keys in a binary tree are the characters in its nodes. This trie implementation is named ternary search trees in [15]. In the recursion tree of three-way radix quicksort, an edge representing a recursive call on $S_<$ or $S_>$ corresponds to a tree edge inside a binary tree implementing a trie node, and an edge representing a recursive call on $S_=$ corresponds to a trie edge.

For the version of the algorithm where the partitioning character v is chosen as the median of all the dth characters, it is not hard to see that the binary trees representing the trie nodes become weighted trees. These are binary trees in which each element x has an associated weight w_x, and searches for x take $O(\log W/w_x)$, where $W = \Sigma_x w_x$ is the sum of all weights in the binary tree. Here, the weight of a binary tree node storing character x is the number of strings which in the trie reside below the corresponding trie edge. As shown in [13], in such a trie implementation, searching for a string P among K stored strings takes time $O(\log K + |P|)$, which is optimal for unbounded (i.e., comparison-based) alphabets. Hence, by the correspondence between the recursion trees of three-way radix quicksort and ternary search trees, three-way radix quicksort may additionally be viewed as a construction algorithm for an efficient dictionary structure for strings.

Other key results in the area of string sorting are now described. The classic string sorting algorithm is radixsort, which assumes a constant-sized alphabet. The least-significant-digit-first variant is easy to implement and runs in $O(N + l|\Sigma|)$ time, where l is the length of the longest string. The most-significant-digit-first variant is more complicated to implement but has a better running time of $O(D + d|\Sigma|)$, where D is the sum of the lengths of the distinguishing prefixes and d is the longest distinguishing prefix. McIlroy et al. [12] discusses in depth efficient implementations of radixsort.

If the alphabet consists of integers, then on a word-RAM the complexity of string sorting is essentially determined by the complexity of integer sorting. More precisely, the time (when allowing randomization) for sorting strings is $\Theta(\text{Sort}_{\text{Int}}(K) + N)$, where $\text{Sort}_{\text{Int}}(K)$ is the time to sort K integers [2], which currently is known to be $O(K\sqrt{\log\log K})$ [11].

Returning to comparison-based model, the papers [8, 10] give generic methods for turning any data structure over one-dimensional keys into a data structure over strings. Using finger search trees, this gives an adaptive sorting method for strings which uses $O(N + K\log(F/K))$ time, where F is the number of inversions among the strings to be sorted.

Concerning space complexity, it has been shown [9] that string sorting can still be done in $O(K\log K + N)$ time using only $O(1)$ space besides the strings themselves. However, this assumes that all strings have equal lengths.

All algorithms so far are designed to work in internal memory, where CPU time is assumed to be the dominating factor. For external memory computation, a more relevant cost measure is the number of I/Os performed, as captured by the I/O model [1], which models a two-level memory hierarchy with an infinite outer memory, an inner memory of size M, and transfer (I/Os) between the two levels taking place in blocks of size B. For external memory, upper bounds

were first given in [4], along with matching lower bounds in restricted I/O models. For a comparison-based model where strings may only be moved in blocks of size B (hence, characters may not be moved individually), it is shown in [4] that string sorting takes $\Theta(N_1/B \log_{M/B}(N_1/B) + K_2 \log_{M/B} K_2 + N/B)$ I/Os, where N_1 is the total length of strings shorter than B characters, K_2 is the number of strings of at least B characters, and N is the total number of characters. This bound is equal to the sum of the I/O costs of sorting the characters of the short strings, sorting B characters from each of the long strings, and scanning all strings. In the same paper, slightly better bounds in a model where characters may be moved individually in internal memory are given, as well as some upper bounds for non-comparison-based string sorting. Further bounds (using randomization) for non-comparison-based string sorting have been given, with I/O bounds of $O(K/B \log_{M/B}(K/M) \log\log_{M/B}(K/M) + N/B)$ [7] and $O(K/B(\log_{M/B}(N/M))^2 \log_2 K + N/B)$ (Ferragina, personal communication).

Returning to internal memory, it may also there be the case that memory hierarchy effects are the determining factor for the running time of algorithms but now due to cache faults rather than disk I/Os. Heuristic algorithms (i.e., algorithms without good worst-case bounds), aiming at minimizing cache faults for internal memory string sorting, have been developed. Of these, the burstsort line of algorithms [16] performs particularly well in experiments.

Applications

Data sets consisting partly or entirely of string data are very common: Most database applications have strings as one of the data types used, and in some areas, such as bioinformatics, Web retrieval, and word processing, string data is predominant. Additionally, strings form a general and fundamental data model, containing, e.g., integers and multidimensional data as special cases. Since sorting is arguably among the most important data processing tasks in any domain, string sorting is a general and important problem with wide practical applications.

Open Problems

As appears from the bounds discussed above, the asymptotic complexity of the string sorting problem is known for comparison-based alphabets. For integer alphabets on the word-RAM, the problem is almost closed in the sense that it is equivalent to integer sorting, for which the gap left between the known bounds and the trivial linear lower bound is small.

In external memory, the situation is less settled. As noted in [4], a natural upper bound to hope for in a comparison-based setting is to meet the lower bound of $\Theta(K/B \log_{M/B} K/M + N/B)$ I/Os, which is the sorting bound for K single characters plus the complexity of scanning the input. The currently known upper bounds only get close to this when leaving the comparison-based setting and allowing randomization.

Experimental Results

In [15], experimental comparison of two implementations (one simple and one tuned) of three-way radix quicksort with a tuned quicksort [6] and a tuned radixsort [12] showed the simple implementation to always outperform the quicksort implementation and the tuned implementation to be competitive with the radixsort implementation.

In [3], experimental comparison among existing and new radixsort implementations (including the one used in [15]), as well as tuned quicksort and tuned three-way radix quicksort, was performed. This study confirms the picture of three-way radix quicksort as very competitive, always being one of the fastest algorithms, and arguably the most robust across various input distributions.

Data Sets

The data sets used in [15]: http://www.cs.princeton.edu/~rs/strings/. The data sets used in [3]: http://dl.acm.org/citation.cfm?id=297136.

URL to Code

Code in C from [15]: http://www.cs.princeton.edu/~rs/strings/.
Code in C from [3]: http://dl.acm.org/citation.cfm?id=297136.
Code in Java from [14]: http://www.cs.princeton.edu/~rs/Algs3.java1-4/code.txt.

Cross-References

- Suffix Array Construction
- Suffix Tree Construction
- Suffix Tree Construction in Hierarchical Memory

Acknowledgments Research supported by Danish Council for Independent Research, Natural Sciences.

Recommended Reading

1. Aggarwal A, Vitter JS (1988) The input/output complexity of sorting and related problems. Commun ACM 31:1116–1127
2. Andersson A, Nilsson S (1994) A new efficient radix sort. In: Proceedings of the 35th annual symposium on foundations of computer science (FOCS'94), Santa Fe. IEEE Computer Society Press, pp 714–721
3. Andersson A, Nilsson S (1998) Implementing radixsort. ACM J Exp Algorithmics 3:7
4. Arge L, Ferragina P, Grossi R, Vitter JS (1997) On sorting strings in external memory (extended abstract). In: Proceedings of the 29th annual ACM symposium on theory of computing (STOC'97), El Paso. ACM, pp 540–548
5. Bentley J, Sedgewick R (1998) Algorithm alley: sorting strings with three-way radix quicksort. Dr Dobb's J Softw Tools 23:133–134, 136–138
6. Bentley JL, McIlroy MD (1993) Engineering a sort function. Softw Pract Exp 23:1249–1265
7. Fagerberg R, Pagh A, Pagh R (2006) External string sorting: faster and cache-oblivious. In: Proceedings of the 23rd annual symposium on theoretical aspects of computer science (STACS'06), Marseille. LNCS, vol 3884. Springer, pp 68–79
8. Franceschini G, Grossi R (2004) A general technique for managing strings in comparison-driven data structures. In: Proceedings of the 31st international colloquium on automata, languages and programming (ICALP'04), Turku. LNCS, vol 3142. Springer, pp 606–617
9. Franceschini G, Grossi R (2005) Optimal in-place sorting of vectors and records. In: Proceedings of the 32nd international colloquium on automata, languages and programming (ICALP'05), Lisbon. LNCS, vol 3580. Springer, pp 90–102
10. Grossi R, Italiano GF (1999) Efficient techniques for maintaining multidimensional keys in linked data structures. In: Proceedings of the 26th international colloquium on automata, languages and programming (ICALP'99), Prague. LNCS, vol 1644. Springer, pp 372–381
11. Han Y, Thorup M (2002) Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In: Proceedings of the 43rd annual symposium on foundations of computer science (FOCS'02), Vancouver. IEEE Computer Society Press, pp 135–144
12. McIlroy PM, Bostic K, McIlroy MD (1993) Engineering radix sort. Comput Syst 6:5–27
13. Mehlhorn K (1979) Dynamic binary search. SIAM J Comput 8:175–198
14. Sedgewick, R (2003) Algorithms in Java, Parts 1–4, 3rd edn. Addison-Wesley, Boston
15. Sedgewick R, Bentley J (1997) Fast algorithms for sorting and searching strings. In: Proceedings of the 8th annual ACM-SIAM symposium on discrete algorithms (SODA'97), New Orleans. ACM, pp 360–369
16. Sinha R, Zobel J, Ring D (2006) Cache-efficient string sorting using copying. ACM J Exp Algorithmics 11: Article No. 1.2

S

Strongly Connected Dominating Set

Zhang Zhao
College of Mathematics Physics and Information Engineering, Zhejiang Normal University, Zhejiang, Jinhua, China

Keywords

Absorbing set; Approximation algorithm; Dominating set; Strongly connected

Years and Authors of Summarized Original Work

2006; Du, Thai, Li, Liu, Zhu
2007: Park, Willson, Wang, Thai, Wu, Du
2009; Li, Du, Wan, Gao, Zhang, Wu
2012; Xu, Li
2014; Zhang, Wu, Wu, Li, Chen

Problem Definition

Let $G = (V, E)$ be a directed graph. For an arc $(u, v) \in E$, u is said to *dominate* v, and v is said to *absorb* u. Vertex u is also called a *dominator* of v, and vertex v is called an *absorber* of u. A vertex set $D \subseteq V$ is a *dominating set* (DS) of G if every vertex in $V \setminus D$ has a dominator in D; it is an *absorbing set* (AS) of G if every vertex in $V \setminus D$ has an absorber in D. A directed graph G is *strongly connected* if for any pair of ordered vertices $u, v \in V$, there is a directed path in G from u to v. The "Minimum Strongly Connected Dominating and Absorbing Set" problem (MSCDAS) is to find a vertex set D such that D is both a dominating set and an absorbing set of G and the subgraph of G induced by D is strongly connected.

Disk graph is a geometric graph which is of particular interest in the study of MSCDAS, since disk graph is a model of heterogeneous wireless sensor network, and as one can see in the application part, MSCDAS plays an important role in wireless sensor network. In a *disk graph*, every vertex u corresponds to a sensor on the plane equipped with an omnidirectional antenna of transmission radius $r(u)$. Another sensor v can correctly decode the message sent by u if and only if v is in the disk centered at u with radius $r(u)$. Hence, there is an arc (u, v) in the disk graph if and only if $\|uv\| \leq r(u)$, where $\|\cdot\|$ is the Euclidean distance between u and v. In particular, if all sensors are equipped with the same transmission radius, then the disk graph degenerates to an undirected graph called *unit disk graph*.

Key Results

Hardness Results

In a general digraph, the MSCDAS problem cannot be approximated within a factor of $(1-\varepsilon)\ln n$ for any real number $\varepsilon > 0$, where n is the number of vertices in the digraph. Even in disk graph, MSCDAS is still NP-hard. These hardness results follow from the fact that their undirected counterparts have these hardness results [1, 5].

MSCDAS in General Digraph

Li et al. [8] gave a $(3H(n-1)-1)$-approximation for MSCDAS, where $H(\gamma) = \sum_{i=1}^{\gamma} 1/i$ is the harmonic number.

The algorithm is based on the following observation. For a vertex u in a digraph G, a spanning *in-arborescence* (resp. *out-arborescence*) rooted at u is a spanning sub-digraph of G in which every vertex except u has in-degree (resp. out-degree) exactly one and vertex u has in-degree (resp. out-degree) zero. For a spanning arborescence T of G, denote by $\text{int}(T)$ the set of internal vertices of T. For any vertex u, suppose T^{in} and T^{out} are spanning in-arborescence and spanning out-arborescence of G rooted at u, respectively. Then $\text{int}(T^{\text{in}}) \cup \text{int}(T^{\text{out}})$ is an SCDAS of G.

Define the problem "Spanning Arborescence with Fewest Internal Vertices" (SAFIV) as follows: given a digraph G and a vertex u, find a spanning arborescence T rooted at u such that $|\text{int}(T)|$ is as small as possible. By the above observation, if SAFIV has a ρ-approximation, then MSCDAS has a 2ρ-approximation. Li et al. gave a $(1.5H(n-1)-0.5)$-approximation for SAFIV, and thus the approximation ratio $(3H(n-1)-1)$ for MSCDAS follows.

The approximation algorithm for SAFIV uses the idea in [6, 7] which study the problem of "Minimum Node-Weighted Steiner Tree" (MNWST). The idea is to iteratively merge smaller arborescences greedily (a vertex is a trivial arborescence) until finally one gets one arborescence including all vertices which is rooted at the given vertex. It was pointed out in [8] that using the method in [6], the approximation ratio for SAFIV can be further reduced to $1.35\ln n$. Since SAFIV is at least as hard as the minimum

connected dominating set problem, it cannot be approximated within factor $(1-\varepsilon)\ln n$. Any progress narrowing the gap between $\ln n$ and $1.35\ln n$ would be interesting.

MSCDAS in Disk Graph

Making use of geometric properties, can the approximation ratio for MSCDAS be better in a disk graph? The answer is yes. Du et al. [2] were the first to give a constant approximation in this setting. Their idea was further explored by Park et al. [11] to output an SCDAS with size at most $9.6(k+1/2)^2\text{opt}+14.8(k+1/2)^2$, where opt is the size of an optimal solution and $k = r_{\max}/r_{\min}$, the ratio between the maximum radius and the minimum radius. The core in their work is an algorithm for SAFIV, which first colors all vertices white and then, by growing a search tree step by step, turns the colors to either black, blue, or gray. The set of black vertices forms a dominating set, and the set of blue vertices connects these black vertices into an out-arborescence. In fact, black vertices are mutually independent, where two vertices u and v are said to be *independent* if either uv or vu is not an arc. Two independent vertices have distance greater than $r_{\min}$. Such a property guarantees an upper bound for the number of black vertices. Furthermore, the structure of a search tree guarantees that the number of blues vertices is no larger than that of black vertices. Then, the desired approximation ratio follows. It should be noted that if $r_{\max}/r_{\min}$ is unbounded, then the approximation ratio is not a constant.

Without a bounded assumption on $r_{\max}/r_{\min}$, Xu and Li [12] showed that a $(2+\varepsilon)$-approximation exists for MDAS, which is a combination of a PTAS for MDS and a PTAS for MAS. In fact, the PTAS for MAS is a special case of the "Geometric Hitting Set" problem studied in [10], and the PTAS for MDS is a variation for the MDS problem in an undirected graph studied in [4]. Both PTASs are obtained through a local search method. The analysis is based on the separator theorem for planar graphs [3, 9]. Zhang et al. [13] also obtained approximation ratio $(2+\varepsilon)$ using the same method. Based on such a DAS, adding Steiner nodes to connect, Zhang et al. showed that a $(4+3\ln(2+\varepsilon)\text{opt}+\varepsilon)$-approximation exists for MSCDAS. When the optimal value opt is substantially smaller than n, this is an improvement on ratio $3H(n-1)-1$ for disk graphs.

Applications

One application of MSCDAS is the communication in wireless sensor network (WSN). In a WSN, information is distributed among sensors by multi-hop transmissions. If all sensors transmit messages in a flooding manner, then a lot of energy is wasted, and large amount of interference is created. To alleviate such problems, it is desirable that only a small fraction of sensors participate in the transmission, while information can still be successfully shared. An SCDAS can serve for this purpose. Suppose D is an SCDAS of directed graph G (the topology of the WSN). If there is a message at source sensor u to be sent to destination sensor v, then the message can be first sent from u to its absorber; since $G[D]$ is strongly connected, it can be successfully relayed to the dominator of v and then sent to v.

Open Problems

It is still open whether there exists a constant approximation algorithm for MSCDAS in disk graph.

Recommended Reading

1. Clark BN, Colbourn CJ, Johnson DS (1991) Unit disk graphs. Ann Discret Math 48:165–177
2. Du D-Z, Thai M, Li Y, Liu D, Zhu S (2006) Strongly connected dominating sets in wireless sensor networks with unidirectional links. In: Proceedings of APWEB, Harbin. LNCS, vol 3841, pp 13–24
3. Frederickson GN (1987) Fast algorithms for shortest paths in planar graphs, with applications. SIAM J Comput 16:1004–1022
4. Gibson M, Pirwani I (2010) Algorithms for dominating set in disk graphs: breaking the $\log n$ barrier. In: Algorithms – ESA, Liverpool. LNCS Vol. 6346, pp 243–254

5. Guha S, Khuller S (1998) Approximation algorithms for connected dominating sets. Algorithmica 20(4):374–387
6. Guha S, Khuller S (1999) Improved methods for approximating node weighted Steiner trees and connected dominating sets. Inf Comput 150:57–74
7. Klein P, Ravi R (1995) A nearly best-possible approximation algorithm for node-weighted Steiner trees. J Algorithms 19:104–114
8. Li D, Du H, Wan P-J, Gao X, Zhang Z, Wu W (2009) Construction of strongly connected dominating sets in asymmetric multihop wireless networks. Theor Comput Sci 410:661–669
9. Lipton RJ, Tarjan RE (1979) A separator theorem for planar graphs. SIAM J Appl Math 36:177–189
10. Mustafa NH, Ray S (2009) PTAS for Geometric hitting set problems via local search. In: SCG'09, Aarhus, 8–10 June 2009
11. Park MA, Willson J, Wang C, Thai M, Wu W, Du D-Z (2007) A dominating and absorbent set in a wireless ad-hoc network with different transmission ranges. In: MobiHoc'07, Montréal
12. Xu X, Li X (2012) Efficient construction of dominating set in wireless networks. http://arxiv.org/abs/1208.5738
13. Zhang Z, Wu W, Wu L, Li Y, Chen Z (accepted) Strongly connected dominating and absorbing set in directed disk graph. To be published in Int J Sens Netw

Subexponential Parameterized Algorithms

Fedor V. Fomin
Department of Informatics, University of Bergen, Bergen, Norway

Keywords

Chordal graph; Exponential time hypothesis; Graph editing; Interval graph; Minimum fill-in; Parameterized complexity

Years and Authors of Summarized Original Work

2009; Alon, Lokshtanov, Saurabh
2013; Fomin, Villanger
2014; Drange, Fomin, Pilipczuk, Villanger

Problem Definition

A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet. The second component is called the *parameter* of the problem. The central notion in parameterized complexity is the notion of *fixed-parameter tractability (FPT)*. A parameterized problem L is called FPT if it can be determined in time $f(k) \cdot n^c$ whether or not $(x,k) \in L$, where $n = |(x,k)|$, f is a computable function depending only on k, and c is a constant independent of n and k. The complexity class containing all fixed-parameter tractable problems is called FPT.

While in the definition of class FPT, we are happy with any computable function f, from application perspective it is often desirable to have the asymptotic growth of f as slow as possible. Take as an example an FPT problem VERTEX COVER which has been subjected to intense scrutiny with progressively faster algorithms designed for it. Let us remind that in the VERTEX COVER problem, we are asked if an n vertex graph G contains a vertex cover of size k or in other words a set of vertices S such that every edge of G has at least one endpoint in S. Starting from a k^k algorithm of Buss and Goldsmith in 1993, there have been algorithms with $f(k) \in \{2^k, 1.324718^k, 1.29175^k, 1.2906^k, 1.271^k, 1.2738^k\}$. The current fastest algorithm for VERTEX COVER runs in time $1.2738^k n^{\mathcal{O}(1)}$ (see the entry ▸ Vertex Cover Search Trees from this book). The ever-decreasing running time leads to the following natural question: can VERTEX COVER admit a *subexponential* time algorithm? That is, can it have an algorithm with running time $2^{o(k)n^{\mathcal{O}(1)}}$? The negative answer to this question would imply that $P \neq NP$. However, using a stronger assumption in complexity theory, namely, exponential time hypothesis (ETH) (see the entry ▸ Exponential Lower Bounds for k-SAT Algorithms in this book), one can show that if ETH holds, then the answer to our question is NO. Moreover, subject to ETH, there are no subexponential algorithms for many other natural *NP*-complete problems. Thus, another natural question arises:

is it true that every *NP*-complete problem cannot be solved in subexponential time? Interestingly, the answer to this question is again NO, and there are examples in the literature of such problems. Coming back to our example of VERTEX COVER problem, if we restrict the input graph to be planar, the problem remains *NP*-complete, but the brute-force algorithm problem can be sped up even more. That is, VERTEX COVER on planar graphs can be solved in time $2^{O(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$, by *a subexponential algorithm*. We refer to more parameterized subexponential algorithms on planar graphs to the ▸ Bidimensionality in this book.

Until recently, the only subexponential algorithms were known for "geometric" graph problems, that is, problems on planar graphs or graphs excluding some fixed graph as minors. In 2009, Alon, Lokshtanov, and Saurabh [1] obtained the first parameterized subexponential algorithm for a natural "nongeometric" problem. This result has acted as catalyst for the discovery of new subexponential time algorithms. In this article, we give a short overview of these algorithms.

Key Results

FAST

In the FEEDBACK ARC SET IN TOURNAMENTS (FAST) problem, we are given an n-vertex tournament T and a positive integer k; the question is whether one can make T into a directed acyclic graph by deleting at most k arcs.

FAST
Input: A tournament $T = (V, E)$ and a nonnegative integer k.
Parameter: k.
Question: Is there $F \subseteq E$, $|F| \leq k$, such that diraph $H = (V, E \setminus F)$ is acyclic?

Alon, Lokshtanov, and Saurabh in [1] obtained a parameterized subexponential algorithm for FAST.

Theorem 1 ([1]) *FAST is solvable in time* $2^{\sqrt{k}\log k} n^{\mathcal{O}(1)}$.

The theorem is proved by making use of a novel randomized technique called *Chromatic Coding*. It appeared that subexponential algorithms exist for several other problems on tournaments (see entry ▸ Computing Cutwidth and Pathwidth of Semi-complete Digraphs in this book).

Fill-In

The next "nongeometric" problem for which a subexponential algorithm was found happened to be the classical MINIMUM FILL-IN problem.

A graph is *chordal* (or triangulated) if every cycle of length at least four contains a chord, i.e., an edge between nonadjacent vertices of the cycle. The MINIMUM FILL-IN problem (also known as MINIMUM TRIANGULATION and CHORDAL GRAPH COMPLETION) is to decide if a given graph G can be transformed into a chordal graph by adding at most k edges.

MINIMUM FILL-IN
Input: A graph $G = (V, E)$ and a nonnegative integer k.
Parameter: k.
Question: Is there $F \subseteq [V]^2$, $|F| \leq k$, such that graph $H = (V, E \cup F)$ is chordal?

Theorem 2 ([6]) MINIMUM FILL-IN *is solvable in time* $2^{\sqrt{k}\log k} n^{\mathcal{O}(1)}$.

The proof of the theorem is based on a combinatorial bound estimating the number of specific objects in the graph, namely, potential maximal cliques.

Completion to Graph Classes

Since discoveries of subexponential algorithms for FAST and MINIMUM FILL-IN, it appeared that several other graph modification problems admit subexponential algorithms. In particular, it was shown that problems of completion to a certain subclass of chordal graphs like trivially perfect, threshold [4], split [7], proper interval [2], and interval graphs [3] admit parameterized subexponential algorithms.

On the other hand, it has been shown that for a number of other graph classes, like cographs, completion to these classes of graphs cannot be done in parameterized subexponential time unless the exponential time hypothesis (ETH) fails [4].

Open Problems

The most natural open question about the given subexponential algorithms is the question about lower bounds. As a concrete example, an algorithm for FAST with running time bound $2^{o(\sqrt{k})}n^{\mathcal{O}(1)}$ would actually be a $2^{o(n)}$ time algorithm which inclines us to suspect that $2^{\mathcal{O}(\sqrt{k})}$ is the best possible dependency on k in the running time for this problem. Unfortunately, there is a big gap here between what we suspect and what we can prove, even assuming ETH. The only tight bound on parameterized subexponential algorithms for graph modification problems we are aware of is the p-CLUSTERING problem [5].

Cross-References

- ▸ Bidimensionality
- ▸ Computing Cutwidth and Pathwidth of Semi–complete Digraphs
- ▸ Exact Algorithms for Treewidth
- ▸ Exponential Lower Bounds for k-SAT Algorithms

Recommended Reading

1. Alon N, Lokshtanov D, Saurabh S (2009) Fast FAST. In: Proceedings of the 36th international colloquium of automata, languages and programming (ICALP). Lecture notes in computer science, vol 5555. Springer, Berlin/New York, pp 49–58
2. Bliznets I, Fomin FV, Pilipczuk M, Pilipczuk M (2014) A subexponential parameterized algorithm for proper interval completion. In: Proceedings of the 22nd annual European symposium on algorithms (ESA 2014). Lecture notes in computer science, vol 8737. Springer, Heidelberg, pp 173–183
3. Bliznets I, Fomin FV, Pilipczuk M, Pilipczuk M (2014) A subexponential parameterized algorithm for interval completion. CoRR. abs/1402.3473
4. Drange PG, Fomin FV, Pilipczuk M, Villanger Y (2014) Exploring subexponential parameterized complexity of completion problems. In: Proceedings of the 31st international symposium on theoretical aspects of computer science (STACS). Leibniz international proceedings in informatics (LIPIcs), vol 25. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, Dagstuhl, pp 288–299
5. Fomin FV, Kratsch S, Pilipczuk M, Pilipczuk M, Villanger Y (2013) Tight bounds for parameterized complexity of cluster editing. In: Proceedings of the 30th international symposium on theoretical aspects of computer science (STACS). Leibniz international proceedings in informatics (LIPIcs), vol 20. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, Dagstuhl, pp 32–43
6. Fomin FV, Villanger Y (2013) Subexponential parameterized algorithm for minimum fill-in. SIAM J Comput 42(6):2197–2216
7. Ghosh E, Kolay S, Kumar M, Misra P, Panolan F, Rai A, Ramanujan MS (2012) Faster parameterized algorithms for deletion to split graphs. In: Proceedings of the 13th Scandinavian symposium and workshops on algorithm theory (SWAT). Lecture notes in computer science, vol 7357. Springer, Berlin/New York, pp 107–118

Subset Sum Algorithm for Bin Packing

Julián Mestre
Department of Computer Science, University of Maryland, College Park, MD, USA
School of Information Technologies, The University of Sydney, Sydney, NSW, Australia

Keywords

Approximation algorithm; Bin packing; Greedy; Knapsack; Subset sum

Years and Authors of Summarized Original Work

1972; Graham
1999; Gupta, Ho

2009; Epstein, Kleiman, Mestre 2011; Epstein, Kleiman

Problem Definition

Bin packing is a classical problem in combinatorial optimization. Given a collection of n items with different sizes, the objective is to pack the items into a minimum number of uniform capacity bins. More formally, the input of the bin packing problem is described by a set of n items $I = \{1, \ldots, n\}$ and a size function $s : I \to [0, 1]$. The output is a packing of the items into bins $B_1, \ldots, B_k \subseteq I$ such that $s(B_j) \leq 1$ for $j = 1, \ldots, k$, where the notation $s(B)$ denotes $\sum_{i \in B} s_i$ for any $B \subseteq I$. The objective is to minimize the number bins used in the packing.

The SUBSET-SUM algorithm is an intuitively appealing greedy heuristic for the bin packing problem: Starting from the empty packing, the algorithm repeatedly finds a subset B of yet-unpacked items maximizing $s(B)$ subject to $s(B) \leq 1$, adds B to the packing, and iterates. Each iteration requires that we solve an instance of the *knapsack* problem. In practice, instead of finding the optimal solution, one can use an fully polynomial time approximation scheme (FPRAS) to compute a $(1 - \epsilon)$-approximate solution [6].

This note is concerned with the worst-case asymptotic performance of the SUBSET-SUM algorithm. For a given instance $s : I \to [0, 1]$, we use $\mathrm{OPT}(s)$ to denote the number of bins used in an optimal packing of s and $\mathrm{SS}(s)$ to denote the number of bins used by the SUBSET-SUM algorithm. Then for a given class $\mathcal{C}$ of instances, we define the worst-case asymptotic approximation ratio of SUBSET-SUM as

$$R_{\mathrm{SS}}^{\infty}(\mathcal{C}) = \lim_{k \to \infty} \sup_{\substack{s \in \mathcal{C} \\ \mathrm{OPT}(s) = k}} \frac{\mathrm{SS}(s)}{\mathrm{OPT}(s)}. \qquad (1)$$

Finally, we use R_{SS}^{∞} to denote the ratio for general instances of the problem.

Key Results

Lower Bound on R_{SS}^{∞}

Graham [4] provided a family of instance exhibiting an approximation ratio that tends to $\sum_{i=1}^{\infty} \frac{1}{2^i - 1} \approx 1.6067$.

Theorem 1 (Graham [4]) $R_{\mathrm{SS}}^{\infty} \geq \sum_{i=1}^{\infty} \frac{1}{2^i - 1} \approx 1.6067$.

Proof Consider the following instance parameterized by two positive integers r and N. For each $j = 1, \ldots, r$, we create N items of size $2^{-i} + \delta$, where $\delta = 2^{-2r}$. Let us denote this instance with s. Provided that $2^i - 1$ divides N for all $i = 1, \ldots, r$, it is not hard to see that SUBSET-SUM first packs the smallest items into $N/(2^r - 1)$ bins, then it packs the second-smallest items into $N/(2^{r-1} - 1)$ bins, and so on, until it packs the largest items into N bins. On the other hand, the optimal solution uses just N bins by packing one item of each size class per bin. Therefore,

$$\frac{\mathrm{SS}(s)}{\mathrm{OPT}(s)} = \sum_{i=1}^{r} \frac{1}{2^i - 1}, \qquad (2)$$

which quickly approaches 1.6067 as r grows. □

Upper Bound on R_{SS}^{∞}

A trivial upper bound on R_{SS}^{∞} is 2. This follows from the fact only the last bin can be less than half full. Caprara and Pferschy [1] gave the first nontrivial upper bound, by showing that R_{SS}^{∞} is at most $4/3 + \ln 4 \approx 1.6210$. Interestingly, Graham [4] had conjectured that the true value of R_{SS}^{∞} should match his lower bound. This conjecture was finally proven by Epstein et al. [2].

Theorem 2 (Epstein et al. [3]) $R_{\mathrm{SS}}^{\infty} \leq \sum_{i=1}^{\infty} \frac{1}{2^i - 1} \approx 1.6067$.

The proof of this result uses *weighting functions* and a *factor revealing mathematical program*. Here we only sketch the high level idea of the approach. Let B be one of the bins opened by SUBSET-SUM. For every item $i \in B$ we define

$$w_i = \begin{cases} \frac{s_i}{s(B)} & \text{if } 1 - s_{\min} \leq s(B), \\ s_i & \text{otherwise,} \end{cases} \qquad (3)$$

S

where $s_{\min}$ is the size of the smallest yet-unpacked item just before opening B.

The weights are used to charge the cost of the packing computed by SUBSET-SUM to an optimal packing. The following lemma allows us to bound the performance of the algorithm provided we can show that the sum of the weights is comparable to the cost of the SUBSET-SUM packing and that no bin in the optimal solution is charged too much.

Lemma 1 *Let $\mathcal{O}$ be an optimal solution and $\mathcal{B}$ be the solution computed* SUBSET-SUM. *If there is a weighting function w such that $w(O) \leq \rho$ for all $O \in \mathcal{O}$ and $|\mathcal{B}| \leq w(I) + \delta$, then $|\mathcal{B}| \leq \rho|\mathcal{O}| + \delta$.*

Proof Because $\mathcal{O}$ is a packing $\sum_{O \in \mathcal{O}} w(O) = w(I)$, therefore,

$$|\mathcal{B}| \leq w(I) + \delta = \sum_{O \in \mathcal{O}} w(O) + \delta \leq \rho|\mathcal{O}| + \delta. \quad \square$$

The key contribution of Epstein et al. [3] was bounding the parameters ρ and δ associated with the weighting function (3). Bounding δ is a relatively straightforward exercise. Bounding ρ is more involved and requires analytically solving a mathematical program. Here we only state their bounds.

Lemma 2 (Epstein et al. [3]) *Let $\mathcal{B}$ be the* SUBSET-SUM *packing and let w be the weighting function* (3) *for $\mathcal{B}$. Then*

1. $|\mathcal{B}| \leq w(I) + 1$,
2. $w(B) \leq \sum_{i=1}^{\infty} \frac{1}{2^i - 1}$ *for all* $B \subseteq I$ *such that* $s(B) \leq 1$.

Theorem 2 follows immediately from Lemmas 1 and 2.

Parametric Case

As it is the case with most bin packing heuristics, the performance of SUBSET-SUM improves when the items are small relative to the capacity of the bin. In a *parametric analysis* of a heuristic, we restrict our attention to instances where the maximum item size is bounded. More formally, for every real $\alpha \in (0, 1]$, we define $\mathcal{C}_\alpha$ to be the class of instances s such that $\max_{i \in I} s_i \leq \alpha$.

Theorem 3 (Epstein et al. [3]) *For every integer $t \geq 1$ and $\alpha \in (\frac{1}{t+1}, \frac{1}{t}]$, we have $R_{\mathrm{SS}}^{\infty}(\mathcal{C}_\alpha) = 1 + \sum_{i=1}^{\infty} \frac{1}{(t+1)2^i - 1}$.*

Notice that this is a strict generalization of Theorems 1 and 2, which only cover the case $\alpha = 1$.

Applications

There is an interesting connection between the performance of the SUBSET-SUM algorithm and the quality of equilibria of a game-theoretic version of bin packing. Let us associate a game with each instance $s : I \to [0, 1]$ of the bin packing problem. The set of players in this game is I, the set of items. Each player can decide in which bin it wants to be packed; this is the player's strategy space. For each bin B chosen in this uncoordinated fashion, if $s(B) > 1$ then the players in B are charged ∞; otherwise, player $i \in B$ is charged $\frac{s_i}{s(B)}$. These payments enforce that a strategy profile is a valid packing if and only if the payments are finite. Furthermore, if the payments are finite, the sum of these payments equals the number of bins in the packing.

A strategy profile is said to be a Nash Equilibrium (NE) if there is no player that can switch bins to decrease its payment. The *price of anarchy* of the bin packing game is the asymptotic worst-case ratio between the number of bins used by an NE and the number of bins in an optimal packing. A packing is said to be a Strong Nash Equilibrium (SNE) if no coalition of players can switch bins to decrease the sum of their payments. The *strong price of anarchy* of the bin packing game is the asymptotic worst-case ratio between the number of bins used by an SNE and the number of bins by an optimal packing.

Theorem 4 (Epstein and Kleiman [2]) *The strong price of anarchy for the bin packing game is exactly R_{SS}^{∞}.*

Notice that every SNE is an NE, since we can think of an NE as requiring that there are no "coalitions" of size 1. Therefore, Theorem 4 establishes a lower bound on the price of anarchy for the bin packing game. However, not every NE is an SNE. In fact, it is known that the price of anarchy for the bin packing game is strictly worse than its strong price of anarchy [2, 3].

Experimental Results

Gupta and Ho [5] performed an experimental evaluation of SUBSET-SUM. (Gupta and Ho call the algorithm *minimum bin slack* because they formulate each iteration as trying to minimize the slack (unused space) of the bin, which is equivalent to maximizing the bin's usage.) The instances used in the evaluation were randomly generated by selecting the item sizes uniformly at random from different numerical ranges. They compared the performance of SUBSET-SUM to two well-known heuristics: FIRST-FIT-DECREASING and BEST-FIT-DECREASING. They observed that SUBSET-SUM performed better on average without incurring a significant computational overhead.

Cross-References

- ▸ Bin Packing
- ▸ Knapsack
- ▸ Price of Anarchy

Recommended Reading

1. Caprara A, Pferschy U (2004) Worst-case analysis of the subset sum algorithm for bin packing. Oper Res Lett 32(2):159–166
2. Epstein L, Kleiman E (2011) Selfish bin packing. Algorithmica 60(2):368–394
3. Epstein L, Kleiman E, Mestre J (2009) Parametric packing of selfish items and the subset sum algorithm. In: Proceedings of the 5th workshop on internet and network economics, Rome, Italy pp 67–78
4. Graham RL (1972) Bounds on multiprocessing anomalies and related packing algorithms. In: Proceedings of the 1972 spring joint computer conference, Atlantic City, New Jersey, USA pp 205–217
5. Gupta J, Ho J (1999) A new heuristic algorithm for the one-dimensional bin-packing problem. Prod Plan Control 10(6):598–603
6. Kellerer H, Pferschy U, Pisinger D (2004) Knapsack problems. Springer, Berlin/New York

Substring Parsimony

Mathieu Blanchette
Department of Computer Science, McGill University, Montreal, QC, Canada

Years and Authors of Summarized Original Work

2001; Blanchette, Schwikowski, Tompa

Problem Definition

The Substring Parsimony Problem, introduced by Blanchette et al. [1] in the context of motif discovery in biological sequences, can be described in a more general framework:
Input:

- A discrete space $\mathcal{S}$ on which an integral distance d is defined (i.e., $d(x, y) \in \mathbb{N}\ \forall x, y \in \mathcal{S}$).
- A rooted binary tree $T = (V, E)$ with n leaves. Vertices are labeled $\{1, 2, \ldots, n, \ldots, |V|\}$, where the leaves are vertices $\{1, 2, \ldots, n\}$.
- Finite sets $S_1, S_2, \ldots, S_n$, where set $S_i \subseteq \mathcal{S}$ is assigned to leaf i, for all $i = 1 \ldots n$.
- A non-negative integer t

Output: All solutions of the form $(x_1, x_2, \ldots, x_n, \ldots, x_{|V|})$ such that:

- $x_i \in \mathcal{S}$ for all $i = 1 \ldots |V|$

- $x_i \in S_i$ for all $i = 1 \ldots n$
- $\sum_{(u,v)\in E} d(x_u, x_v) \leq t$

The problem thus consists of choosing one element x_i from each set S_i such that the Steiner distance of the set of points is at most t. This is done on a Steiner tree T of fixed topology. The case where $|S_i| = 1$ for all $i = 1 \ldots n$ is a standard Steiner tree problem on a fixed tree topology (see [11]). It is known as the Maximum Parsimony Problem and its complexity depends on the space $\mathcal{S}$.

Key Results

The substring parsimony problem can be solved using a dynamic programming algorithm. Let $u \in V$ and $s \in \mathcal{S}$. Let $W_u[s]$ be the score of the best solution that can be obtained for the subtree rooted at node u, under the constraint that node u is labeled with s, i.e.,

$$W_u[s] = \min_{\substack{x_1,\ldots,x_{|V|}\in\mathcal{S} \\ x_u = s}} \sum_{\substack{(i,j)\in E \\ i,j\in \text{subtree}(u)}} d(x_i, x_j).$$

Let v be a child of u, and let $X_{(u,v)}[s]$ be the score of the best solution that can be obtained for the subtree consisting of node u together with the subtree rooted at its child v, under the constraint that node u is labeled with s:

$$X_{(u,v)}[s] = \min_{\substack{x_1,\ldots,x_{|V|}\in\mathcal{S} \\ x_u = s}} \sum_{\substack{(i,j)\in E \\ i,j\in \text{subtree}(v)\cup\{(u,v)\}}} d(x_i, x_j).$$

Then, we have:

$$W_u[s] = \begin{cases} 0 & \text{if } u \text{ is a leaf and } s \in S_u \\ +\infty & \text{if } u \text{ is a leaf and } s \notin S_u \\ \sum_{v\in \text{children}(u)} X_{(u,v)}[s] & \text{if } u \text{ is not a leaf} \end{cases}$$

and

$$X_{(u,v)}[s] = \min_{y'\in\mathcal{S}} W_u[s'] + d(s, s').$$

Tables W and X can thus be computed using a dynamic programming algorithm, proceeding in a post-order traversal of the tree. Solutions can then be recovered by tracing the computation back for all s such that $W_{\text{root}}[s] \leq t$. Note that the same solution may be recovered more than once in this process.

A straight-forward implementation of this dynamic programming algorithm would run in time $O(n \cdot |\mathcal{S}|^2 \cdot \gamma(\mathcal{S}))$, where $\gamma(\mathcal{S})$ is the time needed to compute the distance between any two points in $\mathcal{S}$. Let $N_a(\mathcal{S})$ be the maximum number of a-neighbors a point in $\mathcal{S}$ can have, i.e., $N_a(\mathcal{S}) = \max_{x\in\mathcal{S}} |\{y \in \mathcal{S} : d(x, y) = a\}|$. Blanchette et al. [3] showed how to use a modified breadth-first search of the space $\mathcal{S}$ to compute each table $X_{(u,v)}$ in time $O(|\mathcal{S}| \cdot N_1(\mathcal{S}))$, thus reducing the total time complexity to $O(n \cdot |\mathcal{S}| \cdot N_1(\mathcal{S}))$. Since only solutions with a score of at most t are of interest, the complexity can be further reduced by only computing those table entries which will yield a score of at most t. This results in an algorithm whose running time is $O(n \cdot M \cdot N_{\lfloor t/2 \rfloor}(\mathcal{S}) \cdot N_1(\mathcal{S}))$ where $M = \max_{i=1\ldots n} |S_i|$.

The problem has been mostly studied in the context of biological sequence analysis, where $\mathcal{S} = \{A, C, G, T\}^k$, for some small k ($k = 5, \ldots, 20$ are typical values). The distance d is the Hamming distance, and a phylogenetic tree T is given. The case where $|S_i| = 1$ for all $i = 1 \ldots n$ is known as the Maximum Parsimony Problem and can be solved in time $O(n \cdot k)$ using Fitch's algorithm [9] or Sankoff's algorithm [12]. In the more general version, a long DNA sequence P_u of length L is assigned to each leaf u. The set S_u is defined as the set of all k-substrings of P_u. In this case, $M = L - k + 1 \in O(L)$, and $N_a \in O(\min(4^k, (3k)^a))$, resulting in a complexity of $O(n \cdot L \cdot 3k \cdot \min(4^k, (3k)^{\lfloor d/2 \rfloor}))$. Notice that for a fixed k and d, the algorithm is linear over the whole sequence. The problem was independently shown to be NP-hard by Blanchette et al. [3] and by Elias [7].

Applications

Most applications are found in computational biology, although the algorithm can be applied to a wide variety of domains. The algorithm

for the substring parsimony problem has been implemented in a software package called FootPrinter [5] and applied to the detection of transcription factor binding sites in orthologous DNA regulatory sequences through a method called phylogenetic footprinting [4]. Other applications include the search for conserved RNA secondary structure motifs in orthologous RNA sequences [2]. Variants of the problem have been defined to identify motifs regulating alternative splicing [13]. Blanchette et al. [3] study a relaxation of the problem where one does not require that a substring be chosen from each of the input sequences, but instead asks that substrings be chosen from a sufficiently large subset of the input sequence. Fang and Blanchette [8] formulate another variant of the problem where substring choices are constrained to respect a partial order relation defined by a set of local multiple sequence alignments.

Open Problems

Optimizations taking advantage of the specific structure of the space $\mathcal{S}$ may yield more efficient algorithms in certain cases. Many important variations could be considered. First, the case where the tree topology is not given needs to be considered, although the resulting problems would usually be NP-hard even when $|S_i| = 1$. Another important variation is one where the phylogenetic relationships between trees is not given by a tree but rather by a phylogenetic network [10]. Finally, randomized algorithms similar to those proposed by Buhler et al. [6] may yield important and practical improvements.

URL to Code

http://bio.cs.washington.edu/software.html

Cross-References

- Closest Substring
- Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds
- Local Alignment (with Affine Gap Weights)
- Local Alignment (with Concave Gap Weights)
- Statistical Multiple Alignment
- Steiner Trees

Recommended Reading

1. Blanchette M (2001) Algorithms for phylogenetic footprinting. In: RECOMB01: proceedings of the fifth annual international conference on computational molecular biology, Montreal. ACM, pp 49–58
2. Blanchette M (2002) Algorithms for phylogenetic footprinting. PhD thesis, University of Washington
3. Blanchette M, Schwikowski B, Tompa M (2002) Algorithms for phylogenetic footprinting. J Comput Biol 9(2):211–223
4. Blanchette M, Tompa M (2002) Discovery of regulatory elements by a computational method for phylogenetic footprinting. Genome Res 12:739–748
5. Blanchette M, Tompa M (2003) Footprinter: a program designed for phylogenetic footprinting. Nucleic Acids Res 31(13):3840–3842
6. Buhler J, Tompa M (2001) Finding motifs using random projections. In: RECOMB01: proceedings of the fifth annual international conference on computational molecular biology, pp 69–76
7. Elias I (2006) Settling the intractability of multiple alignment. J Comput Biol 13:1323–1339
8. Fang F, Blanchette M (2006) Footprinter3: phylogenetic footprinting in partially alignable sequences. Nucleic Acids Res 34(2):617–620
9. Fitch WM (1971) Toward defining the course of evolution: minimum change for a specified tree topology. Syst Zool 20:406–416
10. Huson DH, Bryant D (2006) Application of phylogenetic networks in evolutionary studies. Mol Biol Evol 23(2):254–267
11. Sankoff D, Rousseau P (1975) Locating the vertices of a Steiner tree in arbitrary metric space. Math Program 9:240–246
12. Sankoff DD (1975) Minimal mutation trees of sequences. SIAM J Appl Math 28:35–42
13. Shigemizu D, Maruyama O (2004) Searching for regulatory elements of alternative splicing events using phylogenetic footprinting. In: Proceedings of the fourth workshop on algorithms for bioinformatics. Lecture notes in computer science. Springer, Berlin, pp 147–158

Succinct and Compressed Data Structures for Permutations and Integer Functions

Jérémy Barbay
Department of Computer Science (DCC),
University of Chile, Santiago, Chile

Keywords

Adaptive; Compression; Functions; Permutation

Years and Authors of Summarized Original Work

2012; Munro, Raman, Raman, Rao
2012; Barbay, Fischer, Navarro
2013; Barbay, Navarro
2013; Barbay

Problem Definition

A basic building block for compressed data structures for texts and functions is the representation of a permutation of the integers $\{1, \ldots, n\}$, denoted by $[1 \ldots n]$. A permutation π is trivially representable in $n\lceil \lg n \rceil$ bits which is within $O(n)$ bits of the information theoretic bound of $\lg(n!)$, but instances from restricted classes of permutations can be represented using much less space.

We are interested in encodings of permutations that can efficiently access them. Given a permutation π over $[1 \ldots n]$, an integer k and an integer $i \in [1 \ldots n]$, data structures on permutations aim to support the following operators as fast as possible, using as little additional space as possible:

- $\pi(i)$: application of the permutation to i,
- $\pi^{-1}(i)$: application of the inverse permutation to i,
- $\pi^{(k)}(i)$: $\pi()$ iteratively applied k times starting with value i (e.g., $\pi^{(2)}(i) = \pi(\pi(i))$).

Key Results

We distinguish between two types of solutions: the succinct index and two succinct data structures for permutations introduced by Munro et al. [1], and the various compressed data structures proposed later [2–4].

Succinct Data Structures

Munro et al. [1] studied the problem of succinctly representing a permutation to support operators on it quickly. They give several solutions, described below.

"Shortcut" Index Supporting $\pi()$ and $\pi^{-1}()$

Given an integer parameter t, the operators $\pi()$ and $\pi^{-1}()$ can be supported by simply writing down π in an array of n words of $\lceil \lg n \rceil$ bits each, plus an auxiliary array S of at most n/t back pointers called shortcuts: in each cycle of length at least t, every t-th element has a pointer t steps back. Then, $\pi(i)$ is simply the i-th value in the primary structure, and $\pi^{-1}(i)$ is found by moving forward until a back pointer is found and then continuing to follow the cycle to the location that contains the value i.

The trick is in the encoding of the locations of the back pointers: this is done with a simple bit vector B of length n, in which a 1 indicates that a back pointer is associated with a given location. B is augmented using $o(n)$ additional bits so that the number of 1's up to a given position and the position of the r-th 1 can be found in constant time (i.e., using the rank and select operators on binary strings [5]). This gives the location of the appropriate back pointer in the auxiliary array S. As there are back pointers every t elements in the cycle, finding the predecessor requires $O(t)$ memory accesses.

Theorem 1 *For any strictly positive integer n and any permutation π on $[1 \ldots n]$ which can be decomposed into δ cycles of respective sizes $c_1, \ldots, c_\delta$, there is a representation of π using within $(\sum_{i \in [1 \ldots \delta]} \lfloor \frac{c_i}{t} \rfloor) \lg n + 2n + o(n) \subseteq \frac{n \lg n}{t} + 2n + o(n)$ bits to support the operator $\pi()$ in constant time and the operator $\pi^{-1}()$ in time within $O(t)$.*

Interestingly enough, Munro et al. [1] did not notice that their construction is actually an index and that the raw encoding can be replaced by any data structure supporting the operator $\pi()$, including the compressed ones later described [4].

"Cycle" Data Structure Supporting $\pi^k()$

For arbitrary i and k, $\pi^k()$ is supported by writing the cycles of π together with a bit vector B marking the beginning of each cycle. Observe that the cycle representation itself is a permutation in "standard form"; call it σ. The first task is to find i in the representation: it is in position $\sigma^{-1}(i)$. The segment of the representation containing i is found through the rank and select operators on B. Then $\pi^k(i)$ is determined by taking k modulo the cycle length, moving that number of steps around the cycle starting at the position of i, and applying $\sigma()$ to obtain the value to return.

Other than the support of the operators on σ, all operators are performed in constant time; hence the asymptotic supporting time of $\pi^k()$ depends on the supporting time in which the data structure chosen to represent σ supports the operators $\sigma()$ and $\sigma^{-1}()$. Munro et al. [1] proposed the following, using a raw encoding of σ with a shortcut index to support $\sigma^{-1}()$:

Theorem 2 *For any strictly positive integer n and any permutation π on $[1 \dots n]$, there is a representation of π using at most $(1+\varepsilon)n \lg n + O(n)$ bits to support the operator $\pi^k()$ in time within $O(1/\varepsilon)$, for any ε less than 1 and for any arbitrary value of k.*

Under a restricted model of pointer machine, this technique is optimal: using $O(n)$ extra bits (i.e., $O(n/\log n)$ extra words), time within $\Omega(\log n)$ is necessary to support both $\pi()$ and $\pi^{-1}()$.

"Benes Network" Data Structure Supporting $\pi^k()$

Any permutation can be implemented by a communication network composed of switches: this is called a Benes Network and uses even less space under the RAM model than the solutions described in the previous sections. Sparsely adding pointers accelerates the support of $\pi^k()$ to time within $O(\frac{\log n}{\log \log n})$.

Theorem 3 *For any strictly positive integer n and any permutation π on $[1 \dots n]$, there is a representation of π using at most $\lceil \lg(n!) \rceil + O(n)$ bits to support the operator $\pi^k()$ in time within $O(\log n / \log \log n)$.*

This representation uses space within an additive term within $O(n)$ of the optimal, both on average and in the worst case over all permutations over $[1 \dots n]$.

Compressed Data Structures

Any comparison-based sorting algorithm yields an encoding for permutations, and any adaptive sorting algorithm in the comparison model yields a compression scheme for permutations. Supporting operators on such compressed permutation in less time than required to decompress the whole of it requires some more work:

Runs

Barbay and Navarro [2] described how to segment a partition into $\texttt{nRuns}$ *runs* composed of consecutive positions forming already sorted blocks and how to merge those via a wavelet tree. This yields a data structure compressing a permutation within space optimal over all permutations with $\texttt{nRuns}$ runs of sizes given by the vector $\texttt{vRuns}$. This data structure supports the operators $\pi()$ and $\pi^{-1}()$ in sublinear time within $O(1 + \log \texttt{nRuns})$, with the average supporting time within $O(1 + \mathcal{H}(\texttt{vRuns}))$, which decreases with the entropy of the partition of the permutation into runs. Here, the *entropy* of a sequence of positive integers $X = \langle n_1, n_2, \dots, n_r \rangle$ adding up to n is $\mathcal{H}(X) = \sum_{i=1}^{r} \frac{n_i}{n} \lg \frac{n}{n_i}$.

Theorem 4 *For any strictly positive integer n and any permutation π on $[1 \dots n]$ which can be decomposed into $\texttt{nRuns}$ runs of sizes $\texttt{vRuns} = (r_1, \dots, r_{\texttt{nRuns}})$, there is a representation of π using at most $n\mathcal{H}(\texttt{vRuns}) + O(\texttt{nRuns} \log n) + o(n)$ bits to support the computation of $\pi(i)$ and $\pi^{-1}(i)$ in time within $O(1 + \log \texttt{nRuns})$ in the worst case over $i \in [1 \dots n]$ and in*

time within $O(1 + \mathcal{H}(\texttt{vRuns}))$ *on average when* $i \in [1 \ldots n]$ *is uniformly distributed. This compressed data structure can be computed in time within* $O(n(1 + \mathcal{H}(\texttt{vRuns})))$*, which is worst-case optimal in the comparison model over all such permutations decomposed into* nRuns *runs of sizes given by the vector* vRuns.

The partitioning takes only $n - 1$ comparisons, and the construction of the compressed data structure itself is an adaptive sorting algorithm improving over previous results [6,7].

Heads of Strict Runs

A two-level partition of the permutation yields further compression [2]. The first level partitions the permutation into *strict ascending runs* (maximal ranges of positions satisfying $\pi(i + k) = \pi(i) + k$). The second level partitions the *heads* (first position) of those strict runs into conventional ascending runs. This is analogous to the notion of blocks described by Moffat and Petersson [7] for multisets.

Theorem 5 *For any strictly positive integer* n *and any permutation* π *on* $[1 \ldots n]$ *which can be decomposed into* nBlock *strict runs and into* $\texttt{nRuns} \leq \texttt{nBlock}$ *monotone runs, let* vHRuns *be the vector formed by the* nRuns *monotone run lengths in the permutation of strict run heads. Then, there is a representation of* π *using at most* $\texttt{nBlock}\mathcal{H}(\texttt{vHRuns})) + O(\texttt{nBlock} \log \frac{n}{\texttt{nBlock}}) + o(n)$ *bits to support the operator* $\pi()$ *and* $\pi^{-1}()$ *in time within* $O(1 + \log \texttt{nBlock})$*. This compressed data structure can be computed in time within* $O(n(1 + \log \texttt{nBlock}))$.

Shuffled Subsequences

The preorder measures seen so far have considered runs which group contiguous positions in π: this does not need to be always the case. A permutation π over $[1 \ldots n]$ can be decomposed in n comparisons into a minimal number nSUS of *Shuffled Up Sequences*, defined as a set of, not necessarily consecutive, subsequences of increasing numbers that have to be removed from π in order to reduce it to the empty sequence [8]. Then those subsequences can be merged using the same techniques as above, which yields a new adaptive sorting algorithm and a new compressed data structure [2]. An optimal partition of a permutation π over $[1 \ldots n]$ into a minimal number nSMS of *Shuffled Monotone Sequences*, sequences of not necessarily consecutive subsequences of increasing or decreasing numbers, is NP-hard to compute [9], but if such a permutation is given, the same technique applies [10].

LRM Subsequences

LRM trees partition a sequence of values into consecutive sorted blocks and express the relative position of the first element of each block within a previous block. Such a tree can be computed in $2(n - 1)$ comparisons within the array and overall linear time, through an algorithm similar to that of Cartesian Trees [11]. The interest of LRM trees in the context of adaptive sorting and permutation compression is that the values are increasing in each root-to-leaf branch: they form a partition of the array into subsequences of increasing values. Barbay et al. [3] described how to compute the partition of the LRM tree of minimal size-vector entropy, which yields a compressed data structure asymptotically smaller than $\mathcal{H}(\texttt{vRuns})$-adaptive sorting, smaller in practice than $\mathcal{H}(\texttt{vSUS})$-adaptive sorting, as well as a faster adaptive sorting algorithm.

Number of Inversions

The preorder measure nInv counts the number of pairs (i, j) of positions $1 \leq i < j \leq n$ in a permutation π over $[1 \ldots n]$ such that $\pi(i) > \pi(j)$. Its value is exactly the number of comparisons performed by the algorithm `Insertion Sort`, between n and n^2 for a permutation over $[1 \ldots n]$. A variant of `Insertion Sort`, named `Local Insertion Sort`, sorts π in $n(1 + \lceil \lg(\texttt{nInv}/n) \rceil)$ comparisons [6,7].

Simply encoding the n values $(\pi(i) - i)_{i \in [1 \ldots n]}$ using the γ' code from Elias [12], and indexing the positions of the beginning of each code by a compressed bit vector, yields a compressed data structure supporting the operator $\pi()$ in constant time. The resulting data structure uses space within $n(1 + 2 \lg \frac{\texttt{nInv}}{n}) + o(n)$ bits. Support for

the operator $\pi^{-1}()$ can be added in two distinct ways, either encoding both π and π^{-1} using this technique within $2n(1 + 2\lg \frac{\texttt{nInv}}{n}) + o(n)$ bits, which supports both operators $\pi()$ and $\pi^{-1}()$ in constant time, or adding support for the operator $\pi^{-1}()$ using Munro et al.'s shortcut succinct index for permutations [1] described previously.

Removing Elements

The preorder measure `nRem` counts the minimum number of elements that must be removed from a permutation so that what remains is already sorted. Its exact value is n minus the length of the *Longest Increasing Subsequence*, which can be computed in time within $O(n \log n)$. Alternatively, the value of `nRem` can be approximated within a constant factor of 2 in $2(n - 1)$ comparisons. Partitioning π into the removed elements and the remaining ones through a bit vector of n bits, representing the order of the `2nRem` elements in a wavelet tree (using any of the data structures described above), and representing the merging of both into n bits yield a compressed data structure using space within $2n + 2\texttt{nRem}\lg(n/\texttt{nRem}) + o(n)$ bits and supporting the operators $\pi()$ and $\pi^{-1}()$ in sublinear time, within $O(1 + \log(\texttt{nRem} + 1))$.

Applications

Integer Functions

Munro et al. [1] extended the results on permutations to arbitrary functions from $[1 \ldots n]$ to $[1 \ldots n]$. Again $f^k(i)$ indicates the function iterated k times starting at i: if k is nonnegative, this is straightforward. The case in which k is negative is more complicated as the image is a (possibly empty) multiset over $[1 \ldots n]$.

Whereas π is a set of cycles, f can be viewed as a set of cycles in which each node is the root of a tree. Starting at any node (element of $[1 \ldots n]$), the evaluation moves one step along a branch of the tree, or one step along a cycle. Moving k steps in a positive direction is straightforward, and one moves up a tree and perhaps around a cycle. When k is negative, one must determine all nodes at distance k from the starting location, i, in the direction toward the leaves of the trees. The key technical issue is to run across succinct tree representations picking off all nodes at the appropriate levels. Using a raw encoding of the permutation mapping integers to the nodes, and Munro et al.'s shortcut succinct index [1] to support the operations on it, yields the following result:

Theorem 6 *For any fixed ε, $n > 0$ and $f : [1 \ldots n] \to [1 \ldots n]$, there is a representation of f using $(1 + \varepsilon)n \lg n + O(1)$ bits of space to compute $f^k(i)$ in time within $O(1 + |f^k(i)|)$, for any integer k and for any integer $i \in [1 \ldots n]$.*

Open Problems

Other Measures of Disorder

Moffat and Petersson [7] list many measures of preorder and adaptive sorting techniques. Each measure explored above yields a compressed data structure for permutations supporting the operators $\pi()$ and $\pi^{-1}()$ in sublinear time. Each adaptive sorting algorithm in the comparison model yields a compression scheme for permutations, but the encoding thus defined does not necessarily support the simple application of the permutation to a single element without decompressing the whole permutation nor the application of the inverse permutation. More work is required in order to decide whether there are compressed data structures for permutations, supporting the operators $\pi()$ and $\pi^{-1}()$ in sublinear time and using space proportional to the other preorder measures [6, 7] (e.g., `Reg`, `Exc`, `Block`, and `Enc`).

Sorting and Encoding Multisets

Munro and Spira [13] showed how to sort multisets through `MergeSort`, `Insertion Sort`, and `Heap Sort`, adapting them with counters to sort in time within $O(n(1 + \mathcal{H}(\langle m_1, \ldots, m_r \rangle)))$ where m_i is the number of occurrences of i in the multiset (note that this is orthogonal to the results described in this chapter that depend on the distribution of the lengths of monotone runs).

It seems easy to combine both approaches (e.g., on `MergeSort` in a single algorithm using both runs and counters), yet quite hard to *analyze* the complexity of the resulting algorithm and compressed data structure. The difficulty measure must depend not only on both the entropy of the partition into runs and the entropy of the partition of the values of the elements but also on the interaction of those partitions.

Compressed Data Structures Supporting $\pi^k()$

In Munro et al.'s "cycle" data structure [1] for supporting the operator $\pi^k()$ (Theorem 2), the raw encoding of the permutation σ representing the cycles of π can be replaced by any compressed data structure such as those described here, with the warning that the compressibility of σ depends not only on π but also on the order in which its cycles are placed in σ. The question if there is a compressed data structure supporting the operator $\pi^k()$ which takes advantage of this order is open.

Recommended Reading

1. Ian Munro J, Raman R, Raman V, Srinivasa Rao S (2012) Succinct representations of permutations and functions. Theorical Computer Science (TCS) 438:74–88
2. Barbay J, Navarro G (2013) On compressing permutations and adaptive sorting. Theorical Computer Science (TCS) 513:109–123
3. Barbay J, Fischer J, Navarro G (2012) LRM-trees: compressed indices, adaptive sorting, and compressed permutations. Theorical Computer Science (TCS) 459:26–41
4. Barbay J (2013) From time to space: fast algorithms that yield small and fast data structures. In: Brodnik A, López-Ortiz A, Raman V, Viola A (eds) Space-efficient data structures, streams, and algorithms (Ian-Fest). Volume 8066 of Lecture Notes in Computer Science. Springer, Heidelberg, pp 97–111
5. Ian Munro J, Raman V (1997) Succinct representation of balanced parentheses, static trees and planar graphs. In: IEEE symposium on Foundations Of Computer Science, Miami Beach, pp 118–126
6. Estivill-Castro V, Wood D (1992) A survey of adaptive sorting algorithms. ACM Computing Survey 24(4):441–476
7. Moffat A, Petersson O (1992) An overview of adaptive sorting. Aust Comput J 24(2):70–77
8. Levcopoulos C, Petersson O (1990) Sorting shuffled monotone sequences. In: Proceedings of the Scandinavian Workshop on Algorithm Theory (SWAT), Bergen. Springer, London, pp 181–191
9. Levcopoulos C, Petersson O (1994) Sorting shuffled monotone sequences. Information Computing 112(1):37–50
10. Barbay J, Claude F, Gagie T, Navarro G, Nekrich Y (2014) Efficient fully-compressed sequence representations. Algorithmica 69(1):232–268
11. Gabow HN, Bentley JL, Tarjan RE (1984) Scaling and related techniques for geometry problems. In: Proceedings of the Symposium on Theorical Computer (STOC), Washington, DC. ACM, pp 135–143
12. Elias P (1975) Universal codeword sets and representations of the integers. IEEE Transaction on Information Theory 21(2):194–203
13. Ian Munro J, Spira PM (1976) Sorting and searching in multisets. SIAM Journal of Computing 5(1):1–8

Succinct Data Structures for Parentheses Matching

Meng He
School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Keywords

Succinct balanced parentheses

Years and Authors of Summarized Original Work

2001; Munro, Raman

Problem Definition

This problem is to design succinct representation of balanced parentheses in a manner in which a number of "natural" queries can be supported quickly, and use it to represent trees and graphs succinctly. The problem of succinctly representing balanced parentheses was initially proposed by Jacobson [6] in 1989, when he proposed *succinct data structures*, i.e., data structures that occupy space close to the information-theoretic

lower bound to represent them, while supporting efficient navigational operations. Succinct data structures provide solutions to manipulate large data in modern applications. The work of Munro and Raman [8] provides an optimal solution to the problem of balanced parentheses representation under the word RAM model, based on which they design succinct trees and graphs.

Balanced Parentheses

Given a balanced parenthesis sequence of length $2n$, where there are n opening parentheses and n closing parentheses, consider the following operations:

- `findclose`(i) (`findopen`(i)), the matching closing (opening) parenthesis for the opening (closing) parenthesis at position i;
- `excess`(i), the number of opening parentheses minus the number of closing parentheses in the sequence up to (and including) position i;
- `enclose`(i), the closest enclosing (matching parenthesis) pair of a given matching parenthesis pair whose opening parenthesis is at position i.

Trees

There are essentially two forms of trees. An *ordinal tree* is a rooted tree in which the children of a node are ordered and specified by their ranks, while in a *cardinal tree* of degree k, each child of a node is identified by a unique number from the set $\{1, 2, \cdots, k\}$. An *binary tree* is a cardinal tree of degree 2. The information-theoretic lower bound of representing an ordinal tree or binary tree of n nodes is $2n - o(n)$ bits, as there are $\binom{2n}{n}/(n+1)$ different ordinal trees or binary trees.

Consider the following operations on ordinal trees (a node is referred to by its preorder number):

- `child`(x, i), the ith child of node x for $i \geq 1$;
- `child_rank`(x), the number of left siblings of node x;
- `depth`(x), the depth of x, i.e., the number of edges in the rooted path to node x;
- `parent`(x), the parent of node x;
- `nbdesc`(x), the number of descendants of node x;
- `height`(x), the height of the subtree rooted at node x;
- `LCA`(x, y), the lowest common ancestor of node x and node y.

On binary trees, the operations `parent`, `nbdesc` and the following operations are considered:

- `leftchild`(x) (`rightchild`(x)), the left (right) child of node x.

Graphs

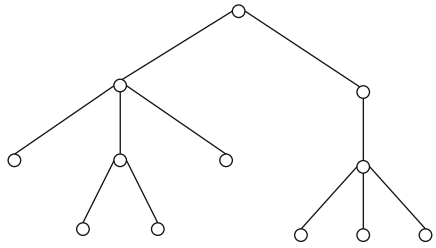
Consider an undirected graph G of n vertices and m edges. Bernhart and Kainen [1] introduced the concept of *page book embedding*. A *k-book embedding* of a graph is a topological embedding of it in a book of k pages that specifies the ordering of the vertices along the spine, and carries each edge into the interior of one page, such that the edges on a given page do not intersect. Thus, a graph with one page is an *outerplanar graph*. The *pagenumber* or *book thickness* [1] of a graph is the minimum number of pages that the graph can be embedded in. A very common type of graphs are planar graphs, and any planar graph can be embedded in at most four pages [15]. Consider the following operations on graphs:

- `adjacency`(x, y), whether vertices x and y are adjacent;
- `degree`(x), the degree of vertex x;
- `neighbors`(x), the neighbors of vertex x.

Key Results

All the results cited are under the word RAM model with word size $\Theta(\lg n)$ bits ($\lg n$ denotes $\lceil \log_2 n \rceil$), where n is the size of the problem considered.

Theorem 1 ([8]) *A sequence of balanced parentheses of length 2n can be represented using*



Balanced parentheses: ((( ) (( ) ( )) ( )) ((( ) ( ) ( ))))

Succinct Data Structures for Parentheses Matching, Fig. 1 An example of the balanced parenthesis sequence of a given ordinal tree

$2n + o(n)$ *bits to support the operations findclose, findopen, excess and enclose in constant time.*

There is a polymorphism between a balanced parenthesis sequence and an ordinal tree: when performing a depth-first traversal of the tree, output an opening parenthesis each time a node is visited, and a closing parenthesis immediately after all the descendants of a node are visited (see Fig. 1 for an example). The work of Munro and Raman proposes a succinct representation of ordinal trees using $2n + o(n)$ bits to support `depth`, `parent` and `nbdesc` in constant time, and `child`(x, i) in $O(i)$ time. Lu and Yeh have further extended this representation to support `child`, `child_rank`, `height` and `LCA` in constant time.

Theorem 2 ([8, 7]) *An ordinal tree of n nodes can be represented using* $2n + o(n)$ *bits to support the operations child, child_rank, parent, depth, nbdesc, height and LCA in constant time.*

A similar approach can be used to represent binary trees:

Theorem 3 ([8]) *A binary tree of n nodes can be represented using* $2n + o(n)$ *bits to support the operations leftchild, rightchild, parent and nbdesc in constant time.*

Finally, balanced parentheses can be used to represent graphs. To represent a one-page graph, the work of Munro and Raman proposes to list the vertices from left to right along the spine, and each node is represented by a pair of parentheses, followed by zero or more closing parentheses and then zero or more opening parentheses, where the number of closing (or opening) parentheses is equal to the number of adjacent vertices to its left (or right) along the spine (see Fig. 2 for an example). This representation can be applied to each page to represent a graph with pagenumber k.

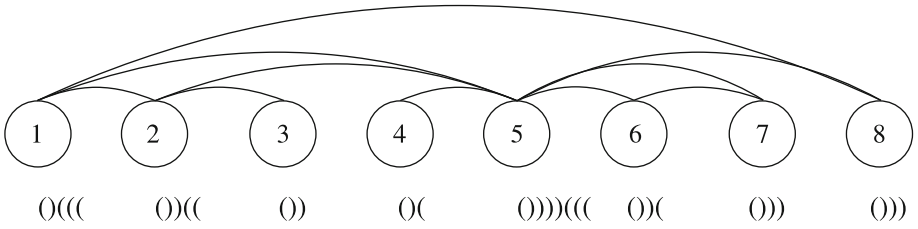
Theorem 4 ([8]) *An outerplanar graph of n vertices and m edges can be represented using* $2n + 2m + o(n + m)$ *bits to support operations adjacency and degree in constant time, and neighbors(x) in time proportional to the degree of x.*

Theorem 5 ([8]) *A graph of n vertices and m edges with pagenumber k can be represented using* $2kn + 2m + o(nk + m)$ *bits to support operations adjacency and degree in* $O(k)$ *time, and neighbors(x) in* $O(d(x) + k)$ *time where d(x) is the degree of x. In particular, a planar graph of n vertices and m nodes can be represented using* $8n + 2m + o(n)$ *bits to support operations adjacency and degree in constant time, and neighbors(x) in* $O(d(x))$ *time where d(x) is the degree of x.*

Applications

Succinct Representation of Suffix Trees

As a result of the growth of the textual data in databases and on the World Wide Web, and also applications in bioinformatics, various indexing techniques have been developed to facilitate pattern searching. Suffix trees [14] are a popular type of text indexes. A suffix tree is constructed over the suffixes of the text as a tree-based data structure, so that queries can be performed by searching the suffixes of the text. It takes $O(m)$ time to use a suffix tree to check whether an arbitrary pattern P of length m is a substring of a given text T of length n, and to count the number of the occurrences, occ, of P in T. $O(occ)$ additional time is required to list all the occurrences



Succinct Data Structures for Parentheses Matching, Fig. 2 An example of the balanced parenthesis sequence of a graph with one page

of P in T. However, a standard representation of a suffix tree requires somewhere between $4n \lg n$ and $6n \lg n$ bits, which is impractical for many applications.

By reducing the space cost of representing the tree structure of a suffix tree (using the work of Munro and Raman), Munro, Raman and Rao [9] have designed space-efficient suffix trees. Given a string of n characters over a fixed alphabet, they can represent a suffix tree using $n \lg n + O(n)$ bits to support the search of a pattern in $O(m + occ)$ time. To achieve this result, they have also extended the work of Munro and Raman to support various operations to retrieve the leaves of a given subtree in an ordinal tree. Based on similar ideas and by applying compressed suffix arrays [5], Sadakane [13] has proposed a different trade-off; his compressed suffix tree occupies $O(n \lg \sigma)$ bits, where σ is the size of the alphabet, and can support any algorithm on a suffix tree with a slight slowdown of a factor of polylog(n).

Succinct Representation of Functions

Munro and Rao [11] have considered the problem of succinctly representing a given function, $f : [n] \to [n]$, to support the computation of $f^k(i)$ for an arbitrary integer k. The straightforward representation of a function is to store the sequence $f(i)$, for $i = 0, 1, \ldots, n-1$. This takes $n \lg n$ bits, which is optimal. However, the computation of $f^k(i)$ takes $\Theta(k)$ time even in the easier case when k is positive. To address this problem, Munro and Rao [11] first extends the representation of balanced parenthesis to support the `next_excess`(i, k) operator, which returns the minimum j such that $j > i$ and `excess`$(j) = k$. They further use this operator to support the `level_anc`(x, i) operator on succinct ordinal trees, which returns the ith ancestor of node x for $i \geq 0$ (given a node x at depth d, its ith ancestor is the ancestor of x at depth $d - i$). Then, using succinct ordinal trees with the support for `level_anc`, they propose a succinct representation of functions using $(1 + \epsilon)n \lg n + O(1)$ bits for any fixed positive constant ϵ, to support $f^k(i)$ in constant time when $k > 0$, and $f^k(i)$ in $O(1 + |f^k(i)|)$ time when $k < 0$.

Multiple Parentheses and Graphs

Chuang et al. [3] have proposed to succinctly represent *multiple parentheses*, which is a string of $O(1)$ types of parentheses that may be unbalanced. They have extended the operations on balanced parentheses to multiple parentheses and designed a succinct representation. Based on the properties of canonical orderings for planar graphs, they have used multiple parentheses and the succinct ordinal trees to represent planar graphs. One of their main results is a succinct representation of planar graphs of n vertices and m edges in $2m + (5 + \epsilon)n + o(m + n)$ bits, for any constant $\epsilon > 0$, to support the operations supported on planar graphs in Theorem 5 in asymptotically the same amount of time. Chiang et al. [2] have further reduced the space cost to $2m + 3n + o(m + n)$ bits. In their paper, they have also shown how to support the operation `wrapped`(i), which returns the number of matching parenthesis pairs whose closest enclosing (matching parenthesis) pair is the pair whose opening parenthesis is at position i, in constant time on balanced parentheses. They

have used it to show how to support the operation `degree`(x), which returns the degree of node x (i.e., the number of its children), in constant time on succinct ordinal trees.

Open Problems

One open research area is to support more operations on succinct trees. For example, it is not known how to support the operation to convert a given node's rank in a preorder traversal into its rank in a level-order traversal.

Another open research area is to further reduce the space cost of succinct planar graphs. It is not known whether it is possible to further improve the encoding of Chiang et al. [2].

A third direction for future work is to design succinct representations of dynamic trees and graphs. There have been some preliminary results by Munro et al. [10] on succinctly representing dynamic binary trees, which have been further improved by Raman and Rao [12]. It may be possible to further improve these results, and there are other related dynamic data structures that do not have succinct representations.

Experimental Results

Geary et al. [4] have engineered the implementation of succinct ordinal trees based on balanced parentheses. They have performed experiments on large XML trees. Their implementation uses orders of magnitude less space than the standard pointed-based representation, while supporting tree traversal operations with only a slight slowdown.

Cross-References

- ▸ Compressed Suffix Array
- ▸ Compressed Text Indexing
- ▸ Rank and Select Operations on Bit Strings
- ▸ Text Indexing

Recommended Reading

1. Bernhart F, Kainen PC (1979) The book thickness of a graph. J Comb Theory B 27(3):320–331
2. Chiang Y-T, Lin C-C, Lu H-I (2005) Orderly spanning trees with applications. SIAM J Comput 34(4):924–945
3. Chuang RC-N, Garg A, He X, Kao M-Y, Lu H-I (2001) Compact encodings of planar graphs via canonical orderings and multiple parentheses. Comput Res Repos. cs.DS/0102005
4. Geary RF, Rahman N, Raman R, Raman V (2006) A simple optimal representation for balanced parentheses. Theor Comput Sci 368(3):231–246
5. Grossi R, Gupta A, Vitter JS (2003) High-order entropy-compressed text indexes. In: Farach-Colton M (ed) Proceedings of the 14th annual ACM-SIAM symposium on discrete algorithms. SIAM, Philadelphia, pp 841–850
6. Jacobson G (1989) Space-efficient static trees and graphs. In: Proceedings of the 30th annual IEEE symposium on foundations of computer science. IEEE, New York, pp 549–554
7. Lu H-I, Yeh C-C (2007) Balanced parentheses strike back. Accepted to ACM Trans Algorithms
8. Munro JI, Raman V (2001) Succinct representation of balanced parentheses and static trees. SIAM J Comput 31(3):762–776
9. Munro JI, Raman V, Rao SS (2001) Space efficient suffix trees. J Algorithms 39(2):205–222
10. Munro JI, Raman V, Storm AJ (2001) Representing dynamic binary trees succinctly. In: Rao Kosaraju S (ed) Proceedings of the 12th annual ACM-SIAM symposium on discrete algorithms. SIAM, Philadelphia, pp 529–536
11. Munro JI, Rao SS (2004) Succinct representations of functions. In: Díaz J, Karhumäki J, Lepistö A, Sannella D (eds) Proceedings of the 31st international colloquium on automata, languages and programming. Springer, Heidelberg, pp 1006–1015
12. Raman R, Rao SS (2003) Succinct dynamic dictionaries and trees. In: Baeten JCM, Lenstra JK, Parrow J, Woeginger GJ (eds) Proceedings of the 30th international colloquium on automata, languages and programming. Springer, Heidelberg, pp 357–368
13. Sadakane K (2007) Compressed suffix trees with full functionality. Theory Comput Syst. Online first. http://dx.doi.org/10.1007/s00224-006-1198-x
14. Weiner P (1973) Linear pattern matching algorithms. In: Proceedings of the 14th annual IEEE symposium on switching and automata theory. IEEE, New York, pp 1–11
15. Yannakakis M (1986) Four pages are necessary and sufficient for planar graphs. In: Hartmanis J (ed) Proceedings of the 18th annual ACM-SIAM symposium on theory of computing. ACM, New York, pp 104–108

Suffix Array Construction

Juha Kärkkäinen
Department of Computer Science, University of Helsinki, Helsinki, Finland

Keywords

Longest common prefix array; Suffix array; Suffix sorting; Text indexing

Years and Authors of Summarized Original Work

2006; Kärkkäinen, Sanders, Burkhardt

Problem Definition

The *suffix array* [4, 15] is the lexicographically sorted array of all the suffixes of a string. It is a popular text index structure with many applications. The subject of this entry is algorithms that construct the suffix array.

More precisely, the input to a suffix array construction algorithm is a *text string* $T = T[0 \ldots n) = t_0 t_1 \ldots t_{n-1}$, i.e., a sequence of n *characters* from an *alphabet* Σ. For $i \in [0 \ldots n]$, let S_i denote the *suffix* $T[i \ldots n) = t_i t_{i+1} \ldots t_{n-1}$. The output is the *suffix array* $SA[0 \ldots n]$ of T, a permutation of $[0 \ldots n]$ satisfying $S_{SA[0]} < S_{SA[1]} < \cdots < S_{SA[n]}$, where $<$ denotes the *lexicographical order* of strings.

Two specific models for the alphabet Σ are considered. An *ordered alphabet* is an arbitrary ordered set with constant time character comparisons. An *integer alphabet* is the integer range $[1 \ldots \sigma]$ for $\sigma = n^{\mathcal{O}(1)}$.

Many applications require that the suffix array is augmented with additional information, most commonly with the *longest common prefix* array $LCP[1 \ldots n]$. An entry $LCP[i]$ of the LCP array is the length of the longest common prefix of the suffixes $S_{SA[i]}$ and $S_{SA[i-1]}$. The *enhanced suffix array* [1] adds two more arrays to obtain a full range of text index functionalities.

There are other important text indexes, most notably suffix trees and compressed text indexes, covered in separate entries. Each of these indexes has their own construction algorithms, but they can also be constructed efficiently from each other. However, in this entry, the focus is on direct suffix array construction algorithms that do not rely on other text indexes.

Key Results

The naive approach to suffix array construction is to use a general sorting algorithm or an algorithm for sorting strings. However, any such algorithm has a worst-case time complexity $\Omega(n^2)$ because the total length of the suffixes is $\Omega(n^2)$.

The first efficient algorithms were based on the *doubling technique* of Karp, Miller, and Rosenberg [10]. The idea is to assign a *rank* to all substrings whose length is a power of two. The rank tells the lexicographic order of the substring among substrings of the same length. Given the ranks for substrings of length h, the ranks for substrings of length $2h$ can be computed using a radix sort step in linear time (doubling). The technique was first applied to suffix array construction by Manber and Myers [15]. The best practical algorithm based on the technique is by Larsson and Sadakane [14].

Theorem 1 (Manber and Myers [15]; Larsson and Sadakane [14]) *The suffix array can be constructed in $\mathcal{O}(n \log n)$ time, which is optimal for the ordered alphabet.*

Faster algorithms for the integer alphabet are based on a different technique, recursion. The basic procedure is as follows.

1. Sort a subset of the suffixes. This is done by constructing a shorter string, whose suffix array gives the order of the desired subset. The suffix array of the shorter string is constructed by recursion.
2. Extend the subset order to full order.

The technique first appeared in suffix tree construction [3], but 2003 saw the independent and simultaneous publication of three linear time suffix array construction algorithms based on the approach but not using suffix trees. Each of the three algorithms uses a different subset of suffixes requiring a different implementation of the second step.

Theorem 2 (Kärkkäinen, Sanders, and Burkhardt [8]; Kim et al. [12]; Ko and Aluru [13]) *The suffix array can be constructed in the optimal linear time for the integer alphabet.*

We will describe the algorithm of Kärkkäinen, Sanders, and Burkhardt [8] called DC3 in more detail. For $k \in \{0, 1, 2\}$, let $\mathcal{R}_k$ be the set of suffixes S_i such that $i \bmod 3 = k$. Let $\mathcal{R}_{12} = \mathcal{R}_1 \cup \mathcal{R}_2$ and define $\mathcal{R}_{01}$ and $\mathcal{R}_{02}$ symmetrically. For example, $\mathcal{R}_{12} = \{S_1, S_2, S_4, S_5, S_7, S_8, \dots\}$. The set $\mathcal{R}_{12}$ is the subset of suffixes sorted first. For $S_i \in \mathcal{R}_{12}$, let $\bar{S}_i$ be the lexicographical rank of S_i in $\mathcal{R}_{12}$. Given those lexicographical ranks, we can compare any two suffixes S_i and S_j in constant time using one of the following ways:

1. If $S_i, S_j \in \mathcal{R}_{12}$, compare the ranks $\bar{S}_i$ and $\bar{S}_j$.
2. If $S_i, S_j \in \mathcal{R}_{01}$, compare the pairs $\langle t_i, \bar{S}_{i+1}\rangle$ and $\langle t_j, \bar{S}_{j+1}\rangle$.
3. If $S_i, S_j \in \mathcal{R}_{02}$, compare the triples $\langle t_i, t_{i+1}, \bar{S}_{i+2}\rangle$ and $\langle t_j, t_{j+1}, \bar{S}_{j+2}\rangle$.

Furthermore, we can radix sort $\mathcal{R}_0$ in linear time by using $\langle t_i, \bar{S}_{i+1}\rangle$ to represent the suffix $S_i \in \mathcal{R}_0$. After this, we can merge $\mathcal{R}_0$ and $\mathcal{R}_{12}$, which takes linear time since we can compare suffixes in constant time.

We still need to describe how to sort $\mathcal{R}_{12}$. Let $\overline{t_i t_{i+1} t_{i+2}}$ be the lexicographical rank of the substring $t_i t_{i+1} t_{i+2}$ among all substrings of length three. Let

$$T_{12} = \overline{t_1 t_2 t_3}\ \overline{t_4 t_5 t_6}\ \overline{t_7 t_8 t_9} \dots \overline{t_2 t_3 t_4}\ \overline{t_5 t_6 t_7}\ \overline{t_8 t_9 t_{10}} \dots .$$

For example if $T =$ `yabbadabbado`, we have

$$T_{12} = \overline{\texttt{abb}}\ \overline{\texttt{ada}}\ \overline{\texttt{bba}}\ \overline{\texttt{do\$}}\ \overline{\texttt{bba}}\ \overline{\texttt{dab}}\ \overline{\texttt{bad}}\ \overline{\texttt{o\$\$}} = 12575648 ,$$

where \$ is a special padding symbol that does not appear in the text and is considered smaller than any normal character. Clearly, sorting the suffixes of T_{12} is equivalent to sorting the set $\mathcal{R}_{12}$. The suffixes of T_{12} are sorted by a recursive call to the algorithm itself. Since the recursive call is for a text of length at most $\lceil 2n/3 \rceil$ and everything outside the recursive call can be done in linear time, the total time complexity of DC3 is $\mathcal{O}(n)$.

The above algorithms and many other suffix array construction algorithms are surveyed in [18]. Worth mentioning among the more recent results are the linear time algorithms of Nong, Zhang, and Chan [17].

The $\Omega(n \log n)$ lower bound for the ordered alphabet mentioned in Theorem 1 comes from the sorting complexity of characters, since the initial characters of the sorted suffixes are the text characters in sorted order. Theorem 2 allows a generalization of this result. For any alphabet, one can first sort the characters of T, remove duplicates, assign a rank to each character, and construct a new string T' over the alphabet $[1 \dots n]$ by replacing the characters of T with their ranks. The suffix array of T' is exactly the same as the suffix array of T. Optimal algorithms for the integer alphabet then give the following result.

Theorem 3 *For any alphabet, the complexity of suffix array construction is the same as the complexity of sorting the characters of the string.*

The result extends to the related arrays.

Theorem 4 (Kasai et al. [11]; Abouelhoda, Kurtz, and Ohlebusch [1]) *The LCP array and the enhanced suffix array can be computed in linear time given the suffix array.*

One of the main advantages of suffix arrays over suffix trees is their smaller space requirement (by a constant factor), and a significant effort has been spent making construction algorithms space efficient, too. The best algorithms need very little extra space.

Theorem 5 (Kärkkäinen, Sanders, and Burkhardt [8]; Nong [16]) *For any* $v = \mathcal{O}(n^{2/3})$*, the suffix array can be constructed in* $\mathcal{O}(n(v + \log n))$ *time and* $\mathcal{O}(n/\sqrt{v})$ *extra space for the ordered alphabet and in* $\mathcal{O}(nv)$ *time and* $\mathcal{O}(n/\sqrt{v})$ *extra space or* $\mathcal{O}(n)$ *time and* $\mathcal{O}(\sigma)$ *extra space for the integer alphabet, where the extra space is the space needed in addition to the input (the string* T*) and the output (the suffix array).*

In the algorithm DC3 described above, all steps can be performed by sorting, prefix sums (assigning lexicographical ranks) and localized computation. This makes it straightforward to adapt to several parallel and hierarchical memory models of computation [8] including the following result for the standard external memory model.

Theorem 6 (Kärkkäinen, Sanders, and Burkhardt [8]) *The suffix array can be constructed in the optimal* $\mathcal{O}(\text{sort}(n))$ *I/Os in the standard external memory model, where* $\text{sort}(n)$ *is the I/O complexity of sorting n elements.*

The above algorithm can be modified to compute the LCP array too in the same I/O complexity [2, 7].

Applications

The suffix array is a simple and powerful text index structure with numerous applications; see [1] and Cross-References. The practical construction of many other text indexes usually starts with the suffix array construction. In particular, the Burrows–Wheeler transform, which is an important technique for text compression and the basis of many compressed text indexes, is easily computed from the suffix array.

Open Problems

Theoretically, the suffix array construction problem is essentially solved. The development of ever more efficient practical algorithms is still going on particularly for external memory and parallel computation. There is currently no external memory algorithm for computing the LCP array from the suffix array in $\mathcal{O}(\text{sort}(n))$ I/Os other than as a side effect of suffix array construction [6].

Experimental Results

Many papers on suffix array construction contain experimental results, but they are usually either out of date (e.g., [18]) or limited in scope (e.g., [16]). The most comprehensive comparison of algorithms is at https://code.google.com/p/libdivsufsort/wiki/SACA_Benchmarks. The best practical algorithms for large data are `divsufsort`, which is an $\mathcal{O}(n \log n)$ time algorithm combining several techniques, and `SAIS`, which is an implementation of the linear time algorithm by Gong, Zhang, and Chan [17] (see below for URLs to code). The comparison and the fastest implementation are by the same person, Yuta Mori, but the implementations are widely used and there are no substantial claims for other, faster algorithms.

There are also experiments for suffix array construction in external memory [2, 5] and for LCP array construction [2, 6, 9].

URLs to Code and Data Sets

The input to a suffix array construction algorithm is simply a text, so an abundance of data exists. Links to many text collections are provided at https://code.google.com/p/libdivsufsort/wiki/SACA_Benchmarks. Worth mentioning is also the Pizza&Chili site with its standard text corpus http://pizzachili.dcc.uchile.cl/texts.html and the repetitive text corpus http://pizzachili.dcc.uchile.cl/repcorpus.html.

Notable implementations of suffix array construction algorithms are available at https://code.google.com/p/libdivsufsort/, at https://sites.google.com/site/yuta256/sais, at http://panthema.net/2012/1119-eSAIS-Inducing-Suffix-and-LCP-Arrays-in-External-Memory/ [2], and at https://

www.cs.helsinki.fi/group/pads/SAscan.html [5]. The latter two work in external memory and provide (links to) LCP array construction too.

Cross-References

- ▸ Burrows-Wheeler Transform
- ▸ Compressed Suffix Array
- ▸ Suffix Trees and Arrays
- ▸ Suffix Tree Construction

Recommended Reading

1. Abouelhoda MI, Kurtz S, Ohlebusch E (2004) Replacing suffix trees with enhanced suffix arrays. J Discret Algorithms 2(1):53–86
2. Bingmann T, Fischer J, Osipov V (2013) Inducing suffix and LCP arrays in external memory. In: Sanders P, Zeh N (eds) Proceedings of the 15th meeting on algorithm engineering and experiments (ALENEX), New Orleans. SIAM, pp 88–102
3. Farach-Colton M, Ferragina P, Muthukrishnan S (2000) On the sorting-complexity of suffix tree construction. J ACM 47(6):987–1011
4. Gonnet G, Baeza-Yates R, Snider T (1992) New indices for text: PAT trees and PAT arrays. In: Frakes WB, Baeza-Yates R (eds) Information retrieval: data structures & algorithms. Prentice-Hall, Englewood Cliffs
5. Kärkkäinen J, Kempa D (2014) Engineering a lightweight external memory suffix array construction algorithm. In: Iliopoulos CS, Langiu A (eds) Proceedings of the 2nd international conference on algorithms for big data (ICABD), Palermo, pp 53–60
6. Kärkkäinen J, Kempa D (2014) LCP array construction in external memory. In: Gudmundsson J, Katajainen J (eds) Proceedings of the 13th symposium on experimental algorithms (SEA), Copenhagen. Lecture notes in computer science, vol 8504. Springer, pp 412–423
7. Kärkkäinen J, Sanders P (2003) Simple linear work suffix array construction. In: Baeten JCM, Lenstra JK, Parrow J, Woeginger GJ (eds) Proceedings of the 30th international conference on automata, languages and programming (ICALP), Eindhoven. Lecture notes in computer science, vol 2719. Springer, pp 943–955
8. Kärkkäinen J, Sanders P, Burkhardt S (2006) Linear work suffix array construction. J ACM 53(6):918–936
9. Kärkkäinen J, Manzini G, Puglisi SJ (2009) Permuted longest-common-prefix array. In: Kucherov G, Ukkonen E (eds) Proceedings of the 20th annual symposium on combinatorial pattern matching (CPM), Lille. Lecture notes in computer science, vol 5577. Springer, pp 181–192
10. Karp RM, Miller RE, Rosenberg AL (1972) Rapid identification of repeated patterns in strings, trees and arrays. In: Proceedings of the 4th annual ACM symposium on theory of computing (STOC), Denver. ACM, pp 125–136
11. Kasai T, Lee G, Arimura H, Arikawa S, Park K (2001) Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Proceedings of the 12th annual symposium on combinatorial pattern matching (CPM), Jerusalem. Lecture notes in computer science, vol 2089. Springer, pp 181–192
12. Kim DK, Sim JS, Park H, Park K (2005) Constructing suffix arrays in linear time. J Discret Algorithms 3(2–4):126–142
13. Ko P, Aluru S (2005) Space efficient linear time construction of suffix arrays. J Discret Algorithms 3(2–4):143–156
14. Larsson NJ, Sadakane K (2007) Faster suffix sorting. Theor Comput Sci 387(3):258–272
15. Manber U, Myers G (1993) Suffix arrays: a new method for on-line string searches. SIAM J Comput 22(5):935–948
16. Nong G (2013) Practical linear-time O(1)-workspace suffix sorting for constant alphabets. ACM Trans Inf Syst 31(3):Article 15, 15 pages
17. Nong G, Zhang S, Chan WH (2011) Two efficient algorithms for linear time suffix array construction. IEEE Trans Comput 60(10):1471–1484
18. Puglisi SJ, Smyth WF, Turpin A (2007) A taxonomy of suffix array construction algorithms. ACM Comput Surv 39(2):Article 4, 31 pages

Suffix Tree Construction

Jens Stoye
Faculty of Technology, Genome Informatics, Bielefeld University, Bielefeld, Germany

Keywords

Full-text index construction

Years and Authors of Summarized Original Work

1973; Weiner
1976; McCreight

1995; Ukkonen
2000; Farach-Colton, Ferragina, Muthukrishnan

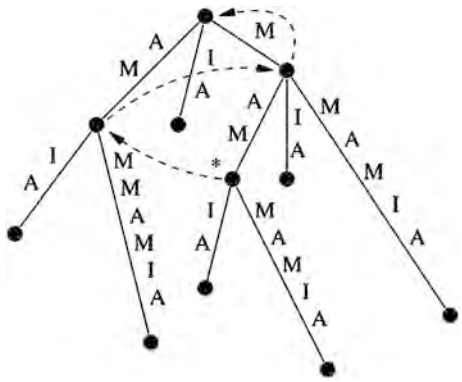
Problem Definition

The suffix tree is perhaps the best-known and most-studied data structure for string indexing with applications in many fields of sequence analysis. After its invention in the early 1970s, several approaches for the efficient construction of the suffix tree of a string have been developed for various models of computation. The most prominent of those that construct the suffix tree in main memory are summarized in this entry.

Notations

Given an alphabet Σ, a *trie* over Σ is a rooted tree whose edges are labeled with strings over Σ such that no two labels of edges leaving the same vertex start with the same symbol. A trie is *compacted* if all its internal vertices, except possibly the root, are branching. Given a finite string $S \in \Sigma^n$, the *suffix tree* of S, $T(S)$, is the compacted trie over Σ such that the concatenations of the edge labels along the paths from the root to the leaves are the suffixes of S. An example is given in Fig. 1.

The concatenation of the edge labels from the root to a vertex v of $T(S)$ is called the *path-label* of v, $P(v)$. For example, the path label of the vertex indicated by the asterisk in Fig. 1 is $P(*) = \text{MAM}$.



Suffix Tree Construction, Fig. 1 The suffix tree for the string $S = \text{MAMMAMIA}$. *Dashed arrows* denote suffix links that are employed by all efficient suffix tree construction algorithms

Constraints

The time complexity of constructing the suffix tree of a string S of length n depends on the size of the underlying alphabet Σ. It may be constant, it may be the alphabet of integers $\Sigma = \{1, 2, \ldots, n\}$, or it may be an arbitrary finite set whose elements can be compared in constant time. Note that the latter case reduces to the previous one if one maps the symbols of the alphabet to the set $\{1, \ldots, n\}$, though at the additional cost of sorting Σ.

Problem 1 (suffix tree construction)

INPUT: A finite string S of length n over an alphabet Σ.
OUTPUT: The suffix tree $T(S)$.

If one assumes that the outgoing edges at each vertex are lexicographically sorted, which is usually the case, the suffix tree allows retrieving the sorted order of S's characters in linear time. Therefore, suffix tree construction inherits the lower bounds from the problem complexity of sorting: $\Omega(n \log n)$ in the general alphabet case and $\Omega(n)$ for integer alphabets.

Key Results

Theorem 1 *The suffix tree of a string of length n can be represented in $O(n \log n)$ bits of space.*

This is easy to see since the number of leaves of $T(S)$ is at most n, and so is the number of internal vertices that, by definition, are all branching, as well as the number of edges. In order to see that each edge label can be stored in $O(\log n)$ bits of space, note that an edge label is always a substring of S. Hence it can be represented by a pair (l, r) consisting of *left pointer* l and *right pointer* r, if the label is $S[l, r]$.

Note that this space bound is not optimal since there are $|\Sigma|^n$ different strings and hence suffix trees, while $n \log n$ bits would allow to represent $n!$ different entities.

Theorem 2 *Suffix trees can be constructed in optimal time, in particular:*

1. For constant-size alphabet, the suffix tree $T(S)$ of a string S of length n can be constructed in $O(n)$ time [11–13]. For general alphabet, these algorithms require $O(n \log n)$ time.
2. For integer alphabet, the suffix tree of S can be constructed in $O(n)$ time [4,9].

Generally, there is a natural strategy to construct a suffix tree: Iteratively all suffixes are inserted into an initially empty structure. Such a strategy will immediately lead to a linear-time construction algorithm if each suffix can be inserted in constant time. Finding the correct position where to insert a suffix, however, is the main difficulty of suffix tree construction.

The first solution for this problem was given by Weiner in his seminal 1973 paper [13]. His algorithm inserts the suffixes from shortest to longest, and the insertion point is found in amortized constant time for constant-size alphabet, using rather a complicated amount of additional data structures. A simplified version of the algorithm was presented by Chen and Seiferas [3]. They give a cleaner presentation of the three types of links that are required in order to find the insertion points of suffixes efficiently, and their complexity proof is easier to follow. Since the suffix tree is constructed while reading the text from right to left, these two algorithms are sometimes called *anti-online* constructions.

A different algorithm was given in 1976 by McCreight [11]. In this algorithm the suffixes are inserted into the growing tree from longest to shortest. This simplifies the update procedure, and the additional data structure is limited to just one type of link: an internal vertex v with path label $P(v) = aw$ for some symbol $a \in \Sigma$ and string $w \in \Sigma^*$ has a *suffix link* to the vertex u with path label $P(u) = w$. In Fig. 1, suffix links are shown as dashed arrows. They often connect vertices above the insertion points of consecutively inserted suffixes, like the vertex with path-label "M" and the root, when inserting suffixes "MAMIA" and "AMIA" in the example of Fig. 1. This property allows reaching the next insertion point without having to search for it from the root of the tree, thus ensuring amortized constant time per suffix insertion. Note that since McCreight's algorithm treats the suffixes from longest to shortest and the intermediate structures are not suffix trees, the algorithm is not an online algorithm.

Another linear-time algorithm for constant-size alphabet is the online construction by Ukkonen [12]. It reads the text from left to right and updates the suffix tree in amortized constant time per added symbol. Again, the algorithm uses suffix links in order to quickly find the insertion points for the suffixes to be inserted. Moreover, since during a single update the edge labels of all leaf edges need to be extended by the new symbol, it requires a trick to extend all these labels in constant time: all the right pointers of the leaf edges refer to the same *end of string* value, which is just incremented.

An even stronger concept than online construction is *real-time* construction, where the worst-case (instead of amortized) time per symbol is considered. Amir et al. [1] present for general alphabet a suffix tree construction algorithm that requires $O(\log n)$ worst-case update time per every single input symbol when the text is read from right to left, and thus requires overall $O(n \log n)$ time, like the other algorithms for general alphabet mentioned so far. They achieve this goal using a binary search tree on the suffixes of the text, enhanced by additional pointers representing the lexicographic and the textual order of the suffixes, called *Balanced Indexing Structure*. This tree can be constructed in $O(\log\ n)$ worst-case time per added symbol and allows maintaining the suffix tree in the same time bound.

The first linear-time suffix tree construction algorithm for integer alphabets was given by Farach-Colton [4]. It uses the so-called *odd-even technique* that proceeds in three steps:

1. Recursively compute the compacted trie of all suffixes of S beginning at odd positions, called the *odd tree* T_o.
2. From T_o compute the *even tree* T_e, the compacted trie of the suffixes beginning at even positions in S.

3. Merge T_o and T_e into the whole suffix tree $T(S)$.

The basic idea of the first step is to encode pairs of characters as single characters. Since at most $n/2$ different such characters can occur, these can be radix-sorted and range-reduced to an alphabet of size $n/2$. Thus, the string S of length n over the integer alphabet $\Sigma = \{1, \ldots, n\}$ is translated in $O(n)$ time into a string S' of length $n/2$ over the integer alphabet $\Sigma' = \{1, \ldots, n/2\}$. Applying the algorithm recursively to this string yields the suffix tree of S'. After translating the edge labels from substrings of S' back to substrings of S, some vertices may exist with outgoing edges whose labels start with the same symbol, because two distinct symbols from Σ' may be pairs with the same first symbol from Σ. In such cases, by local modifications of edge labels or adding additional vertices, the trie property can be regained and the desired tree T_o is obtained.

In the second step, the odd tree T_o from the first step is used to generate the lexicographically sorted list (*lex-ordering* for short) of the suffixes starting at odd positions. Radix-sorting these with the characters at the preceding even positions as keys yields a lex-ordering of the even suffixes in linear time. Together with the longest common prefixes (lcps) of consecutive positions that can be computed in linear time from T_o using constant-time lowest common ancestor queries and the identity

$$\mathrm{lcp}(l_{2i}, l_{2j}) = \begin{cases} \mathrm{lcp}(l_{2i+1}, l_{2j+1}) + 1 & \text{if } S[2i] = S[2j] \\ 0 & \text{otherwise} \end{cases}$$

this ordering allows reconstructing the even tree T_e in linear time.

In the third step, the two tries T_o and T_e are merged into the suffix tree $T(S)$. Conceptually, this is a straightforward procedure: the two tries are traversed in parallel, and every part that is present in one or both of the two trees is inserted in the common structure. However, this procedure is simple only if edges are traversed character by character such that common and differing parts can be observed directly. Such a traversal would, however, require $O(n^2)$ time in the worst case, impeding the desired overall linear running time. Therefore, Farach-Colton suggests to use an oracle that tells for an edge of T_o and an edge of T_e the length of their common prefix.

However, the suggested oracle may overestimate this length, and that is why sometimes the tree generated must be corrected, called *unmerging*. The full details of the oracle and the unmerging procedure can be found in [4].

Overall, if $T(n)$ is the time it takes to build the suffix tree of a string $S \in \{1, \ldots, \mathrm{n}\}^n$, the first step takes $T(n/2) + O(n)$ time and the second and third steps take $O(n)$ time; thus the whole procedure takes $O(n)$ overall time on the RAM model.

Another linear-time construction of suffix trees for integer alphabets can be achieved via linear-time construction of suffix arrays together with longest common prefix tabulation, as described by Kärkkäinen and Sanders in [9].

All previously mentioned algorithms construct the suffix tree in main memory. However, since the data structure may become very large in practice, also methods for building the suffix tree in secondary memory have been studied. Possibly the simplest way is to first construct the suffix array A and the LCP array on disk, as described in the entry ▸ Suffix Array Construction. When this is done, it is only a small final step to construct the suffix tree [4]. The idea is to construct the tree in n phases from left to right, such that after phase i the suffix tree of the strings $A[1], \ldots, A[i]$ has been constructed. Simultaneously, an external-memory stack containing the nodes on the path leading from the root to $A[i]$ is maintained. In phase $i+1$, first, the leaf representing string $A[i+1]$ is created, and then all nodes are popped from the stack whose string length is strictly

greater than LCP[i]. Next, a new node with string depth LCP[i] is created (unless it already exists) whose parent is the top element of the stack and whose children are the last popped element and the new leaf. This new node and the new leaf are finally pushed on the stack. Keeping the two top pages of the stack in internal memory, the algorithm executes a total of $O(n)$ pop and push operations and therefore uses a total of $O(n/B)$ time, where B is the external memory block size.

Other more direct ways to construct the suffix tree on disk have also been developed, e.g., [14, 15].

In some applications the so-called *generalized* suffix tree of several strings is used, a dictionary obtained by constructing the suffix tree of the concatenation of the contained strings. An important question that arises in this context is that of dynamically updating the tree upon insertion and deletion of strings from the dictionary. More specifically, since edge labels are stored as pairs of pointers into the original string, when deleting a string from the dictionary, the corresponding pointers may become invalid and need to be updated. An algorithm to solve this problem in amortized linear time was given by Fiala and Greene [6], and a linear worst-case (and hence real-time) algorithm was given by Ferragina et al. [5].

Applications

The suffix tree supports many applications, most of them in optimal time and space, including exact string matching, set matching, longest common substring of two or more sequences, all-pairs suffix-prefix matching, repeat finding, and text compression. These and several other applications, many of them from bioinformatics, are given in [2] and [8].

Open Problems

Some theoretical questions regarding the expected size and branching structure of suffix trees under more complicated than i. i. d. sequence models are still open. Currently most of the research has moved toward more space-efficient data structures like suffix arrays and compressed string indices or the Burrows-Wheeler Transform.

Experimental Results

Suffix trees are infamous for their high memory requirements. The practical space consumption is between 9 and 11 times the size of the string to be indexed, even in the most space-efficient implementations known [7, 10]. Moreover, [7] also shows that suboptimal algorithms like the very simple quadratic-time *write-only top-down* (WOTD) algorithm can outperform optimal algorithms on many real-world instances in practice, if carefully engineered.

URLs to Code and Data Sets

Several sequence analysis libraries contain code for suffix tree construction. For example, Strmat (http://www.cs.ucdavis.edu/~gusfield/strmat.html) by Gusfield et al. contains implementations of Weiner's and Ukkonen's algorithm. An implementation of the WOTD algorithm by Kurtz can be found at (http://bibiserv.techfak.uni-bielefeld.de/wotd).

Cross-References

- Burrows-Wheeler Transform
- Compressed Suffix Trees
- Suffix Array Construction
- Suffix Trees and Arrays

Recommended Reading

1. Amir A, Kopelowitz T, Lewenstein M, Lewenstein N (2005) Towards real-time suffix tree construction. In: Proceedings of the 12th international symposium on string processing and information retrieval (SPIRE 2005). LNCS, vol 3772. Springer, Berlin, pp 67–78
2. Apostolico A (1985) The myriad virtues of subword trees. In: Apostolico A, Galil Z (eds) Combinatorial algorithms on words. NATO ASI Series, vol F12. Springer, Berlin, pp 85–96
3. Chen MT, Seiferas J (1985) Efficient and elegant subword tree construction. In: Apostolico A, Galil Z (eds) Combinatorial algorithms on words. Springer, New York
4. Farach-Colton M, Ferragina P, Muthukrishnan S (2000) On the sorting-complexity of suffix tree construction. J ACM 47(6):987–1011
5. Ferragina P, Grossi R, Montangero M (1998) A note on updating suffix tree labels. Theor Comput Sci 201:249–262
6. Fiala ER, Greene DH (1989) Data compression with finite windows. Commun ACM 32:490–505

7. Giegerich R, Kurtz S, Stoye J (2003) Efficient implementation of lazy suffix trees. Softw Pract Exp 33:1035–1049
8. Gusfield D (1997) Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, New York
9. Kärkkäinen J, Sanders P (2003) Simple linear work suffix array construction. In: Proceedings of the 30th international colloquium on automata, languages, and programming (ICALP 2003). LNCS, vol 2719. Springer, Berlin, pp 943–955
10. Kurtz S (1999) Reducing the space requirements of suffix trees. Softw Pract Exp 29:1149–1171
11. McCreight EM (1976) A space-economical suffix tree construction algorithm. J ACM 23:262–272
12. Ukkonen E (1995) On-line construction of suffix trees. Algorithmica 14:249–260
13. Weiner P (1973) Linear pattern matching algorithms. In: Proceedings of the 14th annual IEEE symposium on switching and automata theory. IEEE Press, New York, pp 1–11
14. Cheung C-F, Yu JX, Lu H (2005) Constructing suffix tree for gigabyte sequences with megabyte memory. IEEE Trans Knowl. Data Eng. 17:90–105
15. Tian Y, Tata S, Hankins RA, Patel JM (2005) Practical methods for constructing suffix trees. VLDB J 14:281–299

Suffix Tree Construction in Hierarchical Memory

Paolo Ferragina
Department of Computer Science, University of Pisa, Pisa, Italy

Keywords

Full-text index construction; String B-tree construction; Suffix array construction; Suffix tree construction

Years and Authors of Summarized Original Work

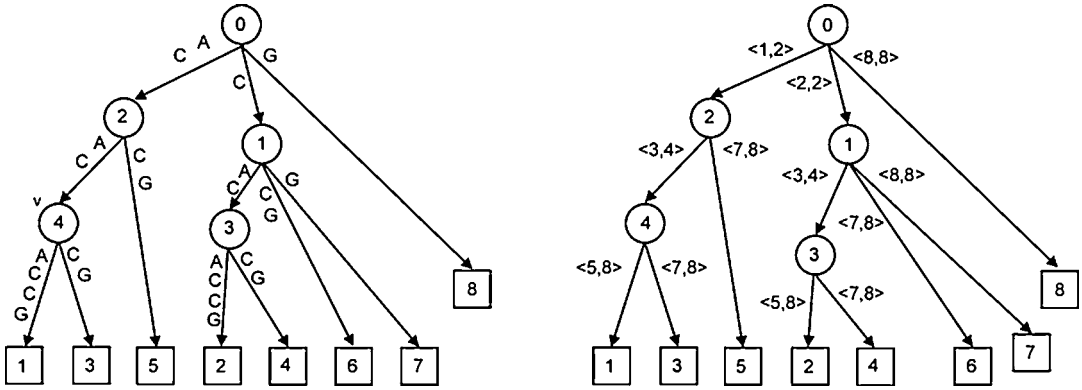
2000; Farach-Colton, Ferragina, Muthukrishnan

Problem Definition

The suffix tree is the ubiquitous data structure of combinatorial pattern matching myriad of situations – just to cite a few, searching, data compression and mining, and bioinformatics [7]. In these applications, the large data sets now available involve the use of numerous memory levels which constitute the storage medium of modern PCs: L1 and L2 caches, internal memory, multiple disks, and remote hosts over a network. The power of this memory organization is that it may be able to offer the expected access time of the fastest level (i.e., cache) while keeping the average cost per memory cell near the one of the cheapest level (i.e., disk), provided that data are properly *cached* and *delivered* to the requiring algorithms. Neglecting questions pertaining to the cost of memory references may even prevent the use of algorithms on large sets of input data. Engineering research is presently trying to improve the input/output subsystem to reduce the impact of these issues, but it is very well known [20] that the improvements achievable by means of a *proper arrangement of data* and a properly *structured algorithmic computation* abundantly surpass the best-expected technology advancements.

The Model of Computation

In order to reason about algorithms and data structures operating on hierarchical memories, it is necessary to introduce a *model of computation* that grasps the essence of real situations so that algorithms that are good in the model are also good in practice. The model considered here is the *external-memory model* [20], which received much attention because of its simplicity and reasonable accuracy. A computer is abstracted to consist of *two memory levels*: the internal memory of size M and the (unbounded) disk memory which operates by reading/writing data in blocks of size B (called *disk pages*). The performance of algorithms is then evaluated by counting (a) the number of disk accesses (I/Os), (b) the internal running time (CPU time), and (c) the number of disk pages occupied by the data structure or used by the algorithm as its working space. This simple model suggests, correctly, that a good external-memory algorithm should exploit both *spatial locality* and *temporal locality*. Of course, "I/O" and "two-level view" refer to any two levels



Suffix Tree Construction in Hierarchical Memory, Fig. 1 The suffix tree of $S = \texttt{ACACACCG}$ on the *left*, and its compact edge-encoding on the *right*. The endmarker # is not shown. Node v spells out the string *ACAC*. Each internal node stores the length of its associated string, and each leaf stores the starting position of its corresponding suffix

of the memory hierarchy with their parameters M and B properly set.

Notation

Let $S[1, n]$ be a string drawn from alphabet Σ, and consider the notation: S_i for the ith *suffix* of string S, $\texttt{lcp}(\alpha, \beta)$ for the *longest common prefix* between the two strings α and β, and $\texttt{lca}(u, v)$ for the *lowest common ancestor* between two nodes u and v in a tree.

The suffix tree of $S[1, n]$, denoted hereafter by $\mathcal{T}_S$, is a tree that stores all suffixes of $S\#$ in a compact form, where $\# \notin \Sigma$ is a special character (see Fig. 1). $\mathcal{T}_S$ consists of n leaves, numbered from 1 to n, and any root-to-leaf path spells out a suffix of $S\#$. The endmarker # guarantees that no suffix is the prefix of another suffix in $S\#$. Each internal node has at least two children and each edge is labeled with a nonempty substring of S. No two edges out of a node can begin with the same character, and sibling edges are ordered lexicographically according to that character. Edge labels are encoded with pairs of integers – say $S[x, y]$ is represented by the pair $\langle x, y \rangle$. As a result, all $\Theta(n^2)$ substrings of S can be represented in $O(n)$ optimal space by $\mathcal{T}_S$'s structure and edge encoding. Furthermore, the rightward scan of the suffix-tree leaves gives the ordered set of S's suffixes, also known as the *suffix array* of S [13]. Notice that the case of a large string collection $\Delta = \{S^1, S^2, \ldots, S^k\}$ *reduces to* the case of one long string $S = S^1 \#_1 S^2 \#_2 \cdots S^k \#_k$, where $\#_i \notin \Sigma$ are special symbols.

Numerous algorithms are known that build the suffix tree optimally in the RAM model (see [3] and references therein). However, most of them exhibit a marked absence of *locality of references* and thus elicit many I/Os when the size of the indexed string is too large to be fit into the internal memory of the computer. This is a serious problem because the slow performance of these algorithms can prevent the suffix tree being used even in medium-scale applications. This encyclopedia's entry surveys algorithmic solutions that deal efficiently with the *construction of suffix trees over large string collections* by executing an optimal number of I/Os. Since it is assumed that the edges leaving a node in $\mathcal{T}_S$ are lexicographically sorted, sorting is an obvious *lower bound* for building suffix trees (consider the suffix tree of a permutation!). The presented algorithms have sorting as their bottleneck, thus establishing that *the complexity of sorting and suffix tree construction match*.

Key Results

Designing a disk-efficient approach to suffix-tree construction has found efficient solutions only in the last few years [4]. The present section surveys

DIVIDE-AND-CONQUER ALGORITHM

(1) Construct the string $S'[j]$ = rank of $\langle S[2j], S[2j+1]\rangle$, and recursively compute $\mathcal{T}_{S'}$.
(2) Derive from $\mathcal{T}_{S'}$ the compacted trie $\mathcal{T}_O$ of all suffixes of S beginning at odd positions.
(3) Derive from $\mathcal{T}_O$ the compacted trie $\mathcal{T}_E$ of all suffixes of S beginning at even positions.
(4) Merge $\mathcal{T}_O$ and $\mathcal{T}_E$ into the whole suffix tree $\mathcal{T}_S$, as follows:
(4.1) Overmerge $\mathcal{T}_O$ and $\mathcal{T}_E$ into the tree $\mathcal{T}_M$.
(4.2) Partially unmerge $\mathcal{T}_M$ to get $\mathcal{T}_S$.

Suffix Tree Construction in Hierarchical Memory, Fig. 2 The algorithm that builds the suffix tree directly

SUFFIXARRAY-BASED ALGORITHM

(1) Construct the suffix array $\mathcal{A}_S$ and the array lcp_S of the string S.
(2) Initially set $\mathcal{T}_S$ as a single edge connecting the root to a leaf pointing to suffix $\mathcal{A}_S[1]$.
(2) For $i = 2,\ldots, n$:
(2.1) Create a new leaf ℓ_i that points to the suffix $\mathcal{A}_S[i]$.
(2.2) Walk up from ℓ_{i-1} until a node u_i is met whose string-length x_i is $\leq \text{lcp}_S[i]$.
(2.3) If $x_i = \text{lcp}_S[i]$, leaf ℓ_i is attached to u_i.
(2.4) If $x_i < \text{lcp}_S[i]$, create node u'_i with string-length x_i, attach it to u_i and leaf ℓ_i to u'_i;

Suffix Tree Construction in Hierarchical Memory, Fig. 3 The algorithm that builds the suffix tree passing through the suffix array

two theoretical approaches which achieve the best (optimal!) I/O-bounds in the worst case; the next section will discuss some practical solutions.

The first algorithm is based on a *Divide-and-Conquer* approach that allows us to reduce the construction process to external-memory sorting and few low-I/O primitives. It builds the suffix tree $\mathcal{T}_S$ by executing four (macro)steps, detailed in Fig. 2. It is not difficult to implement the first three steps in $\text{Sort}(n) = O(\frac{n}{B}\log_{M/B}\frac{n}{B})$ I/Os [20]. The last (merging) step is the most difficult one and its I/O-complexity bounds the cost of the overall approach. Farach-Colton et al. [3] propose an elegant merge for $\mathcal{T}_O$ and $\mathcal{T}_E$: substep (4.1) temporarily relaxes the requirement of getting $\mathcal{T}_S$ in one shot, and thus it blindly (over)merges the paths of $\mathcal{T}_O$ and $\mathcal{T}_E$ by comparing edges only via their first characters; then substep (4.2) refixes $\mathcal{T}_M$ by detecting and undoing in an I/O-efficient manner the (over)merged paths. Note that the time and I/O-complexity of this algorithm follow a nice recursive relation: $T(n) = T(n/2) + O(\text{Sort}(n))$.

Theorem 1 (Farach-Colton et al. [5]) *Given an arbitrary string $S[1,n]$, its suffix tree can be constructed in $O(\text{Sort}(n))$ I/Os, $O(n\log n)$ time and using $O(n/B)$ disk pages.*

The second algorithm [10] is deceptively simple, elegant, and I/O optimal and applies successfully to the construction of other indexing data structures, like the string Btree [5]. The key idea is to derive $\mathcal{T}_S$ from the suffix array $\mathcal{A}_S$ and from the *lcp* array, which stores the longest-common-prefix length of adjacent suffixes in $\mathcal{A}_S$. Its pseudocode is given in Fig. 3. Note that step (1) may deploy any external-memory algorithm for suffix array construction: used here is the elegant and optimal *Skew* algorithm of [9] which takes $O(\text{Sort}(n))$ I/Os. Step (2) takes a total of $O(n/B)$ I/Os by using a stack that stores the nodes on the current rightmost path of $\mathcal{T}_S$ in reversed order, i.e., leaf ℓ_i is on top. Walking upward, splitting edges or attaching nodes in $\mathcal{T}_S$ boils down to popping/pushing nodes from this stack. As a result, the time and I/O-complexity of this algorithm follow the recursive relation: $T(n) = T(2n/3) + O(\text{Sort}(n))$.

Theorem 2 (Kärkkäinen and Sanders 2003, see [10]) *Given an arbitrary string $S[1,n]$, its suffix tree can be constructed in $O(Sort(n))$ I/Os, $O(n \log n)$ time and using $O(n/B)$ disk pages.*

It is not evident which one of these two algorithms is better in practice [10]. The first one exploits a recursion with parameter 1/2 but incurs a large space overhead because of the management of the tree topology; the second one is more space efficient and easier to implement, but exploits a recursion with parameter 2/3.

Applications

The reader is referred to [4] and [7] for a long list of applications of large suffix trees and to [6, 18] for practical implementations.

Open Problems

The recent theoretical and practical achievements mean the idea that "suffix trees are not practical except when the text size to handle is so small that the suffix tree fits in internal memory" is no longer the case [15]. Given a suffix tree, it is known now (see, e.g., [4, 11]) how to map it onto a disk-memory system in order to allow I/O-efficient traversals for subsequent pattern searches. A fortiori, suffix-tree storage, and construction are challenging problems that need further investigation.

Space optimization is closely related to time optimization in a disk-memory system, so the design of *succinct* suffix-tree implementations is a key issue in order to scale to gigabytes of data in reasonable time. This topic is an active area of theoretical research with many fascinating solutions (see, e.g., [16] and the many papers that followed it), which need further exploration in the practical setting.

It is theoretically challenging to design a suffix-tree construction algorithm that takes optimal I/Os and space proportional to the *entropy* of the indexed string. The more compressible is the string, the lighter should be the space requirement of this algorithm. Some results are known [8, 11, 12], but both issues of compression and I/Os have been tackled jointly only recently [6], but more results are foreseen.

Experimental Results

The interest in building large suffix trees arose in the last few years because of the recent advances in sequencing technology, which have allowed the rapid accumulation of DNA and protein data. Some recent papers [1, 2, 9, 17, 18] proposed new practical algorithms that allow us to scale to Gbps/hours. Surprisingly enough, these algorithms are based on *disk-inefficient* schemes, but they properly select the insertion order of the suffixes and exploit carefully the internal memory as a buffer, so that their performance does not suffers significantly from the theoretical I/O-bottleneck.

In [9] the authors propose an *incremental* algorithm, called *PrePar*, which performs multiple passes over the string S and constructs the suffix tree for a *subrange* of suffixes at each pass. For a user-defined parameter q, a suffix subrange is defined as the set of suffixes prefixed by the same q-long string. Suffix subranges induce subtrees of $\mathcal{T}_S$ which can thus be built independently and evicted from internal memory as they are completed. The experiments reported in [9] successfully index 286 Mbps using 2 Gb internal memory.

In [2] the authors propose an improved version of *PrePar*, called *DynaCluster*, that deploys a *dynamic* technique to identify suffix subranges. Unlike *Prepar*, *DynaCluster* does not scan over and over the string S, but it starts from the q-based subranges and then splits them recursively in a DFS-manner if their size is larger than a fixed threshold τ. Splitting is implemented by looking at the next q characters of the suffixes in the subrange. This clustering and lazy-DFS visit of $\mathcal{T}_S$ significantly reduce the number of I/Os incurred by the frequent edge-splitting operations that occur during the suffix-tree construction process and allow it to cope efficiently with skew data.

As a result, *DynaCluster* constructs suffix trees for 200 Mbps with only 16 Mb internal memory.

In [17] authors improved the space requirement and the buffering efficiency, thus being able to construct a suffix tree of 3 Gbps in 30 h, whereas [1] improved the I/O behavior of RAM-algorithms for online suffix-tree construction, by devising a novel low-overhead buffering policy. More recently [14] introduced a new technique, called Elastic Range (ERA), which partitions the tree construction process horizontally and vertically and minimizes I/Os by dynamically adjusting the horizontal partitions independently for each vertical partition, based on the evolving shape of the tree and the available internal memory. This technique is specialized to work also for shared-memory and shared-disk multi-core systems and for parallel shared-nothing architectures. ERA indexes the entire human genome in 19 min on a commodity desktop PC. For comparison, the fastest existing method needs 15 min using 1024 CPUs on an IBM BluGene supercomputer.

Finally [19] observed that increasing memory sizes of current commodity PCs and servers enhance the impact of in-memory tasks on performance. So it is imperative nowadays to reassess the performance of in-memory algorithms and to propose new algorithms that incorporate the characteristics of modern hardware architectures, such as multilevel memory hierarchy and chip multiprocessors (CMPs). Starting from these premises the authors proposed cache-conscious suffix-tree construction algorithms that are tailored to CMP architectures, using novel sample-based cache-partitioning techniques that improved cache performance and exploited on-chip parallelism of CMPs thus achieving satisfactory speedups with increasing number of cores.

Cross-References

- ▸ Cache-Oblivious Sorting
- ▸ Suffix Array Construction
- ▸ Suffix Tree Construction
- ▸ Text Indexing

Recommended Reading

1. Bedathur SJ, Haritsa JR (2004) Engineering a fast online persistent suffix tree construction. In: Proceedings of the 20th international conference on data engineering, Boston, pp 720–731
2. Cheung C, Yu J, Lu H (2005) Constructing suffix tree for gigabyte sequences with megabyte memory. IEEE Trans Knowl Data Eng 17:90–105
3. Farach-Colton M, Ferragina P, Muthukrishnan S (2000) On the sorting-complexity of suffix tree construction. J ACM 47:987–1011
4. Ferragina P (2005) Handbook of computational molecular biology. In: Computer and information science series, ch. 35 on "String search in external memory: algorithms and data structures". Chapman & Hall/CRC, Florida
5. Ferragina P, Grossi R (1999) The string Btree: a new data structure for string search in external memory and its applications. J ACM 46:236–280
6. Ferragina P, Gagie T, Manzini G (2012) Lightweight data indexing and compression in external memory. Algorithmica 63(3):707–730
7. Gusfield D (1997) Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge University Press, Cambridge
8. Hon W, Sadakane K, Sung W (2009) Breaking a time-and-space barrier in constructing full-text indices. SIAM J Comput 38(6):2162–2178
9. Hunt E, Atkinson M, Irving R (2002) Database indexing for large DNA and protein sequence collections. Int J Very Large Data Bases 11:256–271
10. Kärkkäinen J, Sanders P, Burkhardt S (2006) Linear work suffix array construction. J ACM 53:918–936
11. Ko P, Aluru S (2007) Optimal self-adjusting trees for dynamic string data in secondary storage. In: Symposium on string processing and information retrieval (SPIRE), Santiago. LNCS, vol 4726, pp 184–194. Springer, Berlin
12. Mäkinen V, Navarro G (2008) Dynamic entropy-compressed sequences and full-text indexes. ACM Trans Algorithm 4(3)
13. Manber U, Myers G (1993) Suffix arrays: a new method for on-line string searches. SIAM J Comput 22:935–948
14. Mansour E, Allam A, Skiadopoulos S, Kalnis P (2011) ERA: efficient serial and parallel suffix tree construction for very long strings. PVLDB 5(1):49–60
15. Navarro G, Baeza-Yates R (2000) A hybrid indexing method for approximate string matching. J Discr Algorithms 1:21–49
16. Navarro G, Mäkinen V (2007) Compressed full text indexes. ACM Comput Surv 39(1): Article no 2
17. Tian Y, Tata S, Hankins RA, Patel JM (2005) Practical methods for constructing suffix trees. VLDB J 14(3):281–299

18. Thomo A, Barsky M, Stege U (2010) A survey of practical algorithms for suffix tree construction in external memory. Softw Pract Experience 40(11):965–988
19. Tsirogiannis D, Koudas N (2010) Suffix tree construction on modern hardware. In: Proceedings of the 13th international conference on extending database technology (EDBT), Lausanne, pp 263–274
20. Vitter J (2002) External memory algorithms and data structures: dealing with MASSIVE DATA. ACM Comput Surv 33:209–271

Suffix Trees and Arrays

Alberto Apostolico[1] and Fabio Cunial[2]
[1]College of Computing, Georgia Institute of Technology, Atlanta, GA, USA
[2]Department of Computer Science, Helsinki Institute for Information Technology (HIIT), University of Helsinki, Helsinki, Finland

Keywords

Full-text indexing; Pattern matching; String searching; Suffix array; Suffix tree

Years and Authors of Summarized Original Work

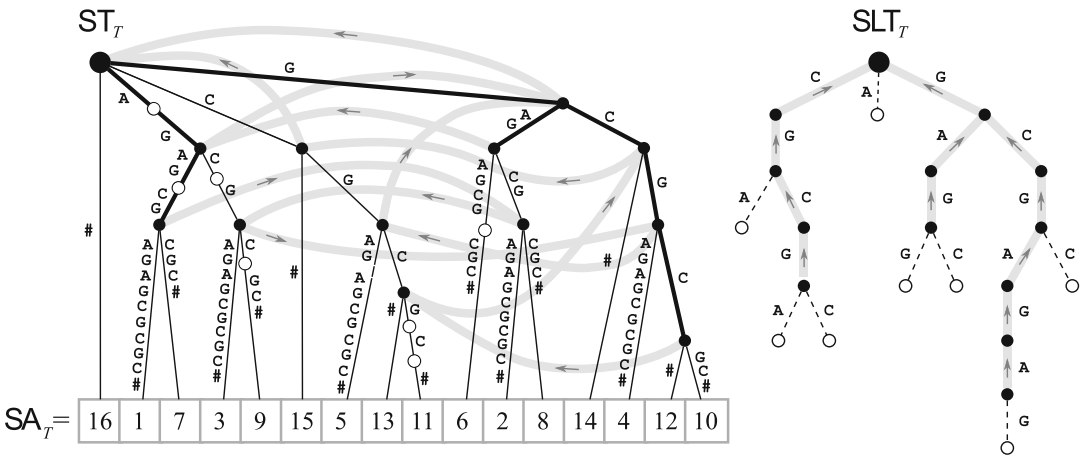
1973; McCreight
1973; Weiner
1993; Manber, Myers
1995; Ukkonen

The suffix tree is one of the oldest full-text inverted indexes and one of the most persistent subjects of study in the theory of algorithms. With extensions and refinements, including succinct and compressed variants that provide some of its expressive power in smaller space, it constitutes a fundamental conceptual tool in the design of string algorithms. The companion structure represented by the suffix array is as powerful as the suffix tree in many applications, but it requires significantly less space. The uses of these data structures are so numerous that it is difficult to account for all of them, while even more are being discovered. Salient applications include searching for a pattern in a text in time proportional to the size of the pattern, various computations on regularities such as repeats and palindromes within a text, statistical tables of substring occurrences, data compression by textual substitution, as well as ancillary yet fundamental tasks in string searching with errors, and more.

Problem Definition

It is well known that searching among n keys in an unsorted table takes optimal linear time. When multiple searches are expected, however, it becomes worth to sort the table once and for all, whereby each subsequent search will require only logarithmic time. It is similarly possible to build an inverted index on a long text so that the search for any query string will take time proportional to the length of the query rather than that of the text. It turns out that the data structures built for this purpose support many more applications, which are the topic of this entry.

Formally, let T be a string of length n on alphabet $\Sigma = [1 \dots \sigma]$, let $\underline{T}$ be its reverse, and let $\# \notin \Sigma$ be a shorthand for zero. To simplify the exposition, we assume throughout that σ is a constant. The *suffix tree* $\mathsf{ST}_T = (\perp, V, E)$ of T is a tree rooted at node $\perp \in V$ with set of nodes V and set of labeled edges E (Fig. 1, left). Edge labels are pointers to substrings of $T\#$: we denote by $\ell(e)$, and equivalently by $\ell(u, v)$, the label of edge $e = (u, v) \in E$, and we denote by $\ell(v)$ the string $\ell(\perp, v_1) \cdot \ell(v_1, v_2) \cdot \dots \cdot \ell(v_{k-1}, v)$, where $\perp, v_1, v_2, \dots, v_{k-1}, v$ is a path in ST_T. We say that node v has *string depth* $|\ell(v)|$. Let $v \in V$ be an internal node, and let $w_1, w_2, \dots, w_k$ be its children: then, $2 \leq k \leq \sigma + 1$, and labels $\ell(v, w_1), \ell(v, w_2), \dots, \ell(v, w_k)$ start with distinct characters. The children of v are ordered lexicographically according to the labels of edges $(v, w_1), (v, w_2), \dots, (v, w_k)$. There is a bijection between the leaves of ST_T and the suffixes of $T\#$, so every leaf is annotated with the starting position of its corresponding suffix. Moreover, if leaf $v \in V$ is associated with the suffix that



Suffix Trees and Arrays, Fig. 1 Relationship between the suffix tree, the suffix array (*left*), and the suffix-link tree (*right*) of string $T = \texttt{AGAGCGAGAGCGCGC\#}$. *Thin black lines*, edges of ST_T; *thick gray lines*, suffix links; *thin dashed lines*, implicit Weiner links; *thick black lines*, the subtree of ST_T induced by maximal repeats. *Black dots*, nodes of ST_T; *large black dot*, $\perp$; *white dots*, destinations of implicit Weiner links. *Squares*, leaves of ST_T and cells of SA_T; numbers, starting position of each suffix in T. For clarity, implicit Weiner links are not overlaid to ST_T, and suffix links from the leaves of ST_T are not drawn

starts at position i, then $\ell(v) = T[i \ldots n]\#$. Since ST_T has exactly $n + 1$ leaves and every internal node has at least two children, there are at most n internal nodes; thus, ST_T takes $O(n)$ space. We drop the subscript from ST whenever the underlying string is clear from the context.

A substring W of $T\#$ is called *right maximal* if both Wa and Wb occur in T, with $\{a, b\} \subseteq \Sigma \cup \{\#\}$ and $a \neq b$. Clearly a substring W is right maximal iff $W = \ell(v)$ for some $v \in V$. Moreover, assume that $\ell(v) = aW$ for some $v \in V$, $a \in \Sigma$, and $W \in \Sigma^*$. Since aW is right maximal, string W is right maximal as well; therefore, there is a node $w \in V$ with $\ell(w) = W$. Thus, the set of labels $\{\ell(v) : v \in V\}$ enjoys the *suffix closure* property, in the sense that if a string W belongs to the set so does every one of its suffixes. We say that there is a *suffix link from v to w labeled by a*, and we write $\texttt{suffixLink}(v) = w$. Clearly, if v is a leaf, then $\texttt{suffixLink}(v)$ is either a leaf or $\perp$. The graph induced by V and by suffix links is a trie rooted at $\perp$: such trie is called the *suffix-link tree* SLT_T of string T (Fig. 1, right). Inverting the direction of all suffix links yields the so-called *explicit Weiner links*. Given a node v and a symbol $a \in \Sigma$, it might happen that string $a\ell(v)$ does occur in T but that it is not the label of any node in V: all such left extensions of nodes in V that end in the middle of an edge of ST are called *implicit Weiner links*. A node in V can have more than one outgoing Weiner link, and all such Weiner links have different labels. The number of suffix links (or, equivalently, of explicit Weiner links) is upper-bounded by $2n-2$, and the same bound holds for the number of implicit Weiner links: in some applications, we thus assume that ST is augmented with unary nodes that correspond to all the destinations of implicit Weiner links. A substring W of $T\#$ is called *left maximal* if both aW and bW occur in $T\#$, with $\{a, b\} \subseteq \Sigma \cup \{\#\}$ and $a \neq b$, where $T\#$ is interpreted as a circular string. A string that is both left and right maximal is called *maximal repeat*. The set of all left-maximal strings enjoys the *prefix closure* property; therefore, there is a bijection between the maximal repeats and the nodes that lie in some paths of ST that start from the root (Fig. 1, left).

The *suffix array* $\mathsf{SA}_T[1 \ldots n + 1]$ of string T is the permutation of $[1 \ldots n + 1]$ such that $\mathsf{SA}_T[k] = i$ iff suffix $T[i \ldots n]\#$ has position k in the list of all suffixes of $T\#$ taken in lexicographic order. In this case, we say that suffix $T[i \ldots n]\#$ has *lexicographic rank k*. Clearly

$\mathsf{SA}_T[1] = n + 1$. The *inverse suffix array* of string T is an array $R_T[1 \ldots n + 1]$ such that $R_T[\mathsf{SA}[i]] = i$ for all $i \in [1 \ldots n + 1]$. A substring W of $T\#$ corresponds to a unique, contiguous *interval* (i_W, j_W) of SA_T, which contains all the suffixes of $T\#$ that are prefixed by W. An additional structure that complements the suffix array in many applications is the *longest common prefix array* $\mathsf{LCP}_T[2 \ldots n + 1]$, which stores at position i the length of the longest prefix shared by suffix $T[\mathsf{SA}_T[i] \ldots n]\#$ and by suffix $T[\mathsf{SA}_T[i - 1] \ldots n]\#$. Clearly $\mathsf{LCP}_T[k] \geq |W|$ for all $k \in [i_W + 1 \ldots j_W]$. Again, we drop the subscript from SA, R, and LCP whenever the underlying string is clear from the context.

Suffix tree, suffix array, and LCP array are strongly intertwined, and they have connections to other substring recognizers, like the *directed acyclic word graph* (DAWG) and its compact variant (CDAWG). SA can be thought of as the ordered set of leaves of ST, and ST can be thought of as a search tree built on top of SA (Fig. 1, left). The full ST, including suffix links, can be built from SA and LCP with a $O(n)$-time scan [1], and SA can be built from ST with a $O(n)$ traversal. LCP itself can be built from SA in $O(n)$ time [18]. A number of ingenious algorithms have been proposed to build ST and SA in linear time directly from the string itself, even in the case of polynomial alphabets: see [10, 17, 19, 20, 25, 32, 33] for a sampler of such algorithms, and see [28] for a detailed taxonomy. Some applications require to maintain the suffix tree after edits to the underlying string: see [12, 13, 22] for a sampler of such algorithms. Finally, see [21] for a comparative study of space-efficient allocations of suffix trees.

Key Results

Suffix trees are extremely versatile indexes that allow one to solve a variety of string matching and analysis problems [2, 9, 14]. We review few such problems, classifying the corresponding algorithmic solutions based on the way they walk on the suffix tree and on the information they store in each node. This classification exposes recurrent design patterns, it highlights which parts of the suffix tree are needed by each application, and it helps decide which algorithms can be implemented on top of more succinct but less powerful representations of the suffix tree. The emphasis of this section is on the power of different traversals of the suffix tree, not necessarily on the most efficient solution of each string analysis problem.

Top-Down

Exact searching inside a string S of length n is the most natural example of top-down traversal of ST_S. Given a query string W, we can just match its characters from the root of ST in $O(|W|)$ time to determine whether W occurs in S or not. Since edges are labeled by substrings of S, the search for W can end in the middle of an edge (u, v): we say that v is the *locus* of W in ST, and we denote it by $\texttt{locus}(W)$. This approach generalizes to a set of patterns $W_1, W_2, \ldots, W_k$ of total length m, by building the suffix tree of the concatenation $W = W_1\#_1 W_2\#_2 \cdots \#_{k-1} W_k$ and by traversing ST_S and ST_W synchronously, where $i \neq j$ implies $\#_i \neq \#_j$ and $\#_i \neq \#$.

The total number of (possibly overlapping) occurrences of the label $\ell(v)$ of a node v of ST equals the number of leaves in the subtree rooted at v, which can be computed by a bottom-up traversal of the tree. All strings that end in the middle of edge (u, v) start exactly at the same positions as $\ell(v)$ in S; therefore, ST with frequency annotation allows one to return the frequency in S of any string W in $O(|W|)$ time. An important consequence of this is the fact that the number of *distinct frequencies* assumed by nonempty substrings of S is at most $|S|$. It is also possible to annotate every node of ST with the smallest and largest leaf in its subtree, supporting $O(|W|)$-time queries on the first and last occurrence in S of any string W. More generally, traversing the tree rooted at $\texttt{locus}(W)$ in $O(|W| + k)$ time allows one to print all the k starting positions of W in S.

Finding all the occurrences of W in S can also be done in $O(|W| \log n)$ time, by binary searching SA_S for strings $W\#$ and $W\$$, where $\$ = \sigma + 1$: the result of these searches are, respectively, the starting and ending position of the interval of all suffixes prefixed by W. Knowing this interval allows one to derive the number of occurrences of W in S in constant time and to output the starting positions of such occurrences in time linear in the size of the output. Using simple properties of LCP_S, it is possible to reduce the time of binary search to $O(|W| + \log n)$, by reusing information during the search [24].

The top-down navigation of a suitably annotated suffix tree of S allows one also to compute the Lempel-Ziv factorization of S [23]. Recall that this factorization scans the string from left to right, and it determines at every position i the longest prefix of $S[i \ldots n]$ that equals a prefix of $S[j \ldots n]$, where $j < i$. Let W be such longest prefix: the factorization outputs the tuple $(j, |W|, S[i + |W|])$. Clearly we can find all this information by annotating every node v of ST with the index j of the smallest leaf in the subtree rooted at v. Then, we can just match suffix $S[i \ldots n]$ from the root of ST until a mismatch occurs or until we find a node with index greater than i. More advanced solutions embed the factorization in an online, one-pass construction of ST [29].

Bottom-Up

A *square* is a string WW where $W \in \Sigma^+$ is not in the form Z^k with $k > 1$ for any $Z \in \Sigma^+$. Clearly, if a square WW occurs at position i in S, then there is a node v in ST_S such that $|\ell(v)| \geq |W|$ and such that leaves i and $i + |W|$ belong to the subtree rooted at v. The converse is also true [4]. Thus, we can output all the repeats of S by using the following bottom-up traversal of ST. Assume without loss of generality that all nodes in ST have exactly two children. Every node u of ST builds its list of occurrences, sorted by position in S, using the lists of its children. Then, it scans its list once to find all pairs of positions at distance at most $|\ell(v)|$ in S that are consecutive in the list: every such pair is a square, and positions at distance at most $|\ell(v)|$ that are not consecutive induce squares that are implied by the consecutive positions.

Let v and w be the two children of v, and assume without loss of generality that the list of occurrences of v is smaller than the list occurrences of w. Then, the list of node u can be built by extracting all elements from the list of node v and by inserting them into the list of node w. As a consequence of such insertions, the occurrences in the list of v move to a list that is at least twice the size of the original list: it follows that an occurrence can be pushed into at most $O(\log n)$ lists; therefore, the total number of extractions and insertions is bounded by $O(n \log n)$. If the lists of occurrences are implemented with balanced trees, the total time to extract all squares from S is $O(n \log^2 n)$. More advanced approaches manage to shave a logarithm, reaching optimal $O(n \log n)$ time [4], and to reduce the complexity to $O(n + \tau)$, where τ is the size of the output [15, 31].

The algorithm for detecting squares can be adapted to compute all the *maximal palindromes* of S, by applying it to string $T = S\#\underline{S}\$$. Note that a variant of the same algorithm can be implemented using the suffix array. First, it is easy to see that a bottom-up, in-order traversal of the internal nodes of ST_S can be simulated by a linear scan of SA_S and of LCP_S, maintaining a stack [1]. It follows that, for every interval (i_v, j_v) in SA of a node v in ST, we can just check whether $\mathsf{SA}[k] + |\ell(v)| \in [i_v \ldots j_v]$ and $S[\mathsf{SA}[k] + |\ell(v)|] \neq S[\mathsf{SA}[k] + 2|\ell(v)|]$, for every $k \in [i_v \ldots j_v]$: in this case, the occurrence of square $\ell(v)$ at position $\mathsf{SA}[k]$ is called *branching*. It is easy to see that all squares can be derived from squares with branching occurrences [31]. Moreover, if the occurrence at position $\mathsf{SA}[k]$ is branching, then suffixes $\mathsf{SA}[k] + |\ell(v)|$ and $\mathsf{SA}[k] + 2|\ell(v)|$ belong to distinct children of node v in ST: we can thus discard the child w of v with the largest number of leaves and check for every $k \in [i \ldots j]$ that does not belong to the interval of w whether $\mathsf{SA}[k] - |\ell(v)| \in [i_v \ldots j_v]$ and $S[\mathsf{SA}[k]] \neq S[\mathsf{SA}[k] + |\ell(v)|]$. The child of v with largest interval can be determined in constant time during the simulated bottom-up traversal of ST, and since the largest

interval is always excluded, the algorithm runs in $O(n \log n)$ time.

Given a collection of k strings of total length n, let S be the concatenation of all such strings, each terminated by a distinct symbol that does not belong to Σ. A bottom-up navigation of ST_S (called also the *generalized suffix tree* of the collection) allows one to compute the length of a longest string that occurs in $x \leq k$ strings. To solve this problem, we can annotate each leaf v of ST with a bitvector `which` of length k, such that `which`$[i] = 1$ iff the suffix associated with v starts inside string i. Then, every node of ST can be annotated with the same bitvector via a bottom-up, $O(nk)$ traversal, in which we compute the bitvector of a node by taking the logical `or` of the bitvectors of its children. More advanced algorithms solve this problem in $O(n)$ time [8]. As a byproduct, this annotation allows one to answer queries on the number of strings in the collection that contain a given substring, a problem known as *document counting*. A germane problem is that of *document listing*, in which we are given a pattern and we are asked to return the set of all documents that contain one or more copies of the pattern [26].

Top-Down and Suffix Links

Given two strings S and T, of length n and m, respectively, the *matching statistics array* $\mathsf{MS}_{S,T}[1 \ldots n]$ is such that $\mathsf{MS}_{S,T}[i]$ stores the length of the longest string that starts at position i in S and that occurs in T [33]. We can compute $\mathsf{MS}_{S,T}$ by scanning S from left to right, while simultaneously issuing child and suffix-link queries on ST_T. This results in a peculiar walk on ST_T that consists of alternating sequences of suffix-tree edges and of suffix links (we can also compute $\mathsf{MS}_{S,T}$ symmetrically, by scanning S from right to left and by simultaneously issuing parent and Weiner-link queries on ST_T [27]).

Specifically, assume that we are at position i in S, and let $W = S[i \ldots i + \mathsf{MS}_{S,T}[i] - 1]$. Note that W can end in the middle of an edge (u, v) of ST_T: let $W = aXY$ where $a \in \Sigma$, $X \in \Sigma^*$, $aX = \ell(u)$, and $Y \in \Sigma^*$. Moreover, let $u' =$ `suffixLink`(u) and $v' =$ `suffixLink`(v). Note that suffix links can project edge (u, v) onto a *path* $u', v_1, v_2, \ldots, v_k, v'$, where $v_j \in V$ for $j \in [1 \ldots k]$. Since $\mathsf{MS}_{S,T}[i+1] \geq \mathsf{MS}_{S,T}[i]-1$, the first step to compute $\mathsf{MS}_{S,T}[i + 1]$ is to find the position of XY in ST_T: we call this phase of the algorithm the *repositioning phase*. To implement the repositioning phase, it suffices to take the suffix link from u, to follow the outgoing edge from u' whose label starts by the first character of Y, and then to iteratively jump to the next internal node of ST_T and to choose the next outgoing edge according to the corresponding character of Y. After repositioning, we start matching the new characters of S on ST_T, i.e., we read characters $S[i + \mathsf{MS}_{S,T}[i]], S[i + \mathsf{MS}_{S,T}[i] + 1], \ldots$ until such an extension becomes impossible in ST_T. We call this phase of the algorithm the *matching phase*. Note that no character of S that has been read during the repositioning phase of $\mathsf{MS}_{S,T}[i + 1]$ will be read again during the repositioning phase of $\mathsf{MS}_{S,T}[i + k]$ with $k > 1$: it follows that every position j of S is consumed at most twice, once in the matching phase of some $\mathsf{MS}_{S,T}[i]$ with $i \leq j$ and once in the repositioning phase of some $\mathsf{MS}_{S,T}[k]$ with $i < k < j$. Since every mismatch can be charged to the position of which it concludes the matching statistics, the total number of mismatches encountered by the algorithm is bounded by the length of S.

These algorithms can be adapted to compute the *shortest unique substring array* $\mathsf{SUS}_S[1 \ldots n]$, which stores at index i the length of the shortest substring of S that occurs only at position i [33]. The average of the matching statistics vector can be used to estimate the cross-entropy of the probability distributions of two stationary, ergodic, stochastic processes with finite memory that generated S and T [11]. Moreover, a number of compositional similarity measures between two strings S and T can be computed by scanning S and by simultaneously navigating ST_T as in matching statistics: this has the advantage of building and annotating the suffix tree of just the shortest string [30]. Matching statistics on a suitably annotated suffix tree of T allows one also to approximate the probability that S was generated by the same variable-length Markov process that produced

T, another measure of similarity not based on sequence alignment [3].

Top-Down in the Suffix-Link Tree

A number of statistical applications require to annotate the nodes of ST_S with *empirical probabilities* rather than with raw frequencies. The empirical probability $p_S(W)$ of a string W is essentially the number of its occurrences $f_S(W)$ divided by the maximum number of occurrences that W can have in a string of length $|S| = n$. This number cannot exceed $n-|W|+1$, but it also depends on the number of overlaps that W has with itself, i.e., on the number of proper *borders* of W: thus, we set $p_S(W) = f_S(W)/b(W)$, where $b(W)$ is the length of the shortest period of W. Note that p_S can change inside an edge of ST. However, if we are interested only in the empirical probability of nodes of ST, we can compute all such values in overall linear time, by mapping the longest-border computation in the KMP algorithm onto a depth-first navigation *of the suffix-link tree* [5].

The exact computation of the *variance* of the frequency of a string W in S can be itself mapped onto the computation of the longest proper border of W. Under suitable statistical assumptions, computing the expectation and variance of the frequency of all right-maximal substrings of S suffices to detect all substrings of S with anomalous frequency: it is thus possible to discover all statistically frequent and rare substrings of S in overall linear time [5].

Any Order

A single pass over all nodes of ST *in any order*, coupled with a number of checks on the children and on the Weiner links of each node, suffices to solve a number of string analysis problems in linear time.

A string W is a *maximal unique match* (MUM) between two strings S and T if it occurs exactly once in S and exactly once in T and if neither aW nor Wb occur in both S and T for any $\{a, b\} \subseteq \Sigma$ (for simplicity, we disregard cases in which W occurs at the beginning or at the end of a string) [14]. Clearly W must be a right-maximal substring of $U = S\#T\$$, where # and \$ are separators not belonging to Σ. Therefore, we just need to iterate over every node v of ST_U in any order, checking the following conditions: (1) v has exactly two leaves as children; (2) the suffixes that correspond to such leaves start before and after position $|S| + 1$ in U, respectively; and (3) v has two Weiner links. A similar approach extends to MUMs of more than two strings, as well as to *maximal* (not necessarily unique) *exact matches* between two strings and to the *maximal repeats* [7] and the *minimal absent words* of a single string [16].

Symmetrically, it is easy to detect the MUMs of two strings S and T by a linear scan of the suffix array of $U = S\#T\$$ and of the corresponding LCP array. Indeed, a MUM corresponds to an interval $(i, i + 1)$ of size two in SA_U such that $\mathsf{LCP}_U[i] < \mathsf{LCP}_U[i + 1]$, $\mathsf{LCP}_U[i + 2] < \mathsf{LCP}_U[i + 1]$, $U[\mathsf{SA}_U[i] - 1] \neq U[\mathsf{SA}_U[i + 1] - 1]$, and $\mathsf{SA}_U[i] < |S| + 1 < \mathsf{SA}_U[i + 1]$. Similar criteria allow one to detect maximal repeats, *supermaximal repeats* [14], and maximal exact matches [1].

String Depth Annotation

Assume that every node v of ST_S is annotated with $|\ell(v)|$. Recall that the *shortest unique substring array* $\mathsf{SUS}_S[1 \ldots n]$ is such that $\mathsf{SUS}_S[i]$ is the length of the shortest substring of S that occurs only at position i. Since $S[i \ldots i + \mathsf{SUS}[i] - 1] = Wa$ where $a \in \Sigma$, since $\texttt{locus}(Wa)$ is a leaf v, and since $\texttt{locus}(W) = \texttt{parent}(v)$, traversing the nodes of ST in any order suffices to compute $\mathsf{SUS}[i]$ for every i. String depth annotations, coupled with a traversal of the nodes of ST in any order, suffice also to compute measures of compositional complexity of S, like the total number of distinct substrings, possibly of a fixed length k.

Frequency Annotation

Recall that $f_S(W)$ is the number of occurrences of string W in S. Assume that we want to compute $p(a|W) = f_S(Wa)/f_S(W)$ for all substrings W of S and for all characters $a \in \Sigma$ such that Wa is a substring of S. Such values are called *conditional probabilities*. Clearly $p(a|W) = 1$ if W ends in the middle of an edge of ST_S: it is thus sufficient to compute conditional probabilities

for the nodes of ST, and this can be done by traversing the nodes of ST in any order and by accessing their children.

String Depth and Frequency Annotation
Assume that every node v of ST_S is also annotated with the number of leaves in the subtree rooted at v. Then, traversing the nodes of ST in any order allows one to compute the longest substring of S that repeats at least τ times, or the most frequent string of length at least τ, for any user-specified threshold τ. String depth and frequency annotations, coupled with a traversal of the nodes of ST in any order, allow one also to compute the number of distinct substrings that occur τ times in S, for every frequency τ in a user-specified range.

Given a substring W of S, let $\texttt{right}(W)$ be the set of characters that occur in S after W. More formally, $\texttt{right}(W) = \{a \in \Sigma : f_S(Wa) > 0\}$. The *kth order empirical entropy* of S is defined as follows:

$$\mathsf{H}(S,k) = \frac{1}{|S|} \sum_{W \in \Sigma^k} \sum_{a \in \texttt{right}(W)} f_S(Wa) \log \left(\frac{f_S(W)}{f_S(Wa)} \right)$$

To compute $\mathsf{H}(S,k)$, it suffices again to traverse the nodes of ST_S in any order, to check whether $|\ell(v)| = k$, and to cumulate the contribution of v to $\mathsf{H}(S,k)$ by reading the frequency of its children. Strings of length k that end in the middle of an edge of ST do not contribute to $\mathsf{H}(S,k)$.

In a similar fashion, given a string S on alphabet Σ, let $\mathbf{S}$ be a vector indexed by all strings in Σ^k for a fixed $k > 0$, such that $\mathbf{S}[W]$ contains the frequency of string W in S. We call $\mathbf{S}$ the k-mer *composition vector* of string S. Given two strings S and T, assume that we want to compute a function $\kappa(S,T)$ that depends only on $N = \sum_{W \in \Sigma^k} f(\mathbf{S}[W], \mathbf{T}[W])$, $D_S = \sum_{W \in \Sigma^k} g(\mathbf{S}[W])$, and $D_T = \sum_{W \in \Sigma^k} h(\mathbf{T}[W])$, where f, g, and h are user-specified functions. $\kappa(S,T)$ if often called *k-mer kernel* in text classification. It is possible to compute $\kappa(S,T)$ in overall linear time by traversing the nodes of the generalized suffix tree of S and T in any order. A similar traversal of ST allows one to compute $\kappa(S,T)$ on composition vectors that are indexed by all possible substrings, of any length. In practice the frequencies used in composition vectors are normalized by their expected values under IID or Markov probability distributions: a number of kernels based on such normalized counts can still be computed in overall linear time by traversing the nodes of ST in any order [6].

Positional Annotations
Given two strings S and T, the longest string W that occurs in both S and T is clearly a right-maximal substring of the concatenation $U = S\#T\$$, where # and \$ are separators not belonging to Σ. Consider thus ST_U, and assume that every node v is annotated with $|\ell(v)|$ and with a bit $\texttt{flag}(v)$ set to one iff the subtree rooted at v contains at least one leaf that starts before position $|S|+1$ in U and at least one leaf starting after position $|S|+1$ in U. Such annotation can be carried out in a bottom-up traversal of ST. We can compute W by iterating over the nodes $v \in \mathsf{ST}$ with $\texttt{flag}(v) = 1$ and by cumulating the maximum of the lengths of the encountered labels. The set of *all common substrings* between S and T is the set of all prefixes of the labels of nodes $v \in \mathsf{ST}$ such that $\texttt{flag}(v) = 1$ and $\texttt{flag}(w) = 0$ for every child w of v. This approach generalizes immediately to more than two strings, and it allows one to compute the length of the longest substring common to *at least* τ strings in a collection of k strings in $O(k|U|)$ time and space. More advanced approaches solve this problem in $O(|U|)$ time [8].

Applications

The primitives discussed above find application in a wide set of domains. A list of the most salient ones includes exact and approximate string

searching, string compression, statistical pattern discovery, alignment-free string comparison, string kernels in learning theory, sequence analysis, and assembly in bioinformatics.

Cross-References

- Approximate Tandem Repeats
- Burrows-Wheeler Transform
- Compressed Suffix Array
- Compressed Suffix Trees
- Document Retrieval on String Collections
- Indexed Approximate String Matching
- Indexed Two-Dimensional String Matching
- Lempel-Ziv Compression
- Lowest Common Ancestors in Trees
- Pattern Matching on Compressed Text
- String Matching
- Suffix Array Construction
- Suffix Tree Construction

Recommended Reading

1. Abouelhoda MI, Kurtz S, Ohlebusch E (2004) Replacing suffix trees with enhanced suffix arrays. J Discret Algorithms 2(1):53–86
2. Apostolico A (1985) The myriad virtues of subword trees. In: Apostolico A, Galil Z (eds) Combinatorial algorithms on words. Springer, Berlin/New York, pp 85–96
3. Apostolico A, Bejerano G (2000) Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. J Comput Biol 7(3–4):381–393
4. Apostolico A, Preparata FP (1983) Optimal off-line detection of repetitions in a string. Theor Comput Sci 22(3):297–315
5. Apostolico A, Bock ME, Lonardi S, Xu X (2000) Efficient detection of unusual words. J Comput Biol 7(1–2):71–94
6. Apostolico A, Denas O et al (2008) Fast algorithms for computing sequence distances by exhaustive substring composition. Algorithms Mol Biol 3(13)
7. Beller T, Berger K, Ohlebusch E (2012) Space-efficient computation of maximal and supermaximal repeats in genome sequences. In: 19th international symposium on string processing and information retrieval (SPIRE 2012), Cartagena de Indias. Lecture notes in computer science, vol 7608. Springer, pp 99–110
8. Chi L, Hui K (1992) Color set size problem with applications to string matching. In: Combinatorial pattern matching, Tucson. Springer, pp 230–243
9. Crochemore M, Hancart C, Lecroq T (2007) Algorithms on strings. Cambridge University Press, New York
10. Farach M (1997) Optimal suffix tree construction with large alphabets. In: Proceedings of the 38th annual symposium on foundations of computer science, 1997, Miami Beach. IEEE, pp 137–143
11. Farach M, Noordewier M, Savari S, Shepp L, Wyner A, Ziv J (1995) On the entropy of DNA: algorithms and measurements based on memory and rapid convergence. In: Proceedings of the sixth annual ACM-SIAM symposium on discrete algorithms (SODA '95), San Francisco. Society for Industrial and Applied Mathematics, pp 48–57
12. Ferragina P (1997) Dynamic text indexing under string updates. J Algorithms 22(2):296–328
13. Fiala ER, Greene DH (1989) Data compression with finite windows. Commun ACM 32(4):490–505. doi:10.1145/63334.63341, http://doi.acm.org/10.1145/63334.63341
14. Gusfield D (1997) Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, Cambridge/New York
15. Gusfield D, Stoye J (2004) Linear time algorithms for finding and representing all the tandem repeats in a string. J Comput Syst Sci 69(4):525–546. doi:10.1016/j.jcss.2004.03.004, http://dx.doi.org/10.1016/j.jcss.2004.03.004
16. Herold J, Kurtz S, Giegerich R (2008) Efficient computation of absent words in genomic sequences. BMC Bioinform 9(1):167
17. Kärkkäinen J, Sanders P, Burkhardt S (2006) Linear work suffix array construction. J ACM 53(6):918–936
18. Kasai T, Lee G, Arimura H, Arikawa S, Park K (2001) Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Combinatorial pattern matching, Jerusalem. Springer, pp 181–192
19. Kim DK, Sim JS, Park H, Park K (2005) Constructing suffix arrays in linear time. J Discret Algorithms 3(2):126–142
20. Ko P, Aluru S (2003) Space efficient linear time construction of suffix arrays. In: Combinatorial pattern matching, Morelia. Springer, pp 200–210
21. Kurtz S (1999) Reducing the space requirement of suffix trees. Softw Pract Exp 29:1149–1171
22. Larsson NJ (1996) Extended application of suffix trees to data compression. In: Data compression conference, Snowbird, pp 190–199
23. Lempel A, Ziv J (1976) On the complexity of finite sequences. IEEE Trans Inf Theory 22:75–81
24. Manber U, Myers G (1993) Suffix arrays: a new method for on-line string searches. SIAM J Comput 22(5):935–948
25. McCreight EM (1976) A space-economical suffix tree construction algorithm. J ACM 23(2):262–272

26. Muthukrishnan S (2002) Efficient algorithms for document retrieval problems. In: Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms (SODA '02), San Francisco. Society for Industrial and Applied Mathematics, Philadelphia, pp 657–666. http://dl.acm.org/citation.cfm?id=545381.545469
27. Ohlebusch E, Gog S, Kügel A (2010) Computing matching statistics and maximal exact matches on compressed full-text indexes. In: XXth international symposium on string processing and information retrieval (SPIRE 2010), Los Cabos, pp 347–358
28. Puglisi SJ, Smyth WF, Turpin AH (2007) A taxonomy of suffix array construction algorithms. ACM Comput Surv 39(2):4
29. Rodeh M, Pratt VR, Even S (1981) Linear algorithm for data compression via string matching. J ACM 28(1):16–24
30. Smola AJ, Vishwanathan S (2003) Fast kernels for string and tree matching. In: Becker S, Thrun S, Obermayer K (eds) Advances in neural information processing systems (NIPS '03) 15, Vancouver. MIT, pp 585–592
31. Stoye J, Gusfield D (2002) Simple and flexible detection of contiguous repeats using a suffix tree. Theor Comput Sci 270(1):843–856
32. Ukkonen E (1995) On-line construction of suffix trees. Algorithmica 14(3):249–260
33. Weiner P (1973) Linear pattern matching algorithms. In: IEEE conference record of 14th annual symposium on switching and automata theory (SWAT '08), Iowa City, 1973. IEEE, pp 1–11

Sugiyama Algorithm

Nikola S. Nikolov
Department of Computer Science and Information Systems, University of Limerick, Limerick, Republic of Ireland

Keywords

Barycentric method; Crossing minimization; Hierarchical graph drawing; Layered graph drawing; Sugiyama algorithm; Sugiyama framework

Years and Authors of Summarized Original Work

1981; Sugiyama, Tagawa, Toda

Problem Definition

Given a directed graph (digraph) $G(V, E)$ with a set of vertices V and a set of edges E, the Sugiyama algorithm solves the problem of finding a 2D hierarchical drawing of G subject to the following readability requirements:

(a) Vertices are drawn on horizontal lines without overlapping; each line represents a level in the hierarchy; all edges point downwards.
(b) Short-span edges (i.e., edges between adjacent levels) are drawn with straight lines.
(c) Long-span edges (i.e., edges between nonadjacent levels) are drawn as close to straight lines as possible.
(d) The number of edge crossings is the minimum.
(e) Vertices connected to each other are placed as close to each other as possible.
(f) The layout of edges coming into (or going out of) a vertex is balanced, i.e., edges are evenly spaced around a common target (or source) vertex.

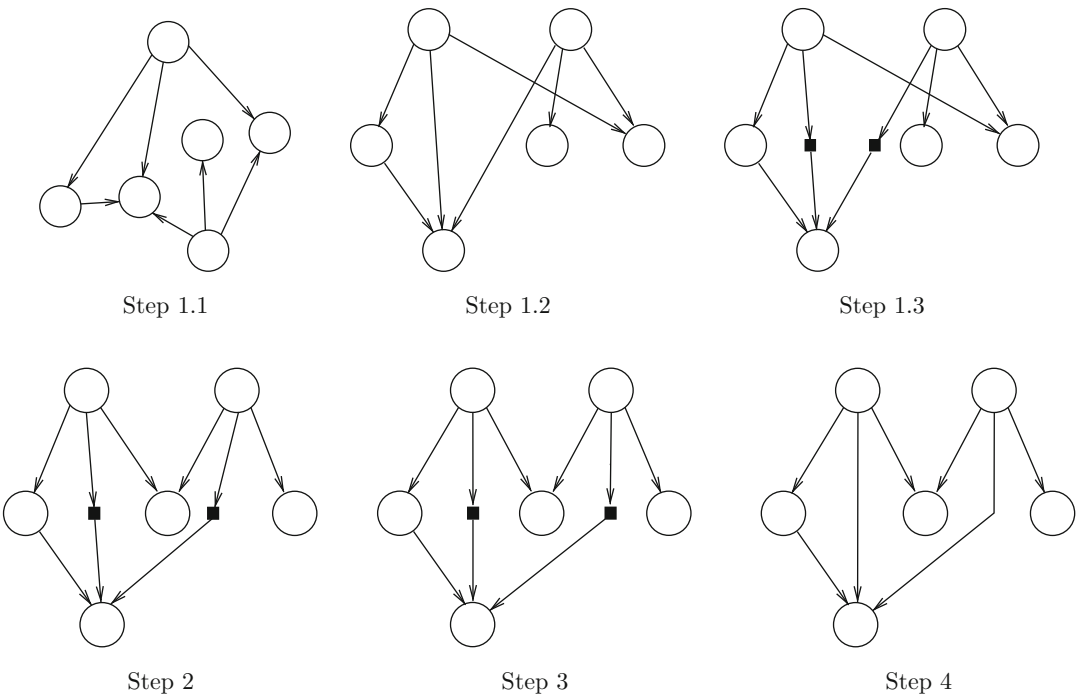
Requirements (a) and (b) are easy to meet and they are imposed as mandatory basic drawing rules. Requirements (c)–(f) are much harder to satisfy and typically they are met approximately [1, 4, 11].

Key Results

Sugiyama et al. propose a four-step procedure for finding a hierarchical drawing of a digraph subject to the readability requirements listed above. It is known as the Sugiyama algorithm, the Sugiyama method, or the Sugiyama framework [19]. The steps of the Sugiyama framework are illustrated in Fig. 1.

The Sugiyama Framework

Step 1: Preparatory step for transforming the input digraph G into a proper hierarchy.



Sugiyama Algorithm, Fig. 1 Illustration of the steps of the Sugiyama framework

Step 1.1: Transform the input digraph G into a directed acyclic graph (dag) by reversing the direction of some edges.

Step 1.2: Transform the dag into a multilevel digraph, called a *hierarchy*, by partitioning V into l levels (or layers) $V_1, V_2, \ldots, V_l$ such that for each edge $e = (v, w) \in E$ if $v \in V_i$ then $w \in V_{i+1}$. Levels are drawn on horizontal lines which determine the $y-$coordinates of the vertices.

Step 1.3 Transform the hierarchy into a *proper hierarchy* by introducing *dummy vertices* along long-span edges; one dummy vertex at each crossing of a long-span edge with a level.

Step 2: For each level V_i, specify a linear order σ_i of the vertices in V_i with the goal of minimizing the total number of edge crossing.

Step 3: Determine the $x-$coordinates of the vertices subject to requirements (c), (e), and (f) while preserving the linear order in the levels.

Step 4: Draw G in a 2D drawing area where dummy vertices are removed and the long-span edges are restored.

Steps 1.3 and 4 are trivial as computational problems. Steps 1.1 and 1.2 can be solved easily if the only readability requirements are those listed above. However, some sensible additional requirements can turn Steps 1.1 and 1.2 into difficult combinatorial optimization problems. For example, if we want to minimize the number of reversed edges at Step 1.1, then we need to solve the MINIMUM FEEDBACK ARC SET problem which is NP-hard [12]. Similarly, if we impose upper bounds on both the number of levels and the number of vertices per level, then the problem in Step 1.2, known as the *layering* problem, becomes NP-complete [4].

Following the work of Sugiyama et al., two types of solutions to the layering problem have been proposed in the research literature. The first type of layer assignment algorithm is list-scheduling algorithms (adapted from the area of static precedence-constrained multiprocessor

scheduling) which produce layer assignments with either the minimum number of levels or a specified maximum number of vertices per level [4]. These include the longest-path algorithm [13] and the Coffman-Graham algorithm [3] as well as the proposed by Nikolov et al. [15] MinWidth and StretchWidth heuristics which take into account the dummy vertices. The second type of algorithm employs network simplex and branch-and-cut techniques, respectively, for minimizing the number of dummy vertices with or without constraints on the number of levels and the number of vertices per level [9, 10].

Steps 2 and 3 are already hard to solve with the readability requirements listed above. It has also been suggested to precede Step 2 by an edge concentration or edge bundling step for achieving a more readable drawing [14, 16]. The other key results in the work of Sugiyama et al., besides defining the four-step framework, are efficient heuristics for Steps 2 and 3, respectively.

Reduction of the Number of Edge Crossings

Consider a proper hierarchy $G(V, E, \mathcal{L})$ with a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$, a set of edges $E = \{e_1, e_2, \ldots, e_m\}$, and a partitioning $\mathcal{L} = \{V_1, V_2, \ldots, V_l\}$ of the vertex set V into l levels (the result of Step 1.3). Let $\sigma_i : V_i \to \{1, 2, \ldots |V_i|\}$ be a linear order of the vertices in level V_i and let S_i be the set of all possible orders σ_i. The problem at Step 2 of the Sugiyama algorithm is to find a set of linear orders $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_l\} \in S_1 \times S_2 \times \ldots \times S_l$ such that the total number of edge crossings is the minimum. Let $K(G, \sigma)$ be the total number of edge crossings for a hierarchy G and a set of linear orders σ, and let $K(V_i, V_{i+1}, \sigma_i, \sigma_{i+1})$ be the number of edge crossings between layers V_i and V_{i+1} with linear orders σ_i and σ_{i+1}, respectively.

The algorithm, proposed by Sugiyama et al. for Step 2, is a heuristic which consists in initially choosing a random order σ_1 for the vertices in level V_1 and then repeatedly executing the following five-step procedure, called Down-Up, until either σ does not change or an initially given maximum number of iterations is reached.

The Down-Up Procedure

Step A: $i \leftarrow 1$.

Step B: With a fixed linear order σ_i, find a linear order σ_{i+1} which minimizes $K(V_i, V_{i+1}, \sigma_i, \sigma_{i+1})$.

Step C: If $i < n - 1$, then $i \leftarrow i + 1$ and go to Step B. Otherwise, go to Step D.

Step D: With a fixed linear order σ_{i+1}, find a linear order σ_i which minimizes $K(V_i, V_{i+1}, \sigma_i, \sigma_{i+1})$.

Step E: If $i > 1$, then $i \leftarrow i - 1$ and go to Step D. Otherwise, stop.

Both Step B and Step D involve minimizing the number of edge crossings between two adjacent layers with the linear order in one of them being fixed. This problem is known as the ONE-SIDED CROSSING MINIMIZATION (OSCM) problem, which has been shown to be NP-hard [5]. Based on previous work by Warfield [20], Sugiyama et al. show how OSCM can be reduced to the MINIMUM FEEDBACK SET problem and propose a heuristic method, called the *barycentric method*, for solving it. Let $A = (a_{ij})$ be the adjacency matrix of G. In essence, with a fixed linear order σ_i, the barycentric method orders the vertices in level V_{i+1} in the increasing order of their barycenters B_j, defined with Eq. (1).

$$B_j = \sum_{k=1}^{|V_i|} a_{kj}\sigma_i(v_k) \Big/ \sum_{k=1}^{|V_i|} a_{kj}, \qquad j \in \{1, 2, \ldots, |V_{i+1}|\} \tag{1}$$

Sugiyama et al. evaluate the Down-Up procedure experimentally with 800 randomly generated hierarchies as well as with five hierarchies from practical applications. Their conclusion is

that the proposed heuristic is effective. It was observed that in most cases the Down-Up procedure requires a single iteration. Reportedly, the heuristic was successfully extended for the case when vertices in each level are partitioned into subsets where the vertices in each subset must be arranged adjacently.

Step 2 is probably the best studied part of the Sugiyama framework. Numerous improvements to the original technique as well as alternative algorithms for crossing minimization have been proposed since the introduction of the Sugiyama framework [1, 4, 5, 7, 9, 11]. Notable among them is the 3-approximation *median method* proposed by Eades and Wormald [5] for solving the OSCM problem. Having the order of the vertices in level V_i fixed, the median method consists of placing each vertex in level V_{i+1} at a position which corresponds to the median of the positions of its neighbors in level V_i. Since the median method is an approximation algorithm, it guarantees to find a solution without edge crossings if such exists.

Determination of x-Coordinates of Vertices

For Step 3 of their framework, Sugiyama et al. propose a version of the Down-Up procedure with the barycenter of a vertex based on the x-coordinates of the connected to it vertices in an adjacent level. Consider the *down* part of the Down-Up procedure (the *up* part is symmetrical). If the x-coordinates of the vertices in level V_i are known, the barycenters B_j^* of the vertices in level V_{i+1} are defined with Eq. (2).

$$B_j^* = \sum_{k=1}^{|V_i|} a_{kj} x(v_k) / \sum_{k=1}^{|V_i|} a_{kj}, \quad j \in \{1, 2, \ldots, |V_{i+1}|\} \qquad (2)$$

The x-coordinates of the vertices in level V_{i+1} are determined according to their priority. The highest priority has the dummy vertices (introduced in Step 1.3), and the priority of each other vertex in level V_{i+1} is the number of vertices in level V_i connected to it. The x-coordinate of each vertex $v_j \in V_{i+1}$ is the integer number which is the closest to B_j^* available horizontal position (without changing the linear order from Step 2 and without displacing already placed vertices with higher priority). In finding this position, it is allowed to displace vertices with a priority lower than the priority of v_j, where this displacement should be as little as possible.

Sugiyama et al. evaluate the effectiveness of this method for improving the readability requirements (c), (e), and (f) experimentally. Reportedly, they have extended their heuristic for the case when the dimensions of the vertices are not insignificant. Both the Step 2 and the Step 3 heuristics were successfully applied to a hierarchy with more than 500 vertices.

Alternative algorithms for Step 3 have been proposed by Gansner et al. [9], Eades et al. [6], and Sander [17]. Probably, the best solution for Step 3 to date is the $O(|V|)$ algorithm of Brandes and Köpf [2]. It assigns x-coordinates to vertices by computing four *extreme* vertex alignments which are then combined into a final layout with at most two bends per edge.

Applications

Hierarchical graph drawings are useful for providing insight into hierarchical structures in complex systems. In recent years, the Sugiyama algorithm has found an important application for visual analysis of large social and biological networks [8, 18].

Cross-References

▸ List Scheduling

Recommended Reading

1. Bastert O, Matuszewski C (2000) Layered drawings of digraphs. In: Kaufman M, Wagner D (eds) Drawing graphs: method and models. Lecture notes in computer science, vol 2025. Springer, Berlin/Heidelberg, pp 87–120

2. Brandes U, Köpf B (2002) Fast and simple horizontal coordinate assignment. In: Mutzel P, Jünger M, Leipert S (eds) Graph drawing. Lecture notes in computer science, vol 2265. Springer, Berlin/Heidelberg, pp 31–44
3. Coffman EG Jr, Graham R (1972) Optimal scheduling for two-processor systems. Acta Inform 1(3):200–213
4. Eades P, Sugiyama K (1990) How to draw a directed graph. J Inf Process 13(4):424–437
5. Eades P, Wormald NC (1994) Edge crossings in drawings of bipartite graphs. Algorithmica 11(4):379–403
6. Eades P, Lin X, Tamassia R (1996) An algorithm for drawing a hierarchical graph. Int J Comput Geom Appl 6(2):145–156
7. Eppstein D, Goodrich MT, Meng JY (2007) Confluent layered drawings. Algorithmica 47(4):439–452
8. Fu X, Hong SH, Nikolov N, Shen X, Wu Y, Xu K (2007) Visualization and analysis of email networks. In: Asia-Pacific symposium on visualization, Sydney, pp 1–8
9. Gansner ER, Koutsofios E, North SC, Vo KP (1993) A technique for drawing directed graphs. IEEE Trans Softw Eng 19(3):214–230
10. Healy P, Nikolov NS (2002) How to layer a directed acyclic graph. In: Mutzel P, Jünger M, Leipert S (eds) Graph drawing. Lecture notes in computer science, vol 2265. Springer, Berlin/Heidelberg, pp 16–30
11. Healy P, Nikolov NS (2013) Hierarchical drawing algorithms. In: Tamassia R (ed) Handbook of graph drawing and visualization. Discrete mathematics and its applications, chap 13. Chapman and Hall/CRC, Boca Raton/London/New York, pp 409–454
12. Lempel A, Cederbaum I (1966) Minimum feedback arc and vertex sets of a directed graph. IEEE Trans Circuit Theory 13(4):399–403
13. Mehlhorn K (1984) Data structures and algorithms, Volume 2: graph algorithms and NP-completeness. Springer, Heidelberg
14. Newbery FJ (1989) Edge concentration: a method for clustering directed graphs. In: Proceedings of the 2nd international workshop on software configuration management (SCM '89), Princeton. ACM, pp 76–85
15. Nikolov NS, Tarassov A, Branke J (2005) In search for efficient heuristics for minimum-width graph layering with consideration of dummy nodes. J Exp Algorithmics 10:1–27
16. Pupyrev S, Nachmanson L, Kaufmann M (2011) Improving layered graph layouts with edge bundling. In: Brandes U, Cornelsen S (eds) Graph drawing. Lecture notes in computer science, vol 6502. Springer, Berlin/Heidelberg, pp 329–340
17. Sander G (1996) A fast heuristic for hierarchical Manhattan layout. In: Brandenburg FJ (ed) Graph drawing. Lecture notes in computer science, vol 1027. Springer, Berlin/Heidelberg, pp 447–458
18. Schwikowski B, Uetz P, Fields S (2000) A network of protein-protein interactions in yeast. Nat Biotechnol 18(12):1257–1261
19. Sugiyama K, Tagawa S, Toda M (1981) Methods for visual understanding of hierarchical system structures. IEEE Trans Syst Man Cybern 11(2):109–125
20. Warfield JN (1977) Crossing theory and hierarchical mapping. IEEE Trans Syst Man Cybern 7(7):502–523

Superiority and Complexity of the Spaced Seeds

Louxin Zhang
Department of Mathematics, National University of Singapore, Singapore, Singapore

Keywords

Homology search; NP-hardness; Sensitivity and hit probability; Spaced seeds

Years and Authors of Summarized Original Work

2006; Ma, Li, Zhang

Problem Definition

In the 1970s, sequence alignment was introduced to demonstrate the similarity of the sequences of genes and proteins [12]. A DNA sequence is a finite sequence over four nucleotides – adenine, guanine, cytosine, and thymine, whereas a protein sequence is over 20 amino acids. Homologous proteins have similar biological functions. Since they evolve from a common ancestral sequence, the sequences of homologous proteins and their encoding genes are often highly similar. Therefore, the DNA or amino acid sequence of a protein is often aligned with the sequences of well-studied proteins to infer the biological functions of the protein.

Formally, an alignment of two sequences, S and T, on an alphabet $\mathcal{B}$ is a two-row matrix with the following properties:

1. The letters in S are listed in order, interspersed with space symbols "–," in a row, where "–" represents the fact that a letter is missing at a position.
2. The letters in T are listed in the other row in the same manner.
3. Each column does not contain two "–."

An alignment of S and T poses a model of the evolution from their least common ancestral sequence to themselves. An alignment is scored using a scoring matrix that has a score for every pair of letters in $\mathcal{B} \cup \{-\}$. The score of an alignment is defined to be the sum of the scores of the pairs of letters appearing in the columns of the alignment.

Proteins often have multiple functions. Two proteins having a common function often have one or several highly similar regions in their DNA and amino acid sequences. Such "conserved" regions are found by solving the local alignment problem:

Input: Two sequences $S = s_1 s_2 \dots s_m$ and $T = t_1 t_2 \cdots t_n$ on an alphabet.
Find: Two subsequences $S' = s_i s_{i+1} \cdots s_j$ $(i \leq j)$ and $T' = t_k t_{k+1} \cdots t_l$ $(k \leq l)$ such that the alignment score of S' and T' is as large as possible.

The alignments between their subsequences are called *local alignments* of S and T.

A dynamic programming approach takes quadratic time to solve the local alignment problem [13]. Unfortunately, it is not fast enough for homology search against a database with millions of DNA or protein sequences. Therefore, a filtration technique was adopted to design fast algorithms for homology search in the 1990s [1], by which good local alignments between two sequences are found by first identifying short consecutive matches of a specified length between the sequences, called *seed hits*, and then extending them to obtain good local alignments.

The filtration technique has a dilemma over sensitivity and speed. Employing a long seed will miss some good local alignments between two sequences, decreasing sensitivity; on the other hand, using a short seed will waste time on extending many seed hits into local alignments that are not biologically meaningful, resulting in low speed.

In PatternHunter [10], Ma, Tromp, and Li introduced the idea of optimized spaced seeds to achieve good balance between the sensitivity and speed of the filtration approach. PatternHunter by default looks for nucleotide match in 11 positions in every region of 18 bases long, specified by the string $111 * 1 * *1 * 1 * *11 * 111$, to trigger the process of local alignment. Such hit patterns, called *spaced seeds*, led to surprisingly higher sensitivity as well as speed than the consecutive seed 11111111111 that has the same number of match positions [10]. Moreover, sensitivity can further be improved by employing multiple spaced seeds that are longer than 18 bases [8,14]. This motivates the study of how to find the optimal spaced seeds of given length and weight [2–5,7].

Key Results

A spaced seed Q can be represented by a string of 1's and $*$'s, where 1's give the match positions in a seed hit. The number of 1's in Q is called its *weight*, denoted by w_Q; the length of the corresponding string is called its *length*, denoted by L_Q. The relative positions in Q are denoted by $\mathcal{RP}(Q)$. For example, for $Q = 111 * 1 * *1 * 1 * *11 * 111$, $\mathcal{RP}(Q) = \{0, 1, 2, 4, 7, 9, 12, 13, 15, 16, 17\}$.

An alignment containing no –'s is called a *ungapped alignment*. A local ungapped alignment can be modeled as a 0-1 sequence by translating match columns (containing two identical letters) into 1's and mismatch columns into 0's. Hence, a hit of Q identifies an alignment if the relative positions of Q match 1's in a region in the corresponding 0-1 string of the alignment.

Assume match occurs independently with probability p at a position in a local ungapped alignment. The *sensitivity* of Q in detecting a local alignment of n columns of two sequences with identity p is then defined to be the probability that Q hits a Bernoulli random sequence, called a *uniform region*, in which 1

and 0 appear with probability p and $(1-p)$, respectively. A spaced seed is *optimal* for aligning sequences with identity p of length n if it has the largest hit probability over a uniform region of length n in which 1 appears with probability p at a position.

A straightforward method for identifying optimal spaced seeds is to exhaustively examine all the spaced seeds of given length and weight by keeping the largest sensitivity (or hit probability) over a uniform region. Unfortunately, the sensitivity of a spaced seed is unlikely computable in polynomial time.

Theorem 1 *Computing the sensitivity of a spaced seed over a uniform region is NP-hard.*

The hit probability of a spaced seed over a uniform region can be computed using a dynamic programming approach [7] or using recurrence relations [4,5]. Not surprisingly, these approaches become impractical for identifying long spaced seeds, because their complexities are an exponential function in the difference of the length and weight of spaced seeds under consideration. Here a simple polynomial-time approximation scheme is presented.

WISESAMPLE ALGORITHM

Input: A spaced seed Q, a positive integer n, $0 < p < 1$, and $\epsilon > 0$.
Find: An estimate of hit probability Q in a uniform region of length n in which bit 1 appears at a position with probability p.

Initialize an array A: $A[i] \leftarrow 0$ for $j = 1, 2, \ldots, n - L_Q$;
$N \leftarrow \lceil 6\epsilon^{-2} n^2 \log n \rceil$;
Repeats N times
  $R[i] \leftarrow 1$ for $i \in \mathcal{RP}(Q)$;
  $R[i] \leftarrow 1$ with probability p for $i \in \{1, 2, \ldots, n\} - \mathcal{RP}(Q)$;
  For $i = 1, 2, \ldots, L - L_Q$
    If Q does not hit the subregion $R[1, i + L_Q - 1]$
      $A[i] \leftarrow A[i] + 1$;
Output $p^{w_Q} \left(1 + N^{-1} \sum_{j=1}^{n-L_Q} n_j\right)$.

Theorem 2 *Let Q be a spaced seed and its hit probability be x on a uniform region with identity p of length n.* WISESAMPLE *outputs an estimate y of x on input Q, n, p, and $\epsilon >$ such that $|y - x| \leq \epsilon x$ with high probability.*

Let Q be a spaced seed and R a uniform region with identity p of length n. Following convention in renewal theory, Q hits R at position k if and only if $R[k - L_Q + i_j + 1] = 1$ for all $1 \leq j \leq w_Q$. Let A_k be the event that Q hits R at position k and $\bar{A}_k$ be the complement event of A_k. Then the probability f_k that Q **first** hits R at the k-th position is:

$$f_k = \Pr[\bar{A}_0 \bar{A}_1 \cdots \bar{A}_{k-2} A_{k-1}].$$

The hit probability $Q_n(p)$ of Q on R is equal to:

$$Q_n(p) = \Pr[A_0 \cup A_1 \cup \cdots \cup A_{n-1}].$$

When seed hits are extended into local alignments, two seed hits will give one local alignment if they overlap. Therefore, the sensitivity of a spaced seed is closely related to the number of its nonoverlapping hits in a uniform region. A nonoverlapping hit of a spaced seed is a recurrent event with the following convention: If a hit at position k is selected as a nonoverlapping hit, then the next nonoverlapping hit is the first hit at or after position $k + L_Q$.

The average distance, μ_Q, between two successive nonoverlapping hits of Q is defined to be

$$\mu_Q = \sum_{j \geq L_Q} j f_j.$$

A spaced seed is *nonuniform* if $g.c.d.(\mathcal{RP}(Q)) = 1$.

Theorem 3 *For any nonuniform spaced seed* Q,

$$\mu_Q \le \sum_{j=1}^{w_Q} p^{-j} + (L_Q - w_q) - (1-p)\left(p^{2-w_Q} - 1\right)/p.$$

Buhler et al. [3] proved that for any spaced seed Q, there are two constants α_Q and λ_Q that are independent of n such that $\lim_{n\to\infty}(1 - Q_n(p))/\left(\alpha_Q \lambda_Q\right) = 1$, where λ_Q is the largest eigenvalue of the transition matrix of a Markov chain model constructed from Q.

Theorem 4 *For the consecutive seed* B *of weight* w,

$$\frac{1}{\sum_{j=1}^{w} p^{-j} - w + 1} \le \lambda_B \le 1 - \frac{1}{\sum_{j=1}^{w}(p^{-j} + p^{j-1}) - w}.$$

For a spaced seed Q,

$$1 - \frac{1}{\mu_Q - L_Q + 1} \le \lambda_Q \le 1 - \frac{1}{\mu_Q}.$$

If $L_Q < (1-p)\left[p^{2-w_Q} - 1\right]/p + 1$, by Theorems 3 and 4, $\lambda_Q \le \lambda_B$. This implies that Q has a larger hit probability than the consecutive seed of the same weight in a long uniform region with identity p.

The detailed proofs of these results can be found in [11, 15].

Applications

Spaced seed approach finds applications in homology search and comparison of genome sequences. PatternHunter was used to compare the mouse and human genomes in the mouse genome project [6]. MegaBLAST and BLASTZ have adopted spaced seeds for homology search. Recently, the approach has also been used in mapping short reads into reference genome sequences.

Interestingly, spaced seed design is found to be closely related to optimal Golomb ruler design [9].

Open Problems

It is proved to be NP-hard to identify the optimal spaced seeds over a nonuniform region [8].

Open problem 1 Is it NP-hard to find the optimal spaced seed of a given length and weight over a uniform region?

It has been shown that a uniform spaced seed has a lower hit probability than the consecutive seed of the same weight over any uniform region [4, 7]. But the following problem is open:

Open problem 2 For any nonuniform spaced seed Q and $0 < p < 1$, is there $n(p, Q)$ such that Q has a larger hit probability than the consecutive seed of the same weight over a uniform region with identity p of length $n \ge n(p, Q)$?

Cross-References

- Local Alignment (with Affine Gap Weights)
- Local Alignment (with Concave Gap Weights)

Recommended Reading

1. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990) Basic local alignment search tool. J Mol Biol 215(3):403–410
2. Brejovà B, Brown D, Vinař T (2004) Optimal spaced seeds for homologous coding regions. J Bioinformatics Comput Biol 1:595–610
3. Buhler J, Keich U, Sun Y (2004) Designing seeds for similarity search in genomic DNA. J Comput Syst Sci 70:342–363
4. Choi KP, Zhang LX (2004) Sensitivity analysis and efficient method for identifying optimal spaced seeds. J Comput Syst Sci 68:22–40

5. Choi KP, Zeng F, Zhang LX (2004) Good spaced seeds for homology search. Bioinformatics 20:1053–1059
6. Intl Mouse Genome Sequencing Consortium (2002) Initial sequencing and comparative analysis of the mouse genome. Nature 409:520–562
7. Keich U, Li M, Ma B, Tromp J (2004) On spaced seeds for similarity search. Discret Appl Math 3:253–263
8. Li M, Ma B, Kisman D, Tromp J (2004) PatternHunter II: highly sensitive and fast homology search. J Bioinformatics Comput Biol 2:417–440
9. Ma B, Yao H (2009) Seed optimization for iid similarities is no easier than optimal Golomb ruler design. Inf Process Lett 109(19):1120–1124
10. Ma B, Tromp J, Li M (2002) PatternHunter: faster and more sensitive homology search. Bioinformatics 18:440–445
11. Ma B, Li M (2007) On the complexity of the spaced seeds. J Comput Syst Sci 73:1024–1034
12. Needleman SB, Wunsch CD (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol 48:443–453
13. Smith TF, Waterman MS (1980) Identification of common molecular subsequences. J Mol Biol 147:195–197
14. Sun Y, Buhler J (2004) Designing multiple simultaneous seeds for DNA similarity search. In: Proceedings RECOMB'04, 2004, San Diego, pp 76–85
15. Zhang LX (2007) Superiority of spaced seeds for homology search. IEEE/ACM Trans Comput Biol Bioinformatics (TCBB) 4:496–505

Support Vector Machines

Nello Cristianini[1] and Elisa Ricci[2]
[1]Department of Engineering Mathematics, and Computer Science, University of Bristol, Bristol, UK
[2]Department of Electronic and Information Engineering, University of Perugia, Perugia, Italy

Keywords

Kernel Methods; Large Margin Methods; Support Vector Machines

Years and Authors of Summarized Original Work

1992; Boser, Guyon, Vapnik

Problem Definition

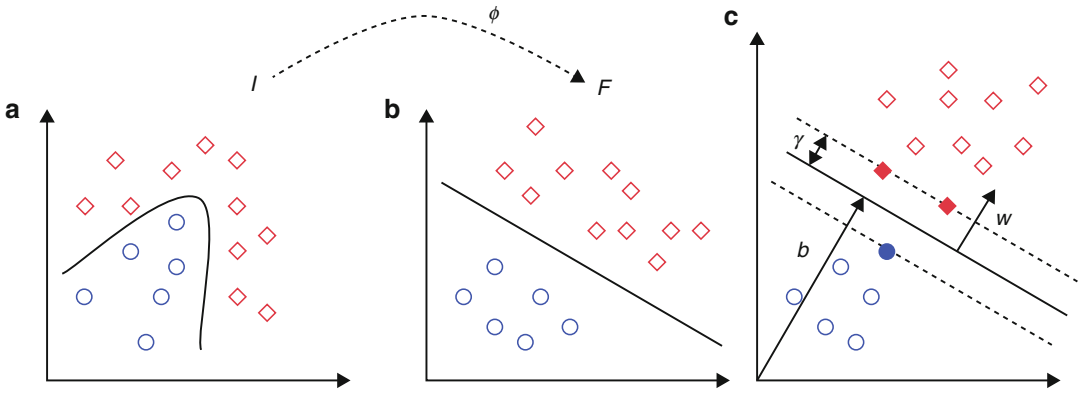
In 1992 Vapnik and coworkers [1] proposed a supervised algorithm for classification that has since evolved into what are now known as support vector machines (SVMs) [2]: a class of algorithms for classification, regression, and other applications that represent the current state of the art in the field. Among the key innovations of this method were the explicit use of convex optimization, statistical learning theory, and kernel functions.

Classification

Given a *training set* $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$ of data points $\mathbf{x}_i$ from $X \subseteq \mathbb{R}^n$ with corresponding labels y_i from $Y = \{-1, +1\}$, generated from an unknown distribution, the task of classification is to learn a function $g: X \to Y$ that correctly classifies new examples $(\mathbf{x}, y)$ (i.e., such that $g(\mathbf{x}) = y$) generated from the same underlying distribution as the training data.

A good classifier should guarantee the best possible generalization performance (e.g., the smallest error on unseen examples). Statistical learning theory [3], from which SVMs originated, provides a link between the expected generalization error for a given training set and a property of the classifier known as its capacity. The SV algorithm effectively regulates the capacity by considering the function corresponding to the hyperplane that separates, according to the labels, the given training data and it is maximally distant from them (*maximal margin hyperplane*). When no linear separation is possible, a nonlinear mapping into a higher dimensional *feature space* is realized. The hyperplane found in the feature space corresponds to a nonlinear decision boundary in the input space.

Let $\phi : I \subseteq \mathbb{R}^n \to F \subseteq \mathbb{R}^N$ be a mapping from the input space I to the feature space F (Fig. 1a). In the learning phase, the algorithm finds a hyperplane defined by the equation $\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle = b$ such that the *margin*



Support Vector Machines, Fig. 1 (**a**) The feature map simplifies the classification task. (**b**) A maximal margin hyperplane with its support vectors highlighted

$$\gamma = \min_{1 \le i \le \ell} \; y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - b)$$
$$= \min_{1 \le i \le \ell} \; y_i g(\mathbf{x}_i) \qquad (1)$$

is maximized, where $\langle , \rangle$ denotes the inner product, $\mathbf{w}$ is a ℓ-dimensional vector of weights, and b is a threshold.

The quantity $(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - b)/||\mathbf{w}||$ is the signed distance of the sample $\mathbf{x}_i$ from the hyperplane. When multiplied by the label y_i, it gives a positive value for correct classification and a negative value for an uncorrect one. Given a new data point $\mathbf{x}$, a label is assigned evaluating the decision function:

$$g(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \phi(\mathbf{x})\rangle - b) \qquad (2)$$

Maximizing the Margin

For linearly separable classes, there exists a hyperplane $(\mathbf{w}, b)$ such that

$$y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - b) \ge \gamma, \quad i = 1, \ldots, \ell. \qquad (3)$$

Imposing $||\mathbf{w}||^2 = 1$, the choice of the hyperplane such that the margin is maximized is equivalent to the following optimization problem:

$$\max_{\mathbf{w},b,\gamma} \gamma$$
$$\text{subject to } y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - b) \ge \gamma, \; i = 1, \ldots, \ell, \qquad (4)$$
$$\text{and } ||\mathbf{w}||^2 = 1.$$

An efficient solution can be found in the dual space by introducing the Lagrange multipliers α_i, $i = 1, \ldots, \ell$. The problem (4) can be recast in the following dual form:

$$\max_{\alpha} \sum_{i=1}^{\ell} \alpha_i - \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle \qquad (5)$$
$$\text{subject to } \sum_{i=1}^{\ell} \alpha_i y_i = 0, \quad \alpha_i \ge 0.$$

This formulation shows how the problem reduces to a *convex* (quadratic) optimization task. A key property of solutions α^* of this kind of problems is that they must satisfy the Karush-Kuhn-Tucker (KKT) conditions that ensure that only a subset of training examples needs to be associated to a nonzero α_i. This property is called *sparseness* of the SVM solution and is crucial in practical applications.

In the solution α^*, often only a subset of training examples is associated to nonzero α_i. These are called *support vectors* and correspond to the points that lie closest to the separating hyperplane (Fig. 1b). For the maximal margin hyperplane, the weight vector $\mathbf{w}^*$ is given by a linear function of the training points:

$$\mathbf{w}^* \sum_{i=1}^{\ell} \alpha_i^* y_i \phi(\mathbf{x}_i). \qquad (6)$$

S

Then the decision function (2) can equivalently be expressed as

$$g(\mathbf{x}) = \text{sign}(\sum_{i=1}^{\ell} \alpha_i^* y_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle - b). \quad (7)$$

For a support vector $\mathbf{x}_i$, it is $\langle \mathbf{w}^*, \phi(\mathbf{x}_i) \rangle - b = y_i$ from which the optimum bias b^* can be computed. However, it is better to average the values obtained by considering all the support vectors [2]. Both the quadratic programming (QP) problem (5) and the decision function (7) depend only on the dot product between data points. The matrix of dot products with elements $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ is called the *kernel matrix*. In the case of linear separation, we simply have $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$, but in general, one can use functions that provide nonlinear decision boundaries. Widely used kernels are the polynomial $K(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + 1)^d$ or the Gaussian $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{\sigma^2}}$ where d and σ are user-defined parameters.

Key Results

In the framework of learning from examples, SVMs have shown several advantages compared to traditional neural network models (which represented the state of the art in many classification tasks up to 1992). The statistical motivation for seeking the maximal margin solution is to minimize an upper bound on the test error that is independent of the number of dimensions and inversely proportional to the separation margin (and the sample size). This directly suggests embedding of the data in a high-dimensional space where a large separation margin can be achieved; this can be done efficiently with kernels using techniques from convex optimization. The sparseness of the solution, implied by the KKT conditions, adds to the efficiency of the result.

The initial formulation of SVMs by Vapnik and coworkers [1] has been extended by many other researchers. Here we summarize some key contributions.

Soft Margin

In the presence of noise the SV algorithm can be subject to overfitting. In this case one needs to tolerate some training errors in order to obtain a better generalization power. This has led to the development of the *soft margin* classifiers [4]. Introducing the slack variables $\xi_i \geq 0$, optimal class separation can be obtained by

$$\begin{aligned} &\min_{\mathbf{w},b\gamma,\xi} -\gamma + C \sum_{i=1}^{\ell} \xi_i \\ &\text{subject to } y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle - b) \geq \gamma - \xi_i, \xi_i \geq 0 \\ &i = 1, \ldots, \ell \text{ and } ||\mathbf{w}||^2 = 1. \end{aligned} \quad (8)$$

The constant C is user defined and controls the trade-off between the maximization of the margin and the number of classification errors. The dual formulation is the same as (5) with the only difference in the bound constraints ($0 \leq \alpha_i \leq C, \quad i = 1, \ldots, \ell$). The choice of soft margin parameter is one of the two main design choices (together with the kernel function) in applications. It is an elegant result [5] that the entire set of solutions for all possible values of C can be found with essentially the same computational cost as finding a single solution: this set is often called the *regularization path*.

Regression

A SV algorithm for regression, called support vector regression (SVR), was proposed in 1996 [6]. A linear algorithm is used in the kernel-induced feature space to construct a function such that the training points are inside a tube of given radius ε. As for classification the regression function only depends on a subset of the training data.

Speeding Up the Quadratic Program

Since the emergence of SVMs, many researchers have developed techniques to effectively solve the problem (5): a quite time-consuming task,

especially for large training sets. Most methods decompose large-scale problems into a series of smaller ones. The most widely used method is that of Platt [7] and it is known as sequential minimal optimization.

Kernel Methods

In SVMs, both the learning problem and the decision function can be formulated only in terms of dot products between data points. Other popular methods (i.e., principal component analysis, canonical correlation analysis, fisher discriminant) have the same property. This fact has led to a huge number of algorithms that effectively use kernels to deal with nonlinear functions keeping the same complexity as the linear case. They are referred to as *kernel methods* [8, 9].

Choosing the Kernel

The main design choice when using SVMs is the selection of an appropriate kernel function, a problem of model selection that roughly relates to the choice of a topology for a neural network. It is a nontrivial result [10] that also this key task can be translated into a convex optimization problem (a semi-definite program) under general conditions. A kernel can be optimally selected from a kernel space resulting from all linear combinations of a basic set of kernels.

Kernels for General Data

Kernels are not just useful tools to allow us to deploy methods of linear statistics in a nonlinear setting. They also allow us to apply them to nonvectorial data: kernels have been designed to operate on sequences, graphs, text, images, and many other kinds of data [8].

Applications

Since their emergence, SVMs have been widely used in a huge variety of applications. To give some examples, good results have been obtained in text categorization, handwritten character recognition, and biosequence analysis.

Text Categorization

In automatic text categorization, text documents are classified into a fixed number of predefined categories based on their content. In the works performed by Joachims [11] and Dumais et al. [12], documents are represented by vectors with the so-called bag-of-words approach used in the information retrieval field. The distance between two documents is given by the inner product between the corresponding vectors. Experiments on the collection of Reuters news stories showed good results for SVMs compared to other classification methods.

Handwritten Character Recognition

This is the first real-world task on which SVMs were tested. In particular two publicly available data sets (USPS and NIST) have been considered since they are usually used for benchmarking classifiers. A lot of experiments, mainly summarized in [13], were performed which showed that SVMs can perform as well as other complex systems without incorporating any detailed prior knowledge about the task.

Bioinformatics

SVMs have been widely used also in bioinformatics. For example, Jaakkola and Haussler [14] applied SVMs to the problem of protein homology detection, i.e., the task of relating new protein sequences to proteins whose properties are already known. Brown et al. [15] describe a successful use of SVMs for the automatic categorization of gene expression data from DNA microarrays.

URL to Code

Many free software implementations of SVMs are available at the website

- www.support-vector.net/software.html

Two in particular deserve a special mention for their efficiency:

- *SVMlight*: Joachims T. Making large-scale SVM learning practical. In: Schölkopf B, Burges CJC, and Smola AJ (eds) Advances in Kernel Methods Support Vector Learning, MIT Press, 1999. Software available at http://svmlight.joachims.org
- *LIBSVM*: Chang CC, and Lin CJ, LIBSVM: a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

Cross-References

▸ PAC Learning
▸ Perceptron Algorithm

Recommended Reading

1. Boser B, Guyon I, Vapnik V (1992) A training algorithm for optimal margin classifiers. In: Proceedings of the fifth annual workshop on computational learning theory, Pittsburgh
2. Cristianini N, Shawe-Taylor J (2000) An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press, Cambridge. Book website: www.support-vector.net
3. Vapnik V (1995) The nature of statistical learning theory. Springer, New York
4. Cortes C, Vapnik V (1995) Support-vector network. Mach Learn 20:273–297
5. Hastie T, Rosset S, Tibshirani R, Zhu J (2004) The entire regularization path for the support vector machine. J Mach Learn Res 5:1391–1415
6. Drucker H, Burges CJC, Kaufman L, Smola A, Vapnik V (1997) Support vector regression machines. Adv Neural Inf Process Syst (NIPS) 9:155–161. MIT
7. Platt J (1999) Fast training of support vector machines using sequential minimal optimization. In: Schölkopf B, Burges CJC, Smola AJ (eds) Advances in kernel methods support vector learning. MIT, Cambridge, pp 185–208
8. Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, Cambridge. Book website: www.kernel-methods.net
9. Scholkopf B, Smola AJ (2002) Learning with kernels. MIT, Cambridge
10. Lanckriet GRG, Cristianini N, Bartlett P, El Ghaoui L, Jordan MI (2004) Learning the kernel matrix with semidefinite programming. J Mach Learn Res 5:27–72
11. Joachims T (1998) Text categorization with support vector machines. In: Proceedings of European conference on machine learning (ECML), Chemnitz
12. Dumais S, Platt J, Heckerman D, Sahami M (1998) Inductive learning algorithms and representations for text categorization. In: 7th international conference on information and knowledge management, Bethesda
13. LeCun Y, Jackel LD, Bottou L, Brunot A, Cortes C, Denker JS, Drucker H, Guyon I, Muller UA, Sackinger E, Simard P, Vapnik V (1995) Comparison of learning algorithms for handwritten digit recognition. In: Fogelman-Soulie F, Gallinari P (eds) Proceedings international conference on artificial neural networks (ICANN), Paris, vol 2. EC2, pp 5360
14. Jaakkola TS, Haussler D (1999) Probabilistic kernel regression models. In: Proceedings of the 1999 Conference on AI and Statistics, Fort Lauderdale
15. Brown M, Grundy W, Lin D, Cristianini N, Sugnet C, Furey T, Ares M Jr, Haussler D (2000) Knowledge-based analysis of mircoarray gene expression data using support vector machines. Proc Natl Acad Sci 97(1):262–267

Surface Reconstruction

Nina Amenta
Department of Computer Science, University of California, Davis, CA, USA

Keywords

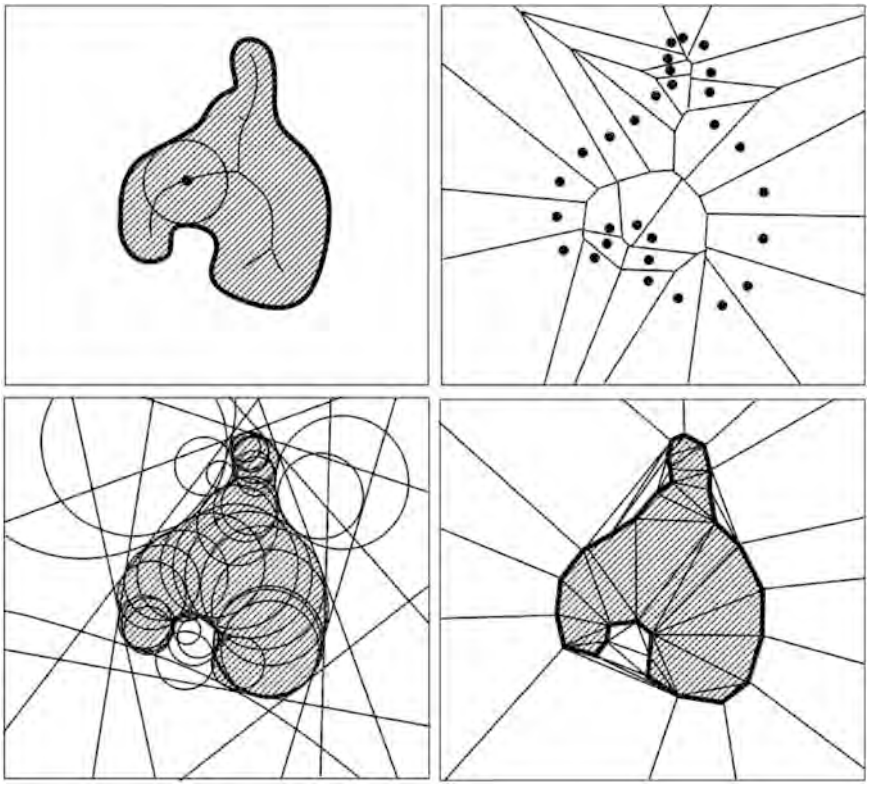
Delaunay triangulation; Local feature size; Medial axis; Surface reconstruction; Voronoi diagram

Years and Authors of Summarized Original Work

1999; Amenta, Bern
2000; Amenta, Choi, Dey, Leekha
2001; Amenta, Choi, Kolluri
2004; Dey, Goswami

Problem Definition

Surface reconstruction, here, is the problem of producing a piecewise-linear representation of a two-dimensional surface S in $\mathbb{R}^3$, given as input a set P of point samples from the surface. Very



Surface Reconstruction, Fig. 1 The medial axis of an object; the Voronoi diagram of a set of samples from the object boundary; the set of polar balls, with those inside the object shaded; the corresponding cells of the weighted Voronoi diagram, again with those inside the object shaded

sparse sets of point samples clearly do not convey much about S, so in order to prove correctness, we need to assume that the sample P is somehow sufficiently dense. The minimum required density could vary across the surface, with more detailed areas requiring denser sampling. This idea is captured in the following definition [2]. Let S be a two-dimensional surface in $\mathbb{R}^3$. The *medial axis* of S is the closure of the set of points that have more than one nearest point on S; a two-dimensional example is shown in Fig. 1, top left.

Definition 1 The *local feature size* $f(x)$ at a point x is the minimum distance from x to the medial axis of S.

The distance from the medial axis to the surface is zero at a sharp feature such as a corner or a crease, so we usually assume that S is smooth. The algorithms described here make the following ϵ-*sampling assumption*: the minimum distance, at any surface point x, to the nearest sample point is at most $\epsilon f(x)$, for some small constant ϵ. This leads to algorithms that are provably correct in the following sense.

INPUT*: A point set P that is an ϵ-sample from a smooth surface S without boundary.*

OUTPUT*: A piecewise-linear manifold without boundary, homeomorphic to S, that everywhere lies within distance $O(\epsilon f(x))$ of S.* The monograph [7] is an excellent reference for this line of research.

Key Results

One key idea is that in the neighborhood of any point $p \in P$ sampled from S, the surface is well approximated by a plane. Specifically, for any surface point x closer to p than to any other

sample, the distance of x from the tangent plane at p is $O(\epsilon f(x))$, as is the difference between the surface normal at x and the surface normal at p [2] (with the corrected proof [3]). Another key idea is that some subset of the Voronoi vertices of P approximates the medial axis of S, as in Fig. 1, top right.

Crust Algorithm

The crust algorithm [2] approximates the medial axis with a subset of the three-dimensional Voronoi vertices, called the poles. Each sample point in $p \in P$ selects the vertex of its Voronoi cell farthest from p as its first pole and the vertex farthest in the opposite direction as its second. We then eliminate any Delaunay triangle all of whose circumspheres contain a pole; this is easy to implement by computing the Delaunay triangulation of the set P augmented with the set of poles and eliminating any output triangle adjacent to a pole. A subset of the remaining surface triangles can then be selected as the piecewise-linear output surface.

Cocone Algorithm

The cocone algorithm [4] provides a simpler way of selecting a set of surface Delaunay triangles, requiring only one Voronoi diagram computation. It relies on the fact that the direction vector from a sample $p \in P$ to its first pole is within $O(\epsilon)$ of the surface normal at p, under the ϵ-sampling assumption. We define the cocone at p as the complement of a double cone, such that the angle between the cone surface and this approximate normal vector is at least $\pi - \pi/8$. We consider the intersection of the cocone at p with the Voronoi cell of p; the Delaunay triangles dual to any edge in this intersection are marked as potential surface triangles. Triangles marked by all three of their vertices are included in the set of surface triangles.

Powercrust Algorithm

While it is easy in theory to select a subset of the surface triangles to form a piecewise-linear output surface, it can be difficult in practice when the sampling density fails to meet the assumption, as is inevitable at sharp features. The power crust algorithm [5] eliminates this issue by producing a piecewise-linear output surface. The Voronoi ball centered at a pole is the ball with its nearest input samples on the boundary; see Fig. 1, lower left. We begin by labeling the Voronoi balls of all of the poles either as inside or outside the object bounded by S, using an iterative algorithm. We then compute the weighted Voronoi diagram, also known as the power diagram, of these polar Voronoi balls. Any Voronoi face separating the cell of an inner pole from the cell of an outer pole is output as part of the surface (Fig. 1, lower right). The faces of the piecewise-linear output surface are convex polygons but not in general triangles.

Noisy Samples

When the input sample points have noise, not every pole will be near the medial axis. Nonetheless, if the level of noise is everywhere small relative to the local feature size f, some subset of Voronoi vertices will still approximate the medial axis. In [8], this idea is developed into a provably correct algorithm. In addition to the ϵ-sampling assumption, we need to assume that the noise level is $O(\epsilon^2 f(x))$ and that the distance from any sample p to the kth nearest sample p' is $O(\epsilon f(x))$. This allows us to recognize a Voronoi vertex of p as a pole only when it is significantly farther from p than the k-nearest neighbors of p. These poles are then labeled as either inner or outer. This algorithm produces a triangulation of the boundary of the union of the inner polar balls as the output surface.

Complexity

The complexity of all of these algorithms depends on the complexity of the Voronoi diagram. While in general the Voronoi diagram of n points in $\mathbb{R}^3$ might have complexity $O(n^2)$, Attali, Boissonnat, and Lietier [6] proved that the complexity of the Voronoi diagram for points distributed uniformly on a nondegenerate smooth surface in $\mathbb{R}^3$ is $O(n \lg n)$. Another idea, employed by Funke and Ramos [9] and advanced by Cheng et al. [12], is to replace the Voronoi diagram with a less computationally expensive structure to get an $O(n \lg n)$ algorithm.

Applications

Interest in this problem was motivated by the advent of laser-range and LiDAR scanners [10], which produce depth maps sampled by point clouds. It is often reasonable to assume noise-free surface samples, since there are preprocessing methods, such as moving least squares (MLS) [1], that attract noisy point clouds onto nearby surfaces; there has also been theoretical work on MLS. MLS, or simply local plane-fitting, can be used to produce a normal vector at each sample point. Another common assumption is that the normal vectors can be consistently oriented. Poisson surface reconstruction [11] is an optimization technique that constructs manifold surfaces from possibly noisy points with normals. Because of its very efficient implementations, it is currently the most popular method in practice.

Open Problems

Subsequent work in surface reconstruction, both in computer graphics and in computational geometry, has focused on the identification and reconstruction of sharp features and then using them to construct surfaces that are non-manifold. Proving that the complexity of the Voronoi diagram of points distributed on a generic smooth surface with noise or with boundary is $o(n^2)$ remains open.

URLs to Code and Data Sets

There is code available for the cocone algorithm (http://web.cse.ohio-state.edu/~tamaldey/cocone.html), with several subsequent variants. There is also code for the power crust algorithm (http://www.cs.ucdavis.edu/~amenta/powercrust.html). There is a set of benchmark data sets for surface reconstruction (http://www.cs.utah.edu/~bergerm/recon_bench).

Cross-References

- ▸ Curve Reconstruction
- ▸ Manifold Reconstruction

Recommended Reading

1. Alexa M, Behr J, Cohen-Or D, Fleishman S, Levin D, Silva CT (2003) Computing and rendering point set surfaces. IEEE Trans Vis Comput Graph 9(1):3–15
2. Amenta N, Bern M (1999) Surface reconstruction by Voronoi filtering. Discret Comput Geom 22(4):481–504
3. Amenta N, Dey TK (2007) Normal variation for adaptive feature size, arXiv
4. Amenta N, Choi S, Dey TK, Leekha N (2000) A simple algorithm for homeomorphic surface reconstruction. In: Proceedings of the sixteenth annual symposium on computational geometry, Hong Kong. ACM, pp 213–222
5. Amenta N, Choi S, Kolluri RK (2001) The power crust, unions of balls, and the medial axis transform. Comput Geom Theory Appl 19(2):127–153
6. Attali D, Boissonnat JD, Lieutier A (2003) Complexity of the Delaunay triangulation of points on surfaces: the smooth case. In: Proceedings of the nineteenth annual symposium on computational geometry, San Diego. ACM, pp 201–210
7. Dey TK (2006) Curve and surface reconstruction: algorithms with mathematical analysis. Cambridge monographs on applied and computational mathematics. Cambridge University Press, Leiden
8. Dey TK, Goswami S (2004) Provable surface reconstruction from noisy samples. In: Proceedings of the twentieth annual symposium on computational geometry, Brooklyn. ACM, pp 330–339
9. Funke S, Ramos EA (2002) Smooth-surface reconstruction in near-linear time. In: Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms, San Francisco. Society for Industrial and Applied Mathematics, pp 781–790
10. Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W (1992) Surface reconstruction from unorganized points. ACM Trans Graph (TOG) 26(2):71–78
11. Kazhdan M, Bolitho M, Hoppe H (2006) Poisson surface reconstruction. In: Proceedings of the fourth eurographics symposium on geometry processing, Cagliari, pp 61–70
12. Cheng S-W, Jin J, Lau M-K (2012) A fast and simple surface reconstruction algorithm. In: Proceedings of the 28th annual symposium on computational geometry, Chapel Hill, pp 69–78

Symbolic Model Checking

Adnan Aziz[1] and Amit Prakash[2]
[1]Department of Electrical and Computer Engineering, University of Texas, Austin, TX, USA
[2]Microsoft, MSN, Redmond, WA, USA

Keywords

Formal hardware verification

Years and Authors of Summarized Original Work

1990; Burch, Clarke, McMillan, Dill

Problem Definition

Design verification is the process of taking a design and checking that it works correctly. More specifically, every design verification paradigm has three components [6]: (1) a language for specifying the design in an unambiguous way, (2) a language for specifying properties that are to be checked of the design, and (3) a checking procedure, which determines whether the properties hold off the design.

The verification problem is very general: it arises in low-level designs, e.g., checking that a combinational circuit correctly implements arithmetic, as well as high-level designs, e.g., checking that a library written in high-level language correctly implements an abstract data type.

Hardware Verification

The verification of hardware designs is particularly challenging. Verification is difficult in part because the large number of concurrent operations make it very difficult to conceive of and construct all possible corner cases, e.g., one unit initiating a transaction at the same cycle as another receiving an exception. In addition, software models used for simulation run orders of several magnitude slower than the final chip operates at. Faulty hardware is usually impossible to correct after fabrication, which means that the cost of a defect is very high, since it takes several months to go through the process of designing and fabricating new hardware. Wile et al. [15] provide a comprehensive account of hardware verification.

State Explosion

Since the number of state-holding elements in digital hardware is bounded, the number of possible states that the design can be in is infinite, so complete automated verification is, in principle, possible. However, the number of states that a hardware design can reach from the initial state can be exponential in the size of the design; this phenomenon is referred to as "state explosion." In particular, algorithms for verifying hardware that explicitly record visited states, e.g., in a hash table, have very high time complexity, making them infeasible for all but the smallest designs. The problem of complete hardware verification is known to be PSPACE-hard, which means that any approach must be based on heuristics.

Hardware Model

A hardware design is formally described using *circuits* [4, 8]. A *combinational circuit* consists of *Boolean combinational elements* connected by *wires*. The Boolean combinational elements are *gates* and *primary inputs*. Gates come in three types: *NOT, AND*, and *OR*. The NOT gate functions as follows: it takes a single Boolean-valued *input* and produces a single Boolean-valued *output* which takes value 0 if the input is 1 and 1 if the input is 0. The AND gate takes two Boolean-valued inputs and produce a single output; the output is 1 if both inputs are 1 and 0 otherwise. The OR gate is similar to AND, except that its output is 1 if one or both inputs are 1. A circuit can be represented as a directed graph where the nodes represent the gates and

wires represent edges in the direction of signal flow.

A circuit can be represented by a directed graph where the nodes represent the gates and primary inputs, and edges represent wires in the direction of signal flow. Circuits are required to be acyclic, that is, there is no cycle of gates. The absence of cycles implies that a Boolean assignment to the primary inputs can be propagated through the gates in topological order.

A *sequential circuit* extends the notion of circuit described above by adding *stateful elements*. Specifically, a sequential circuit includes *registers*. Each register has a single input, which is referred to as its *next-state input*.

A *valuation* on a set V is a function whose domain is V. A *state* in a sequential circuit is a Boolean-valued valuation on the set of registers. An *input* to a sequential circuit is a Boolean-valued valuation on the set of primary inputs. Given a state s and an input i, the logic gates in the circuit uniquely define a Boolean-valued valuation t to the set of register inputs – this is referred to as the next state of the circuit at state s under input i and say s *transitions* to t on input i. It is convenient to denote such a transition by $s \xrightarrow{i} t$.

A sequential circuit can naturally be identified with a *finite state machine* (FSM), which is a graph defined over the set of all states; an edge (s, t) exists in the FSM graph if there exists an input i, state s transitions to t on input i.

Invariant Checking

An *invariant* is a set of states; informally, the term is used to refer to a set of states that are "good" in some sense. One common way to specify an invariant is to write a Boolean formula on the register variables – the states which satisfy the formula are precisely the states in the invariant.

Given states r and s, define r to be *reachable* from s if there is a sequence of inputs $i_0, i_1, \ldots, i_{n-1}$ such that $s = s_0 \xrightarrow{i_0} s_1 \xrightarrow{i_1} \cdots s_n = t$. A fundamental problem in hardware verification is the following: given an invariant A, and a state s, does there exists a state r reachable from s which is not in A?

Key Results

Symbolic model checking (SMC) is a heuristic approach to hardware verification. It is based on the idea that rather than representing and manipulating states one at a time, it is more efficient to use symbolic expressions to represent and manipulate sets of states.

A key idea in SMC is that given a set $A \subset \{0,1\}^n$, a Boolean function A can be constructed such that f_A: $\{0,1\}^n \to \{0,1\}$ given by $f(\alpha_1, \ldots, \alpha_n) = 1$ iff $(\alpha_1, \ldots, \alpha_n) \in A$. Note that given a characteristic function f_A, A can be obtained and vice versa.

There are many ways in which a Boolean function can be represented: formulas in DNF, general Boolean formulas, combinational circuits, etc. In addition to an efficient representation for state sets, the ability to perform fast computations with sets of states is also important, for example, in order to determine if an invariant holds, it is required to compute the set of states reachable from a given state. BDDs [2] are particularly well suited to representing Boolean functions, as they combine succinct representation with efficient manipulation; they are the data structure underlying SMC.

Image Computation

A key computation that arises in verification is determining the *image* of a set of states A in a design D – the image of A is the set of all states t for which there exists a state in A and an input i such that state s transitions to t under input i. The image of A is denoted by $\mathrm{Img}(A)$.

The *transition relation* of a design is the set of (s, i, t) triples such that s transitions to t under input i. Let the design have n registers and m primary inputs; then the transition relation is subset of $\{0,1\}^n \times \{0,1\}^m \times \{0,1\}^n$.

Conceptually, the transition relation completely captures the dynamics of the design – given an initial state, and input sequence, the evolution of the design is completely determined by the transition relation.

Since the transition relation is a subset of $\{0,1\}^{n+m+n}$, it has a characteristic function $f_T : \{0,1\}^{n+m+n} \rightarrow \{0,1\}$. View f_T as being defined over the variables $x_0, \ldots, x_{n-1}$, $i_0, \ldots, i_{m-1}$, $y_0, \ldots, y_{n-1}$. Let the set of states A be represented by the function f_A defined over variables $x_0, \ldots, x_{n-1}$. Then the following identity holds

$$\mathrm{Img}(A) = (\exists x_0 \cdot \exists x_{n-1} \exists i_0 \cdots \exists i_{m-1})(f_A \cdot f_T).$$

The identity holds because $(\beta_0, \ldots, \beta_{n-1})$ satisfies the right-hand side expression exactly when there are values $\alpha_0, \ldots, \alpha_{n-1}$, and $\iota_0, \ldots, \iota_{m-1}$ such that $(\alpha_0, \ldots, \alpha_{n-1}) \in A$ and the state $(\alpha_0, \ldots, \alpha_{n-1})$ transitions to $(\beta_0, \ldots, \beta_{n-1})$ on input $(\iota_0, \ldots, \iota_{m-1})$.

Invariant Checking

The set of all states reachable from a given set A is the limit as n tends to infinity of the sequence of states $R_0, R_1, \ldots$ defined below:

$$R_0 = A$$
$$R_{i+1} = R_i \cup \mathrm{Img}(R_i).$$

Since for all i, $R_i \subseteq R_{i+1}$ and the number of distinct state sets is finite, the limit is reached in some finite number of steps, i.e., for some n, it must be that $R_{n+1} = R_n$. It is straightforward to show that the limit is exactly equal to the set of states reachable from A – the basic idea is to inductively construct input sequences that lead from states in A to R_i and to show that state t is reachable from a state in A under an input sequence of length l, then t must be in R_l.

Given BDDs F and G representing functions f and g, respectively, there is an algorithm based on dynamic programming for performing conjunction, i.e., for computing the BDD for $f \cdot g$. The algorithm has polynomial complexity, specifically $O(|F| \cdot |G|)$, where $|B|$ denotes the number of nodes in the BDD B. There are similar algorithms for performing disjunction ($f + g$) and computing cofactors (f_x and $f_{x'}$). Together these yield an algorithm for the operation of existential quantification, since $(\exists x) f = f_x + f'_x$.

It is straightforward to build BDDs for f_A and $f_T : A$ is typically given using a propositional formula, and the BDD for f_A can be built up using functions for conjunction, disjunction, and negation. The BDD for f_T is built using from the BDDs for the next-state nodes, over the register and primary input variables. Since the only gate types are AND, OR, and NOT, the BDD can be built using the standard BDD operators for conjunction, disjunction, and negation. Let the next-state functions be $f_0, \ldots, f_{n-1}$; then f_T is $(y_0 = f_0) \cdot (y_1 = f_1) \cdot \cdots \cdot (y_{n-1} = f_{n-1})$, and so the BDD for f_T can be constructed using the usual BDD operators.

Since the image computation operation can be expressed in terms of f_A and F_T, and conjunction and existential quantification operations, it can be performed using BDDs. The computation of R_i involves an image operation, and a disjunction, and since BDDs are canonical, the test for fixed point is trivial.

Applications

The primary application of the technique described above is for checking properties of hardware designs. These properties can be invariants described using propositional formulae over the register variables, in which case the approach above is directly applicable. More generally, properties can be expressed in a *temporal logic* [5], specifically through formulae which express acceptable sequences of outputs and transitions.

CTL is one common temporal logic. A CTL formula is given by the following grammar: if x is a variable corresponding to a register, then $\mathbf{x}$ is a CTL formula; otherwise, if φ and ψ are CTL

formulas, then so as $(\neg\phi)$, $(\phi \vee \psi)$, $(\phi \wedge \psi)$, $(\phi \rightarrow \psi)$, and $EX\phi$, $E\phi U\psi$, and $EG\phi$.

A CTL formula is interpreted as being true at a state; a formula **x** is true at a state if that register is 1 in that state. Propositional connectives are handled in the standard way, e.g., a state satisfies a formula $(\phi \wedge \psi)$ if it satisfies both φ and ψ. A state s satisfies $EG\phi$ if there exists a state t such that s transitions to, and t satisfies φ. A state s satisfies $E\phi U\psi$ if there exists a sequence of inputs $i_0, \ldots, i_n$ leading through state $s_0 = s$, $s_1, s_2, \ldots, s_{n+1}$ such that s_{n+1} satisfies ψ, and all states $s_i, i \leq n+1$ satisfy φ. A state s satisfies $EG\phi$ if there exists an infinite sequence of inputs $i_0, i_1, \ldots$ leading through state $s_0 = s, s_1, s_2, \ldots$ such that all states s_i satisfy φ.

CTL formulas can be checked by a straightforward extension of the technique described above for invariant checking. One approach is to compute the set of states in the design satisfying subformulas of φ, starting from the subformulas at the bottom of the parse tree for φ. A minor difference between invariant checking and this approach is that the latter relies on *pre-image* computation; the pre-image of A is the set of all states t for which there exists an input i such that t transitions under i to a state in A.

Symbolic analysis can also be used to check the equivalence of two designs by forming a new design which operates the two initial designs in parallel and has a single output that is set to 1 if the two initial designs differ [14]. In practice this approach is too inefficient to be useful, and techniques which rely more on identifying common substructures across designs are more successful.

The complement of the set of reachable states can be used to identify parts of the design which are redundant and to propagate don't care conditions from the input of the design to internal nodes [12].

Many of the ideas in SMC can be applied to software verification – the basic idea is to "finitize" the problem, e.g., by considering integers to lie in a restricted range or setting an a priori bound on the size of arrays [7].

Experimental Results

Many enhancements have been made to the basic approach described above. For example, the BDD for the entire transition relation can grow large, so *partitioned transition relations* [11] are used instead; these are based on the observation that $\exists x.(f \cdot g) = f \cdot \exists x.g$, in the special case that f is independent of x. Another optimization is the use of *don't cares*; for example, when computing the image of A, the BDD for f_T can be simplified with respect to transitions originating at A' [13]. Techniques based on SAT have enjoyed great success recently. These approaches case the verification problem in terms of satisfiability of a CNF formula. They tend to be used for bounded checks, i.e., determining that a given invariant holds on all input sequences of length k [1]. Approaches based on *transformation-based verification* complement symbolic model checking by simplifying the design prior to verification. These simplifications typically remove complexity that was added for performance rather than functionality, e.g., pipeline registers.

The original paper by Clarke et al. [3] reported results on a toy example, which could be described in a few dozen lines of a high-level language. Currently, the most sophisticated model checking tool for which published results are ready is SixthSense, developed at IBM [10].

A large number of papers have been published on applying SMC to academic and industrial designs. Many report success on designs with an astronomical number of states – these results become less impressive when taking into consideration the fact that a design with n registers has 2^n states.

It is very difficult to define the complexity of a design. One measure is the number of registers in the design. Realistically, a hundred registers is at the limit of design complexity that can be handled using symbolic model checking. There are cases of designs with many more registers that have been successfully verified with symbolic model checking, but these registers are invariably part of a very regular structure, such as a memory array.

Data Sets

The SMV system described in [9] has been updated, and its latest incarnation nuSMV (http://nusmv.irst.itc.it/) includes a number of examples.

The VIS (http://embedded.eecs.berkeley.edu/pubs/downloads/vis) system from UC Berkeley and UC Boulder also includes a large collection of verification problems, ranging from simple hardware circuits to complex multiprocessor cache systems.

The SIS (http://embedded.eecs.berkeley.edu/pubs/downloads/sis/) system from UC Berkeley is used for logic synthesis. It comes with a number of sequential circuits that have been used for benchmarking symbolic reachability analysis.

Cross-References

▸ Binary Decision Graph

Recommended Reading

1. Biere A, Cimatti A, Clarke E, Fujita M, Zhu Y (1999) Symbolic model checking using sat procedures instead of BDDs. In: ACM design automation conference, New Orleans
2. Bryant R (1986) Graph-based algorithms for Boolean function manipulation. IEEE Trans Comput C-35:677–691
3. Burch JR, Clarke EM, McMillan KL, Dill DL (1992) Symbolic model checking: 10^{20} states and beyond. Inf Comput 98(2):142–170
4. Cormen TH, Leiserson CE, Rivest RH, Stein C (2001) Introduction to algorithms. MIT, Cambridge
5. Emerson EA (1990) Temporal and modal logic. In: van Leeuwen J (ed) Formal models and semantics. Volume B of handbook of theoretical computer science. Elsevier Science, Amsterdam, pp 996–1072
6. Gupta A (1993) Formal hardware verification methods: a survey. Form Method Syst Des 1:151–238
7. Jackson D (2006) Software abstractions: logic, language, and analysis. MIT, Cambridge
8. Katz R (1993) Contemporary logic design. Benjamin/Cummings Publishing Company, Redwood City
9. McMillan KL (1993) Symbolic model checking. Kluwer Academic, Boston
10. Mony H, Baumgartner J, Paruthi V, Kanzelman R, Kuehlmann A (2004) Scalable automated verification via expert-system guided transformations. In: Formal methods in CAD, Austin
11. Ranjan R, Aziz A, Brayton R, Plessier B, Pixley C (1995) Efficient BDD algorithms for FSM synthesis and verification. In: Proceedings of the international workshop on logic synthesis, Tahoe City, May 1995
12. Savoj H (1992) Don't cares in multi-level network optimization. Ph.D. thesis, Electronics Research Laboratory, College of Engineering, University of California, Berkeley
13. Shiple TR, Hojati R, Sangiovanni-Vincentelli AL, Brayton RK (1994) Heuristic minimization of BDDs using don't cares. In: ACM design automation conference, San Diego, June 1994
14. Touati H, Savoj H, Lin B, Brayton RK, Sangiovanni-Vincentelli AL (1990) Implicit state enumeration of finite state machines using BDDs. In: IEEE international conference on computer-aided design, Santa Clara, pp 130–133, Nov 1990
15. Wile B, Goss J, Roesner W (2005) Comprehensive functional verification. Morgan-Kaufmann

Symmetric Graph Drawing

Seokhee Hong
School of Information Technologies, University of Sydney, Sydney, NSW, Australia

Keywords

Graph automorphism; Graph drawing; Planar graph; Symmetry

Years and Authors of Summarized Original Work

2006; Hong, McKay and Eades

Problem Definition

Symmetry is one of the most important aesthetic criteria in graph drawing that clearly reveals the structure and properties of a graph. Many graphs in Graph Theory textbooks are often symmetric.

A symmetry of a drawing D of a graph G induces an *automorphism* ϕ of the graph G, a permutation of the vertex set that preserves

adjacency. If an automorphism ϕ can be displayed as a symmetry in a drawing of the graph G, then it is called a *geometric automorphism* [6]. A geometric automorphism ϕ of a *planar* graph G is a *planar automorphism*, if there is a *planar* drawing of G which displays ϕ. Note that not every automorphism is geometric, and not every geometric automorphism is planar.

In general, algorithms for constructing symmetric drawings of graphs have two steps:

1. *Symmetry finding step*: Find the geometric automorphisms of a graph
2. *Symmetry drawing step*: Draw the graph displaying these automorphisms as symmetries.

Note that the first step is more difficult than the second step. For example, finding automorphism of a graph is isomorphism-hard; however finding geometric automorphism of a graph is NP-hard in general [18]. For planar graphs, computing isomorphism (therefore, automorphism) of a graph can be solved in linear time [7, 17]. However, finding the *best plane embedding* of planar graphs that displays the maximum number of symmetries in a drawing of a planar graph is challenging, because a planar graph can have exponential number of possible plane embeddings.

Furthermore, the product of two geometric automorphisms is not necessarily geometric, because they may be displayed by different drawings. A subgroup A of the automorphism group of a graph is a *geometric automorphism group*, if there is a single drawing of the graph that displays every element of A. Therefore, to construct a maximally symmetric drawing of a graph, one needs to compute a *maximum size* geometric automorphism group for the graph. Therefore, the main research problem for Symmetric Graph Drawing can be defined as below.

Symmetric Graph Drawing Problem
Input: A graph G.
Output: A maximum size geometric automorphism group A of G, A symmetric drawing D of G that displays all elements of A.

Key Results

There are two types of symmetry in two-dimensional drawings: *rotational symmetry* (i.e., a rotation about a point) and *axial* (or *reflectional*) *symmetry* (i.e., a reflection about an axis). The *order* of an automorphism α is the smallest positive integer k such that α^k equals the identity I. A group-theoretic characterization of geometric automorphism group was given by Eades and Lin [6] as follows:

- A group of order 2 generated by an *axial automorphism*;
- A *cyclic group* of order k generated by a *rotational automorphism*;
- A *dihedral group* of order $2k$ generated by a rotational automorphism of order k and an axial automorphism. In this case there are k axial symmetries.

In two dimensions, the problem of determining whether a given graph can be drawn symmetrically is NP-complete in general [18]. Exact algorithms are devised based on Branch and Cut approach by Buchheim and Junger [3] and a group-theoretic approach by Abelson et al. [1]. Linear-time algorithms are available for trees and outerplanar graphs by Manning and Atallah [19, 20] and for series-parallel digraphs by Hong et al. [14]. Linear-time algorithms are presented for maximally symmetric drawings of triconnected planar graphs by Hong et al. [15] and for biconnected, oneconnected, and disconnected planar graphs by Hong and Eades [10, 12, 13]. Hong and Nagamochi presented a linear-time algorithm for constructing a *symmetric convex* drawings of internally trconnected planar graphs [16]. For a survey on symmetric drawings of graphs in two dimensions, see [5].

In three dimensions, the problem of determining whether a graph can be drawn symmetrically in three dimensions is *NP-hard* in general [8]. A group-theoretic characterization of symmetric drawing in n-dimensions and exact algorithms based on a group-theoretic approach are given

by Abelson et al. [1]. Linear-time algorithms are available for trees by Hong and Eades [9], series-parallel digraphs by Hong et al. [11], and biconnected and oneconnected planar graphs [8].

In this article, we review a linear-time algorithm for constructing maximally symmetric straight-line drawings of *triconnected planar graphs* by Hong, McKay, and Eades [15]. The following theorem summarizes their main results.

Theorem 1 *There is a linear-time algorithm that constructs straight-line drawings of maximally symmetric planar drawings of triconnected planar graphs.*

Computing a Planar Automorphism Group of Maximum Size

We first review the first step of the algorithm, i.e., symmetry finding step for triconnected planar graphs [15]. A geometric automorphism group A of a graph G is a *planar automorphism group*, if there is a *planar* drawing of the graph that displays every element of A.

Suppose that A is a group acting on a set X. The *stabilizer* of $x \in X$, denoted by $stab_A(x)$, is $\{g \in A \mid g(x) = x\}$, and the *orbit* of x, denoted by $orbit_A(x)$, is $\{g(x) \mid g \in A\}$. We say that $g \in A$ *fixes* $x \in X$ if $g(x) = x$; if g fixes x for every $g \in A$, then A fixes x. If $X' \subseteq X$ and $\phi(x') \in X'$ for all $x' \in X'$, then g *fixes* X'. Automorphisms $g_1, g_2, \ldots, g_k$ are called *generators* of $\langle g_1, g_2, \ldots, g_k \rangle$; the group consists of all permutations formed from products of elements of $\{g_1, g_2, \ldots, g_k\}$.

Hong et al. [15] characterize planar automorphisms as below.

Lemma 1 *Let G be a triconnected planar graph. An automorphism of G is a planar automorphism if and only if it fixes a face of G.*

To find the best plane embedding to compute a planar automorphism group with a maximum size, the algorithm uses the Stabilizer-Orbit theorem in group theory [2].

Theorem 2 (Stabilizer-Orbit theorem) *Suppose that A is a group acting on a set X and let $x \in X$. Then $|A| = |orbit_A(x)| \times |stab_A(x)|$.*

The overall algorithm computing a maximum size planar automorphism group of a triconnected planar graph can be described as follows;

Algorithm `Compute_Max_PAG`

1. Find a plane embedding which has a maximum size planar automorphism group.
2. Perform "star triangulation" for the given embedding.
3. Compute the *generators* of the planar automorphism group of the new embedding.

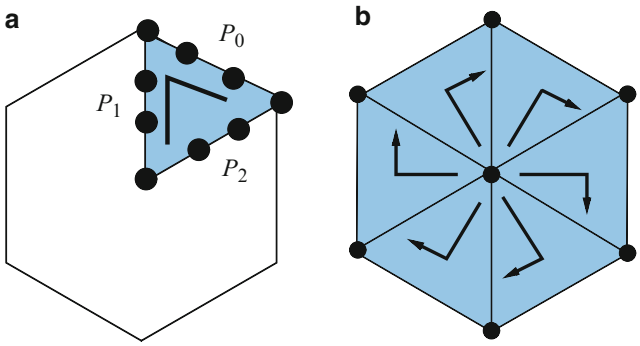
The first step of `Compute_Max_PAG` uses two applications of an algorithm of Fontet [7], which computes the orbits on vertices of the (full) automorphism group of a triconnected planar graph in linear time.

Theorem 3 *Fontet's algorithm [7] can be used to find a plane embedding of a triconnected graph G such that the corresponding planar automorphism group is maximized in linear time.*

Proof Based on Lemma 1, we take a dual graph of G^* of G and compute the orbits of G^* using Fontet's algorithm [7]. Choose an orbit O of minimum size; the stabilizer O has the maximum size, by Theorem 2. Taking a face $f \in O$ as the outer face of the plane embedding of G, we have an embedding that displays the maximum number of symmetries.

Once the outer face and thus the plane embedding is chosen, the second step of `Compute_Max_PAG` performs *star triangulation*, i.e., triangulate each internal face f by inserting a new vertex v in the face and joining v to each vertex of f. Clearly, this step takes linear time and simplifies the drawing algorithm.

The final step of `Compute_Max_PAG` is to compute the planar automorphism group for star-triangulated plane graph. Since an explicit representation of the planar automorphism group may take more than linear space, for a more compact representation, an algorithm for computing *minimal generators* was devised. For details on a linear-time algorithm for computing generators of a planar automorphism group, see [15].



Symmetric Graph Drawing, Fig. 1 Example of (**a**) a wedge and (**b**) merging step

Overview of the Drawing Algorithm

We now review a linear-time drawing algorithm for constructing a symmetric drawing of a triconnected planar graph that achieves that maximum with straight-line edges. The main characteristic of symmetric drawings is the repetition of congruent drawings of isomorphic subgraphs. To exploit this property, the drawing algorithm uses a divide and conquer approach: (i) divide the graph into isomorphic subgraphs; (ii) compute a drawing for a subgraph; and (iii) merge multiple copies of drawings of subgraphs to construct a symmetric drawing of the whole graph. Overall, each step of the drawing algorithm runs in linear time.

The input of the drawing algorithm is a triconnected planar graph with fixed plane embedding and a specified outer face, which maximize the number of symmetries. The symmetric drawing algorithm takes a different approach for each type of planar automorphism group: i.e., *cyclic* case, *one axial* case, and *dihedral* case.

The Cyclic Case

Here we describe how to display k rotational symmetries. Note that after star triangulation, there is a *central* vertex c, which is fixed by the planar automorphism group for $k \geq 3$. If $k = 2$, there exits either a central vertex or a central edge. If there is a central edge, then we preprocess the graph by inserting a dummy central vertex c into the central edge with two dummy edges.

The rotational symmetric drawing algorithm consists of three steps:

Algorithm **Cyclic**

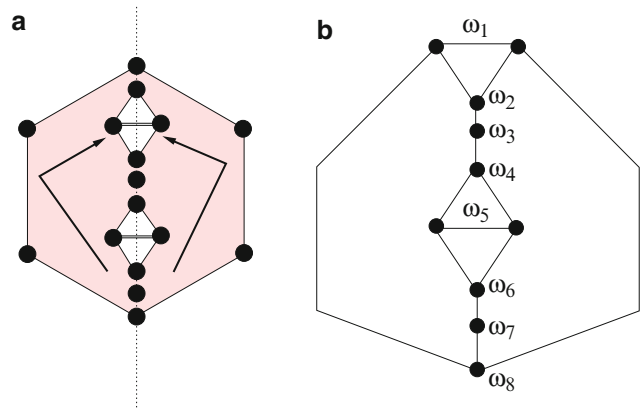
1. `Find_Wedge_Cyclic`.
2. `Draw_Wedge_Cyclic`.
3. `Merge_Wedges_Cyclic`.

The first step is to find a subgraph *wedge* W, which takes linear time:

Algorithm `Find_Wedge_Cyclic`

1. Find the *central* vertex c.
2. Find a shortest path P_1, from c to a vertex v_1 on the outer face, using breadth first search.
3. Find the path P_2 which is a mapping of P_1 under a minimal generator of the rotation.
4. Find the wedge W (see Fig. 1a), an induced subgraph of G enclosed by the cycle formed from P_1, P_2 and a path P_0 along the outer face from v_1 to v_2.

The second step, `Draw_Wedge_Cyclic`, constructs a drawing D of the wedge W using Algorithm `CYN`, the linear-time convex drawing algorithm by Chiba et al. [4], such that P_1, P_2, and P_0 are drawn as straight lines. The input to Algorithm `CYN` is an internally triconnected plane graph G with given outer face S and a straight-line drawing S^* of S as a *weakly convex polygon*, i.e., not every vertex of the outer face needs to be at an apex (i.e., the interior angle is less than π) of the polygon. Algorithm `CYN` chooses a vertex v and deletes it from G together with incident edges and divides the resulting graph $G' = G - v$ into the biconnected components $B_1, B_2, \ldots, B_p$, $p \geq 1$. It defines a convex polygon S_i^* of the outer facial cycle S_i of



Symmetric Graph Drawing, Fig. 2 Example of (**a**) a fixed string of diamonds and (**b**) ω_ℓ

each B_i and recursively applies the algorithm to draw B_i with S_i^* as outer boundary. For details, see [4].

The last step, `Merge_Wedges_Cyclic`, constructs a drawing of the whole graph G by replicating the drawing D of W, k times. Note that this merge step relies on the fact that P_1 and P_2 are drawn as straight lines. See Fig. 1b.

It is clear that Algorithm `Cyclic` constructs a straight-line drawing of a triconnected plane graph which shows k rotational symmetry in linear time.

One Axial Symmetry

Consider a drawing of a star-triangulated plane graph with one axial symmetry. There are fixed vertices, edges, and/or fixed faces on the axis; we need to characterize the subgraph formed by these.

A *diamond* is either a triangle or the 4-vertex graph. A *string of diamonds* is a graph formed from a path $P = (v_1, v_2, \ldots, v_k)$, $k \geq 2$, by a number (zero or greater) of "splitting" operations, as follows. If $1 \leq i \leq k-1$, then the edge (v_i, v_{i+1}) may be replaced by a diamond. Alternatively, each of the end edges (v_1, v_2) and (v_{k-1}, v_k) may be replaced by a triangle. Note that a string of diamonds is basically a path consisting of edges and diamonds; each end of the path may be a triangle; see Fig. 2a.

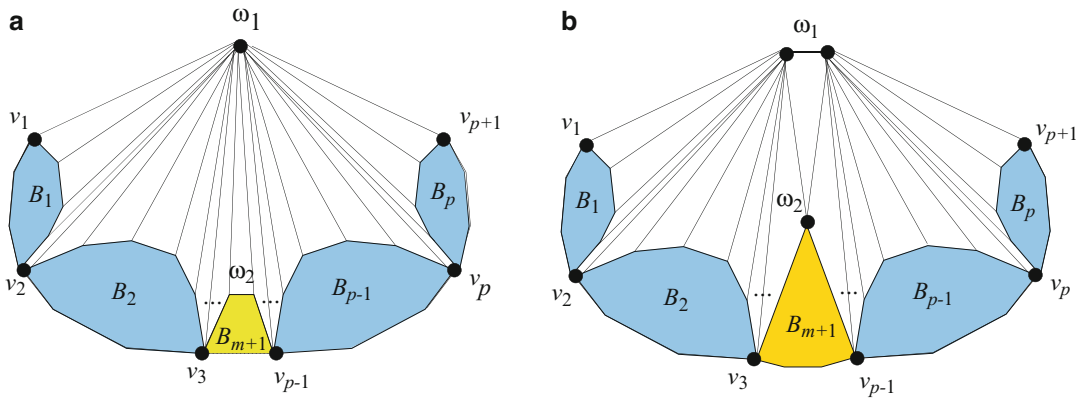
To display a single axial symmetry, we need two steps. First we identify the *fixed string of diamonds*; then use Algorithm `Symmetric_CYN`, a modified version of Algorithm `CYN`. More formally, the algorithm `One_Axial` is described below.

Algorithm `One_Axial`

1. Find a fixed string of diamonds. Suppose that $\omega_1, \omega_2, \ldots, \omega_k$ are the fixed edges and vertices in the fixed string of diamonds, in order from the outer face (ω_1 is on the outer face). For each ℓ, ω_ℓ may be a vertex or an edge (see Fig. 2b).
2. Choose a symmetric convex polygon S^* for the outer face S of G.
3. `Symmetric_CYN`$(1, S^*, G, y_1)$.

The main ingredient in Algorithm `One_Axial` is Algorithm `Symmetric_CYN`. To modify Algorithm `CYN` to display a single axial symmetry, the following three conditions should be satisfied:

- Choose the first vertex or edge on the fixed string of diamonds ω_1 (see Fig. 3).
- Let $D(B_i)$ be the drawing of B_i and α be the axial symmetry. Then, $D(B_i)$ should be a reflection of $D(B_j)$, where $B_j = \alpha(B_i)$, $i = 1, 2, \ldots, m$ and $m = \lfloor p/2 \rfloor$: To satisfy this condition, define S_j^* to be the reflection of S_i^*, $i = 1, 2, \ldots, m$. Then we apply Algorithm `CYN` for $B_i, i = 1, 2, \ldots, m$ and construct $D(B_j)$ using a reflection of $D(B_i)$.
- If p is odd, then $D(B_{m+1})$ should display axial symmetry: To satisfy this condition, we recursively apply Algorithm `Symmetric_CYN` to B_{m+1}.



Symmetric Graph Drawing, Fig. 3 Example of a symmetric version of CYN

Note that the position of ω_2 in Fig. 3 can be chosen arbitrarily along the axis of symmetry of S^* within S^*. This means that we can specify the positions of the fixed vertices and middle edges along the axis of symmetry a priori, that is, as input to the algorithm. The Algorithm `Symmetric_CYN` can be described as below:

Algorithm `Symmetric_CYN`

input: ℓ: index of vertex or middle edge on the fixed string of diamonds.
input: S^*: a weakly convex polygon of the outer face of S of G.
input: G: a triangulated planar graph.
input: y_ℓ: a position on the axis of symmetry for the fixed vertex or the fixed edge ω_ℓ.

1. Delete ω_ℓ from G together with edges incident to ω_ℓ. Divide the resulting graph $G' = G - \omega_\ell$ into the blocks $B_1, B_2, \ldots, B_p$, $p \geq 1$, ordered anticlockwise around the outer face. Let $m = \lfloor p/2 \rfloor$.
2. Determine a convex polygon S_i^* of the outer facial cycle S_i of each B_i such that B_i with S_i^* satisfy the conditions for convex drawing algorithm `CYN` and S_{p-i+1}^* is a reflection of S_i^*.
3. For each $i = 1$ to m,
   (a) Construct a drawing $D(B_i)$ of B_i using Algorithm `CYN`.
   (b) Construct $D(B_{p-i+1})$ as a reflection of $D(B_i)$.
4. If p is odd, then construct a drawing $D(B_{m+1})$ using `Symmetric_CYN`$(\ell + 1, S_{m+1}^*, B_{m+1}, y_{\ell+1})$.
5. Merge the $D(B_i)$ to form a drawing of G, placing ω_ℓ at y_ℓ.

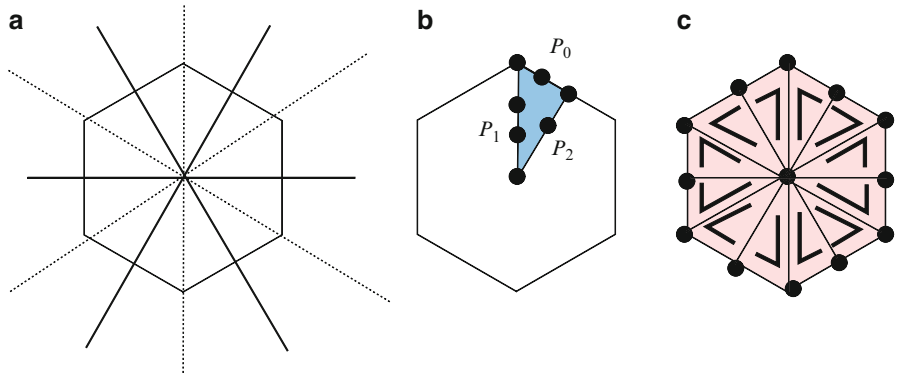
Since Algorithm `CYN` [4] runs in linear time, clearly Algorithm `Symmetric_CYN` and Algorithm `One_Axial` takes linear time.

The Dihedral Case

We now review an algorithm for displaying a dihedral group $< \rho, \alpha >$, where ρ is a rotation of order k and α is an axial automorphism. As with the cyclic case, we assume that there is a central vertex.

The drawing algorithm adopts the same strategy as for the cyclic case: (i) divide the graph into "wedges"; (ii) draw each wedge; and (iii) merge the drawings of wedges to construct a symmetric drawing of the whole graph. However, the dihedral case is more difficult than the cyclic case, because an axial symmetry in the dihedral group can have fixed faces as well as fixed edges; i.e., the boundary of a wedge may be a fixed string of diamonds as in the one axial case. To achieve dihedral symmetry, the axis of symmetry must be the perpendicular bisector of the middle edge of each diamond. This makes the merging operation more difficult.

Consider a drawing of a triconnected planar graph with a dihedral symmetry group of size $2k$.



Symmetric Graph Drawing, Fig. 4 Wedge for the dihedral case

There are k axial symmetries, with axes at angles of $\pi i/k$, $0 \leq i \leq k-1$, to the x axis, as in Fig. 4a. Roughly speaking, a wedge is the area between two adjacent axes, as in Fig. 4b. Note that in these wedges, the boundaries P_1 and P_2 may be strings of diamonds. These may terminate in a triangle.

As with the cyclic case, Algorithm `Dihedral` has three steps: (i) `Find_Wedge_Dihedral`, (ii) `Draw_Wedge_Dihedral`, and (iii) `Merge_Wedges_Dihedral`.

The first step is to define the "wedge" subgraph by finding two fixed strings of diamonds. Note that one can find the central vertex c and the two fixed strings of diamonds P_1 and P_2 in linear time using the generators of the group.

Algorithm `Find_Wedge_Dihedral`

1. Find the central vertex c.
2. Find a string of diamonds P_1 that is fixed by α, from c to a vertex v or an edge e on the outer face.
3. Traverse the outer face, clockwise from v (or e) to the vertex v' or edge e' that is fixed by $\rho^{-1}\alpha\rho$. Let P_0 denote the path so traversed.
4. Find the string of diamonds P_2 for $\rho^{-1}\alpha\rho$, from c to v' (or e').
5. Define the wedge W to be the subgraph enclosed by P_0, P_1, and P_2, including the vertices and edges of P_0, P_1, and P_2.

The second step, `Draw_Wedge_Dihedral`, constructs a drawing of the wedge, which is the most complicated step of the drawing algorithm. This step must ensure that the middle edge of each diamond on the boundary is orthogonal to the axis of reflection.

Roughly speaking, the algorithm `Draw_Wedge_Dihedral` runs as follows: (i) Find all *special diamonds* of P_1 and P_2 that share fixed vertices or fixed edges, and draw them first using algorithm `Draw_Special_Diamonds`; (ii) choose the positions of all the fixed vertices of P_1 and P_2 that have not been drawn so far; (iii) subdivide the wedge in various ways to form "subwedges"; (iv) draw each of these subwedges using Algorithms `CYN` and `Symmetric_CYN` accordingly. For details, see [15].

The final step, Algorithm `Merge_Wedges_Dihedral` simply constructs a drawing for the whole graph by merging the drawing D of the wedge W. Clearly each step of Algorithm `Dihedral` takes linear time.

Cross-References

▸ Convex Graph Drawing

Recommended Reading

1. Abelson D, Hong S, Taylor DE (2007) Geometric automorphism groups of graphs. Discret Appl Math 155(17):2211–2226. Elsevier
2. Armstrong MA (1988) Groups and symmetry. Springer, New York

3. Buchheim C, Junger M (2003) Detecting symmetries by branch and cut. Math Progr (Ser B) 98:369–384
4. Chiba N, Yamanouchi T, Nishizeki T (1984) Linear algorithms for convex drawings of planar graphs. In: Adrian Bondy J, Murty USR (eds) Progress in graph theory. Academic, Toronto/Orlando, pp 153–173
5. Eades P, Hong S (2013) Detection and display of symmetries. In: Tamassia R (ed) Handbook of graph drawing and visualisation. Chapman and Hall/CRC
6. Eades P, Lin X (2000) Spring algorithms and symmetry. Theor Comput Sci 240(2):379–405
7. Fontet M (1976) Linear algorithms for testing isomorphism of planar graphs. In: Proceedings of third colloquium on automata, languages and programming, Edinburgh, pp 411–423
8. Hong S (2002) Drawing graphs symmetrically in three dimensions. In: Proceeding of graph drawing 2001, Vienna. Lecture notes in computer science, vol 2265. Springer, pp 189–204
9. Hong S, Eades P (2003) Drawing trees symmetrically in three dimensions. Algorithmica 36(2):153–178. Springer
10. Hong S, Eades P (2003) Symmetric layout of disconnected graphs. In: Algorithms and computation (Proceedings of ISAAC 2003, Kyoto). Lecture notes in computer science, vol 2906. Springer, Berlin/New York, pp 405–414
11. Hong S, Eades P (2004) Linkless symmetric drawings of series parallel digraphs. Comput Geom Theory Appl 29(3):191–222. Elsevier
12. Hong S, Eades P (2005) Drawing planar graphs symmetrically II: biconnected graphs. Algorithmica 42(2):159–197. Springer
13. Hong S, Eades P (2006) Drawing planar graphs symmetrically III: oneconnected graphs. Algorithmica 44(1):67–100. Springer
14. Hong S, Eades P, Lee S (2000) Drawing series parallel digraphs symmetrically. Comput Geom Theory Appl 17(3–4):165–188
15. Hong S, McKay B, Eades P (2006) A linear time algorithm for constructing maximally symmetric straight-line drawings of triconnected planar graphs. Discret Comput Geom 36(2):283–311. Springer
16. Hong S, Nagamochi H (2010) Linear time algorithm for symmetric convex drawings of planar graphs. Algorithmica 58(2):433–460. Springer
17. Hopcroft JE, Wong JK (1974) Linear time algorithm for isomorphism of planar graphs. In: Proceedings of ACM symposium on theory of computing, Seattle, pp 172–184
18. Lubiw A (1981) Some NP-complete problems similar to graph isomorphism. SIAM J Comput 10(1):11–21
19. Manning J, Atallah MJ (1988) Fast detection and display of symmetry in trees. Congr Numer 64:159–169
20. Manning J, Atallah MJ (1992) Fast detection and display of symmetry in outerplanar graphs. Discret Appl Math 39:13–35

Synchronizers, Spanners

Michael Elkin
Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Keywords

Low-stretch spanning subgraphs; Network synchronization

Years and Authors of Summarized Original Work

1985; Awerbuch

Problem Definition

Consider a communication network, modeled by an n-vertex undirected unweighted graph $G = (V, E)$, for some positive integer n. Each vertex of G hosts a processor of unlimited computational power; the vertices have unique identity numbers, and they communicate via the edges of G by sending messages of size $O(\log n)$ each.

In the *synchronous* setting, the communication occurs in discrete *rounds*, and a message sent in the beginning of a round R arrives at its destination before the round R ends. In the *asynchronous* setting, each vertex maintains its own clock, and clocks of distinct vertices may disagree. It is assumed that each message sent (in the asynchronous setting) arrives at its destination within a certain time τ after it was sent, but the value of τ is not known to the processors.

It is generally much easier to devise algorithms that apply to the synchronous setting (henceforth, synchronous algorithms) rather than to the asynchronous one (henceforth, asynchronous algorithms). In [1] Awerbuch initiated the study of simulation techniques that translate synchronous algorithms to asynchronous ones. These simulation techniques are called *synchronizers*.

To devise the first synchronizers, Awerbuch [1] constructed a certain graph partition which is of its own interest. In particular, Peleg and Schäffer noticed [8] that this graph partition induces a subgraph with certain interesting properties. They called this subgraph a *graph spanner*. Formally, for a positive integer parameter k, a k-spanner of a graph $G = (V, E)$ is a subgraph $G' = (V, H)$, $H \subseteq E$, such that for every edge $e = (v, u) \in E$, the distance between the vertices v and u in H, $\text{dist}'_G(v, u)$, is at most k.

Key Results

Awerbuch devised three basic synchronizers, called α, β, and γ. The synchronizer α is the simplest one; using it results in only a constant overhead in time, but in a very significant overhead in communication. Specifically, the latter overhead is linear in the number of edges of the underlying network. Unlike the synchronizer α, the synchronizer β requires a somewhat costly initialization stage. In addition, using it results in a significant time overhead (linear in the number of vertices n), but it is more communication efficient than α. Specifically, its communication overhead is linear in n.

Finally, the synchronizer γ represents a trade-off between the synchronizers α and β. Specifically, this synchronizer is parametrized by a positive integer parameter k. When k is small, then the synchronizer behaves similarly to the synchronizer α, and when k is large, it behaves similarly to the synchronizer β. A particularly important choice of k is $k = \log n$. At this point on the trade-off curve, the synchronizer γ has a logarithmic in n time overhead and a linear in n communication overhead. The synchronizer γ has, however, a quite costly initialization stage.

The main result of [1] concerning spanners is that for every $k = 1, 2, \ldots$, and every n-vertex unweighted undirected graph $G = (V, E)$, there exists an $O(k)$-spanner with $O(n^{1+1/k})$ edges. (This result was explicated by Peleg and Schäffer [8].)

Applications

Synchronizers are extensively used for constructing asynchronous algorithms. The first applications of synchronizers are constructing the *breadth-first-search tree* and computing the *maximum flow*. These applications were presented and analyzed by Awerbuch in [1]. Later synchronizers were used for maximum matching [10], for computing shortest paths [7], and for other problems.

Graph spanners were found useful for a variety of applications in distributed computing. In particular, some constructions of synchronizers employ graph spanners [1, 9]. In addition, spanners were used for routing [4] and for computing almost shortest paths in graphs [5].

Open Problems

Synchronizers with improved properties were devised by Awerbuch and Peleg [3] and Awerbuch et al. [2]. Both these synchronizers have polylogarithmic time and communication overheads. However, the synchronizers of Awerbuch and Peleg [3] require a large initialization time. (The latter is at least linear in n.) On the other hand, the synchronizers of [2] are randomized. A major open problem is to obtain *deterministic* synchronizers with polylogarithmic time and communication overheads and sublinear in n initialization time. In addition, the degrees of the logarithm in the polylogarithmic time and communication overheads in synchronizers of [2, 3] are quite large. Another important open problem is to construct synchronizers with improved parameters.

In the area of spanners, spanners that distort large distances to a significantly smaller extent than they distort small distances were constructed by Elkin and Peleg in [6]. These spanners fall short from achieving a *purely additive distortion*. Constructing spanners with a purely additive distortion is a major open problem.

Cross-References

▸ Sparse Graph Spanners

Recommended Reading

1. Awerbuch B (1985) Complexity of network synchronization. J ACM 4:804–823
2. Awerbuch B, Patt-Shamir B, Peleg D, Saks ME (1992) Adapting to asynchronous dynamic networks. In: Proceedings of the 24th annual ACM symposium on theory of computing, Victoria, 4–6 May 1992, pp 557–570
3. Awerbuch B, Peleg D (1990) Network synchronization with polylogarithmic overhead. In: Proceedings of the 31st IEEE symposium on foundations of computer science, Sankt Louis, 22–24 Oct 1990, pp 514–522
4. Awerbuch B, Peleg D (1992) Routing with polynomial communication-space tradeoff. SIAM J Discret Math 5:151–162
5. Elkin M (2001) Computing almost shortest paths. In: Proceedings of the 20th ACM symposium on principles of distributed computing, Newport, 26–29 Aug 2001, pp 53–62
6. Elkin M, Peleg D (2001) Spanner constructions for general graphs. In: Proceedings of the 33th ACM symposium on theory of computing, Heraklion, 6–8 July 2001, pp 173–182
7. Lakshmanan KB, Thulasiraman K, Comeau MA (1989) An efficient distributed protocol for finding shortest paths in networks with negative cycles. IEEE Trans Softw Eng 15:639–644
8. Peleg D, Schäffer A (1989) Graph spanners. J Graph Theory 13:99–116
9. Peleg D, Ullman JD (1989) An optimal synchronizer for the hypercube. SIAM J Comput 18:740–747
10. Schieber B, Moran S (1986) Slowing sequential algorithms for obtaining fast distributed and parallel algorithms: maximum matchings. In: Proceedings of 5th ACM symposium on principles of distributed computing, Calgary, 11–13 Aug 1986, pp 282–292

T

Table Compression

Raffaele Giancarlo[1] and Adam L. Buchsbaum[2]
[1]Department of Mathematics and Applications, University of Palermo, Palermo, Italy
[2]Madison, NJ, USA

Keywords

Compression and transmission of tables; Compression of multidimensional data; Compressive estimates of entropy; Storage

Years and Authors of Summarized Original Work

2003; Buchsbaum, Fowler, Giancarlo

Problem Definition

Table compression was introduced by Buchsbaum et al. [3] as a unique application of compression, based on several distinguishing characteristics. Tables are collections of fixed-length records and can grow to be terabytes in size. They are often generated by information systems and kept in data warehouses to facilitate ongoing operations. These data warehouses will typically manage many terabytes of data online, with significant capital and operational costs. In addition, the tables must be transmitted to different parts of an organization, incurring additional costs for transmission. Typical examples are tables of transaction activity, like phone calls and credit card usage, which are stored once but then shipped repeatedly to different parts of an organization: for fraud detection, billing, operations support, etc. The goals of table compression are to be fast, online, and effective: eventual compression ratios of 100:1 or better are desirable. Reductions in required storage and network bandwidth are obvious benefits.

Tables are different than general databases [3]. Tables are written once and read many times, while databases are subject to dynamic updates. Fields in table records are fixed in length, and records tend to be homogeneous; database records often contain intermixed fixed- and variable-length fields. Finally, the goals of compression differ. Database compression stresses index preservation, the ability to retrieve an arbitrary record, under compression [7]. Tables are typically not indexed at the level of individual records; rather, they are scanned in toto by downstream applications.

Consider each record in a table to be a row in a matrix. A naive method of table compression is to compress the string derived from scanning the table in row-major order. Buchsbaum et al. [3] observe experimentally that partitioning the table into contiguous intervals of columns and compressing each interval separately in this fashion can achieve significant compression improvement. The partition is generated by a one-time, off-line training procedure, and the resulting compression strategy is applied online

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

to the table. In their application, tables are generated continuously, so off-line training time can be ignored. They also observe heuristically that certain rearrangements of the columns prior to partitioning further improve compression by grouping dependent columns more closely. For example, in a table of addresses and phone numbers, the area code can often be predicted by the zip code when both are defined geographically. In information-theoretic terms, these dependencies are *contexts*, which can be used to predict parts of a table. Analogously to strings, where knowledge of context facilitates succinct codings of symbols, the existence of contexts in tables implies, in principle, the existence of a more succinct representation of the table.

Three main avenues of research have followed, one based on the notion of combinatorial dependency [3, 4], another on the notion of column dependency [17, 18], and the third on the notion of motifs and templates [1]. The first formalizes dependencies analogously to the joint entropy of random variables, while the second does so analogously to conditional entropy [8]. The third finds inspiration in classic paradigms of data compression such as textual substitution [19, 20]. These approaches to table compression have deep connections to universal similarity metrics [12], based on Kolmogorov complexity and compression, and their later uses in classification [6]. The first two approaches are instances of a new emerging paradigm for data compression, referred to as *boosting* [9], where data are reorganized to improve the performance of a given compressor. A software platform to facilitate the investigation of such invertible data transformations is described by Vo [16]

Notations

Let T be a table of $n = |T|$ columns and m rows. Let $T[i]$ denote the ith column of T. Given two tables T_1 and T_2, let T_1T_2 be the table formed by their juxtaposition. That is, $T = T_1T_2$ is defined so that $T[i] = T_1[i]$ for $1 \leq i \leq |T_1|$ and $T[i] = T_2[i - |T_1|]$ for $|T_1| < i \leq |T_1| + |T_2|$. We use the shorthand $T[i, j]$ to represent the projection $T[i]\cdots T[j]$ for any $j \geq i$. Also, givena sequence P of column indices, we denote by $T[P]$ the table obtained from T by projecting the columns with indices in P.

Combinatorial Dependency and Joint Entropy of Random Variables

Fix a compressor $\mathcal{C}$: e.g., gzip, based on LZ77 [19]; compress, based on LZ78 [20]; or bzip, based on Burrows-Wheeler [5]. Let $H_{\mathcal{C}}(T)$ be the size of the result of compressing table T as a string in row-major order using $\mathcal{C}$. Let $H_{\mathcal{C}}(T_1, T_2) = H_{\mathcal{C}}(T_1, T_2).H_{\mathcal{C}}(\cdot)$ is thus a cost function defined on the ordered power set of columns. Two tables T_1 and T_2, which might be projections of columns from a common table T, are *combinatorially dependent* if $H_{\mathcal{C}}(T_1, T_2) < H_{\mathcal{C}}(T_1) + H_{\mathcal{C}}(T_2)$ – if compressing them together is better than compressing them separately – and *combinatorially independent* otherwise. Buchsbaum et al. [3] show that combinatorial dependency is a compressive estimate of statistical dependency when formalized by the joint entropy of two random variables, i.e., the statistical relatedness of two objects is measured by the gain realized by compressing them together rather than separately. Indeed, combinatorial dependency becomes statistical dependency when $H_{\mathcal{C}}$ is replaced by the joint entropy function [8]. Analogous notions starting from Kolmogorov complexity are derived by Li et al. [12] and used for classification and clustering [6]. Figure 1 exemplifies why rearranging and partitioning columns may improve compression.

9	0	8	2	7	3
9	0	8	3	7	5
9	0	8	5	7	6
9	0	8	2	7	5

Table Compression, Fig. 1 The first three columns of the table, taken in row-major order, form a repetitive string that can be very easily compressed. Therefore, it may be advantageous to compress these columns separately. If the fifth column is swapped with the fourth, we get an even longer repetitive string that, again, can be compressed separately from the other two columns

Problem 1 Find a partition $\mathcal{P}$ of T into sets of contiguous columns that minimizes $\sum_{Y \in \mathcal{P}} H_C(Y)$ over all such partitions.

Problem 2 Find a partition $\mathcal{P}$ of T that minimizes $\sum_{Y \in \mathcal{P}} H_C(Y)$ over all partitions.

The difference between Problems 1 and 2 is that the latter does not require the parts of $\mathcal{P}$ to be sets of contiguous columns.

Column Dependency and Conditional Entropy of Random Variables

Definition 1 For any table T, a *dependency relation* is a pair (P, c) in which P is a sequence of distinct column indices (possibly empty) and $c \notin P$ is another column index. If the length of P is less than or equal to k, then (P, c) is called a k-relation. P is the *predictor sequence* and c is the *predictee*.

Definition 2 Given a dependency relation (P, c), the *dependency transform* $dt_P(c)$ of c is formed by permuting column $T[c]$ based on the permutation induced by a stable sort of the rows of P.

Definition 3 A collection D of dependency relations for table T is said to be a k-transform if and only if (a) each column of T appears exactly once as a predictee in some dependency relation (P, c), (b) the dependency hypergraph $G(D)$ is acyclic, and (c) each dependency relation (P, c) is a k-relation.

Let $\omega(P, c)$ be the cost of the dependency relation (P, c), and let $\delta(m)$ be an upper bound on the cost of computing $\omega(P, c)$. Intuitively, $\omega(P, c)$ gives an estimate of how well a rearrangement of column c will compress, using the rows of P as contexts for its symbols. We will provide an example after the formal definitions.

Problem 3 Find a k-transform D of minimum cost $\omega(D) = \sum_{(P,c) \in D} \omega(P, c)$.

Definition 1 extends to columns the notion of context that is well known for strings. Definition 3 defines a microtransformation that reorganizes the column symbols by grouping together those that have similar contexts. The context of a column symbol is given by the corresponding row in $T[P]$. The fundamental ideas here are the same as in the Burrows and Wheeler transform [5]. Finally, Problem 3 asks for an optimal strategy to reorganize the data prior to compression. The cost function ω provides an estimate of how well c can be compressed using the knowledge of $T[P]$.

Vo and Vo [18] connect these ideas to the conditional entropy of random variables. Let S be a sequence, $\mathcal{A}(S)$ its distinct elements, and f_a the frequency of each element a. The *zeroth-order empirical entropy* of S [15] is

$$H_0(S) = -\frac{1}{|S|} \sum_{\alpha \in \mathcal{A}(S)} f_a \lg \frac{f_a}{|S|},$$

and the *modified zeroth-order empirical entropy* [15] is

$$H_0^*(S) = \begin{cases} 0 & \text{if} |S| = 0, \\ (1 + \lg |S|)/|S| & \text{if} |S| \neq 0 \text{ and } H_0(S) = 0, \\ H_0(S) & \text{otherwise.} \end{cases}$$

For a dependency relation (P, c) with nonempty P, the *modified conditional empirical entropy* of c given P is then defined as

$$H_P^*(c) = \frac{1}{m} \sum_{\rho \in \mathcal{A}(T[P])} |\rho_c| H_0^*(\rho_c),$$

where ρ_c is the string formed by catenating the symbols in c corresponding to positions of ρ in $T[P]$ [15]. A possible choice of $\omega(P, c)$ is given by $H_P^*(c)$. Vo and Vo also develop another notion of entropy, called *run length entropy*, to approximate more effectively the compressibility of low-entropy columns and define another cost function ω accordingly.

Key Results

Combinatorial Dependency

Problem 1 admits a polynomial-time algorithm, based on dynamic programming. Using the definition of combinatorial dependency, one can show:

Theorem 1 ([3]) *Let $E[i]$ be the cost of an optimal, contiguous partition of $T[1, i]$. $E[n]$ is thus the cost of a solution to Problem 1. Define $E[0] = 0$; then, for $1 \leq i \leq n$,*

$$E[i] = \min_{0 \leq j < i} E[j] + H_{\mathcal{C}}(T_{j+1}, \ldots, T_i). \quad (1)$$

The actual partition with cost $E[n]$ can be maintained by standard backtracking.

The only known algorithmic solution to Problem 2 is the trivial one based on enumerating all possible feasible solutions to choose an optimal one. Some efficient heuristics based on asymmetric TSP, however, have been devised and tested experimentally [4]. Define a weighted, complete, directed graph, $G(T)$, with a vertex T_i for each column $T[i] \in T$; the *weight* of edge $\{T_i, T_j\}$ is $w(T_i, T_j) = \min(H_{\mathcal{C}}(T_i, T_j), H_{\mathcal{C}}(T_i) + H_{\mathcal{C}}(T_j))$. One then generates a set of tours of various weights by iteratively applying standard optimizations (e.g., 3-opt, 4-opt). Each tour induces an ordering of the columns, which are then optimally partitioned using the dynamic program (1).

Buchsbaum et al. [4] also provide a general framework for studying the computational complexity of several variations of table compression problems based on notions analogous to combinatorial dependence, and they give some initial MAX-SNP-hardness results. Particularly relevant is the set of abstract problems in which one is required to find an optimal arrangement of a set of strings to be compressed, which establishes a nontrivial connection between table compression and the classical shortest common superstring problem [2]. Giancarlo et al. [11] connect table compression to the Burrows and Wheeler transform [5] by deriving the latter as a solution to an analog of Problem 2.

Column Dependency

Theorem 2 ([17, 18]) *For $k \geq 2$, Problem 3 is NP-hard.*

Theorem 3 ([17, 18]) *An optimum 1-transform for a table T can be found in $O(n^2\delta(m))$ time.*

Theorem 4 ([17, 18]) *A 2-transform can be computed in $O(n^2\delta(m))$ time.*

Theorem 5 ([18]) *For any dependency relation (P, c) and some constant ϵ, $|\mathcal{C}(dt_P(c))| \leq 5m H_p^*(c) + \epsilon$.*

Motifs

Apostolico et al. [1] propose improved versions of Table Compression based on Motifs, i.e., regular expressions characterizing a set of templates based on which the rows of a table are compressed by textual substitution. They also discuss applications of the technique in Computational Biology.

Applications

Storage and transmission of alphanumeric tables. Moreover, the Citing Articles of the papers in [1, 3, 4, 17, 18] in Google Scholar provide a full range of applications and related work.

Open Problems

All the techniques discussed use the general paradigms of context-dependent data rearrangement for compression boosting. It remains open to apply these paradigms to other domains, e.g., XML data [13, 14], where high-level structures

can be exploited, and to domains where pertinent structures are not known a priori.

Experimental Results

Buchsbaum et al. [3] showed that optimal partitioning alone (no column rearrangement) yielded about 55 % better compression compared to gzip on telephone usage data, with small training sets. Buchsbaum et al. [4] experimentally supported the hypothesis that good TSP heuristics can effectively reorder the columns, yielding additional improvements of 5–20 % relative to partitioning alone. They extended the data sets used to include other tables from the telecom domain as well as biological data. Vo and Vo [17,18] showed further 10–35 % improvement over these combinatorial dependency methods on the same data sets.

Data Sets

Some of the data sets used for experimentation are public [4].

URL to Code

The pzip package, based on combinatorial dependency, is available at http://www.research.att.com/~gsf/pzip/pzip.html. The Vcodex package, related to invertible transforms, is available at http://www.research.att.com/~gsf/download/ref/vcodex/vcodex.html. Although for the time being Vcodex does not include procedures to compress tabular data, it is a useful toolkit for their development.

Cross-References

- Binary Decision Graph
- Burrows-Wheeler Transform
- Compressed Tree Representations
- Compressing and Indexing Structured Text
- Lempel-Ziv Compression

Recommended Reading

1. Apostolico A, Cunian F, Kaul V (2008) Table compression by record intersection. In: Proceedings of the IEEE data compression conference (DCC), Snowbird, pp 13–22
2. Blum A, Li M, Tromp J, Yannakakis M (1994) Linear approximation of shortest superstrings. J ACM 41:630–647
3. Buchsbaum AL, Caldwell DF, Church KW, Fowler GS, Muthukrishnan S (2000) Engineering the compression of massive tables: an experimental approach. In: Proceedings of the 11th ACM-SIAM symposium on discrete algorithms, San Francisco, pp 175–184
4. Buchsbaum AL, Fowler GS, Giancarlo R (2003) Improving table compression with combinatorial optimization. J ACM 50:825–851
5. Burrows M, Wheeler D (1994) A block sorting lossless data compression algorithm. Technical report 124, Digital Equipment Corporation
6. Cilibrasi R, Vitanyi PMB (2005) Clustering by compression. IEEE Trans Inf Theory 51:1523–1545
7. Cormack G (1985) Data compression in a data base system. Commun ACM 28:1336–1350
8. Cover TM, Thomas JA (1990) Elements of information theory. Wiley Interscience, New York
9. Ferragina P, Giancarlo R, Manzini G, Sciortino M (2005) Boosting textual compression in optimal linear time. J ACM 52:688–713
10. Ferragina P, Luccio F, Manzini G, Muthukrishnan S (2005) Structuring labeled trees for optimal succinctness, and beyond. In: Proceedings of the 45th annual IEEE symposium on foundations of computer science, Pittsburgh, pp 198–207
11. Giancarlo R, Sciortino M, Restivo A (2007) From first principles to the Burrows and Wheeler transform and beyond, via combinatorial optimization. Theor Comput Sci 387:236–248
12. Li M, Chen X, Li X, Ma B, Vitanyi PMB (2004) The similarity metric. IEEE Trans Inf Theory 50:3250–3264
13. Liefke H, Suciu D (2000) XMILL: an efficient compressor for XML data. In: Proceedings of the 2000 ACM SIGMOD international conference on management of data, Dallas. ACM, New York, pp 153–164
14. Lifshits Y, Mozes S, Weimann O, Ziv-Ukelson M (2009) Speeding up HMM decoding and training by exploiting sequence repetitions. Algorithmica 54:379–399
15. Manzini G (2001) An analysis of the Burrows-Wheeler transform. J ACM 48:407–430
16. Vo K-P (2006) Compression as data transformation. In: DCC: data compression conference, Snowbird. IEEE Computer Society TCC, Washington DC, p 403
17. Vo BD, Vo K-P (2004) Using column dependency to compress tables. In: DCC: data compression conference, Snowbird. IEEE Computer Society TCC, Washington DC, pp 92–101

18. Vo BD, Vo K-P (2007) Compressing table data with column dependency. Theor Comput Sci 387:273–283
19. Ziv J, Lempel A (1977) A universal algorithm for sequential data compression. IEEE Trans Inf Theory 23:337–343
20. Ziv J, Lempel A (1978) Compression of individual sequences via variable length coding. IEEE Trans Inf Theory 24:530–536

Tail Bounds for Occupancy Problems

Paul (Pavlos) Spirakis
Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece
Computer Science, University of Liverpool, Liverpool, UK
Computer Technology Institute (CTI), Patras, Greece

Keywords

Balls and bins

Years and Authors of Summarized Original Work

1995; Kamath, Motwani, Palem, Spirakis

Problem Definition

Consider a *random allocation* of m balls to n bins where each ball is placed in a bin chosen uniformly and independently. The properties of the resulting distribution of balls among bins have been the subject of intensive study in the probability and statistics literature [3, 4]. In computer science, this process arises naturally in randomized algorithms and probabilistic analysis. Of particular interest is the *occupancy problem* where the random variable under consideration is the number of empty bins.

In this entry a series of bounds are presented (reminiscent of the Chernoff bound for binomial distributions) on the tail of the distribution of the number of empty bins; the tail bounds are successively tighter, but each new bound has a more complex closed form. Such strong bounds do not seem to have appeared in the earlier literature.

Key Results

The following notation in presenting sharp bounds on the tails of distributions. The notation $F \sim G$ will denote that $F = (1 + o(1))G$; further, $F \asymp G$ will denote that $\ln F \sim \ln G$. The proof that $f \asymp g$, is used for the purposes of later claiming that $2^f \asymp 2^g$. These asymptotic equalities will be treated like actual equalities and it will be clear that the results claimed are unaffected by this "approximation".

Consider now the probabilistic experiment of throwing m balls, independently and uniformly, into n bins.

Definition 1 Let Z be the number of empty bins when m balls are placed randomly into n bins, and define $r = m/n$. Define the function $H(m, n, z)$ as the probability that $Z = z$. The expectation of Z is given by

$$\mu = \mathbf{E}[Z] = n\left(1 - \frac{1}{n}\right)^m \sim n\,\mathrm{e}^{-r}.$$

The following three theorems provide the bounds on the tail of the distribution of the random variable Z. The proof of the first bound is based on a martingale argument.

Theorem 1 (Occupancy Bound 1) *For any* $\theta > 0$,

$$\boldsymbol{P}\left[|Z - \mu| \geq \theta\mu\right] \leq 2\exp\left(-\frac{\theta^2\mu^2(n - \frac{1}{2})}{n^2 - \mu^2}\right).$$

Remark that for large r this bound is asymptotically equal to

$$2\exp\left(-\frac{\theta^2\,\mathrm{e}^{-2r}n}{1 - \mathrm{e}^{-2r}}\right).$$

The reader may wish to compare this with the following heuristic estimate of the tail probability assuming that the distribution of Z is well approximated by the approximating normal distribution also far out in the tails [3, 4].

$$\boldsymbol{P}\left[|Z-\mu| \geq \theta\mu\right] \leq 2\exp\left(-\frac{\theta^2 \mathrm{e}^{-r} n}{2\left(1-(1+r)\mathrm{e}^{-r}\right)}\right).$$

The next two bounds are in terms of point probabilities rather than tail probabilities (as was the case in the Binomial Bound), but the unimodality of the distribution implies that the two differ by at most a small (linear) factor. These more general bounds on the point probability are essential for the application to the satisfiability problem. The next result is obtained via a generalization of the Binomial Bound to the case of dependent Bernoulli trials.

Theorem 2 (Occupancy Bound 2) *For* $\theta > -1$,

$$H(m,n,(1+\theta)\mu) \leq \exp\left(-\left((1+\theta)\ln[1+\theta]-\theta\right)\mu\right).$$

In particular, for $-1 \leq \theta < 0$,

$$H(m,n,(1+\theta)\mu) \leq \exp\left(-\frac{\theta^2\mu}{2}\right).$$

The last result is proved using ideas from large deviations theory (Weiss A (1993) Personal Communication).

Theorem 3 (Occupancy Bound 3) *For* $|z-\mu| = \Omega(n)$,

$$H(m,n,z) \asymp \exp\left(\left[-n\left(\int_0^{1-\frac{z}{n}} \ln\left[\frac{k-x}{1-x}\right] \mathrm{d}x - r\ln k\right)\right]\right)$$

where k *is defined implicitly by the equation* $z = n(1-k(1-\mathrm{e}^{-r/k}))$.

Applications

Random allocations of balls to bins is a basic model that arises naturally in many areas in computer science involving choice between a number of resources, such as communication links in a network of processors, actuator devices in a wireless sensor network, processing units in a multi-processor parallel machine etc. For such situations, randomization can be used to "spread" the load evenly among the resources, an approach particularly useful in a parallel or distributed environment where resource utilization decisions have to be made locally at a large number of sites without reference to the global impact of these decisions. In the process of analyzing the performance of such algorithms, of particular interest is the occupancy problem where the random variable under consideration is the number of empty bins (i.e., machines with no jobs, routes with no load, etc.). The properties of the resulting distribution of balls among bins and the corresponding tails bounds may help in order to analyze the performance of such algorithms.

Cross-References

- Approximation Schemes for Bin Packing
- Bin Packing

Recommended Reading

1. Kamath A, Motwani R, Spirakis P, Palem K (1995) Tail bounds for occupancy and the satisfiability threshold conjecture. J Random Struct Algorithms 7(1):59–80
2. Janson S (1994) Large deviation inequalities for sums of indicator variables. Technical report No. 34, Department of Mathematics, Uppsala University
3. Johnson NL, Kotz S (1977) Urn models and their applications. Wiley, New York
4. Kolchin VF, Sevastyanov BA, Chistyakov VP (1978) Random allocations. Wiley, New York
5. Motwani R, Raghavan P (1995) Randomized algorithms. Cambridge University Press, New York
6. Shwartz A, Weiss A (1994) Large deviations for performance analysis. Chapman-Hall, Boca Raton

Technology Mapping

Kurt Keutzer[1] and Kaushik Ravindran[2]
[1]Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA
[2]National Instruments, Berkeley, CA, USA

Keywords

Library-based technology mapping; Technology-dependent optimization

Years and Authors of Summarized Original Work

1987; Keutzer

Problem Definition

Technology mapping is the problem of implementing a sequential circuit using the gates of a particular technology library. It is an integral component of any automated VLSI circuit design flow. In the prototypical chip design flow, combinational logic gates and sequential memory elements are composed to form sequential circuits. These circuits are subject to various logic optimizations to minimize area, delay, power, and other performance metrics. The resulting optimized circuits still consist of primitive logic functions such as AND and OR gates. The next step is to efficiently realize these circuits in a specific VLSI technology using a library of gates available from the semiconductor vendor. Such a library would typically consist of gates of varying sizes and speeds for primitive logic functions (AND and OR) and more complex functions (exclusive-OR, multiplexer). However, a naïve translation of generic logic elements to gates in the library will fall short of realistic performance goals. The challenge is to construct a mapping that maximally utilizes the gates in the library to implement the logic function of the circuit and achieve some performance goal, for example, minimum area with the critical path delay less than a target value. This is accomplished by *technology mapping*. For the sake of simplicity, in the following discussion, it is presumed that the sequential memory elements are stripped from the digital circuit and mapped directly into memory elements of the particular technology. Then, only Boolean circuits composed of combinational logic gates remain to be mapped. Further, each remaining Boolean circuit is necessarily a directed acyclic graph (DAG).

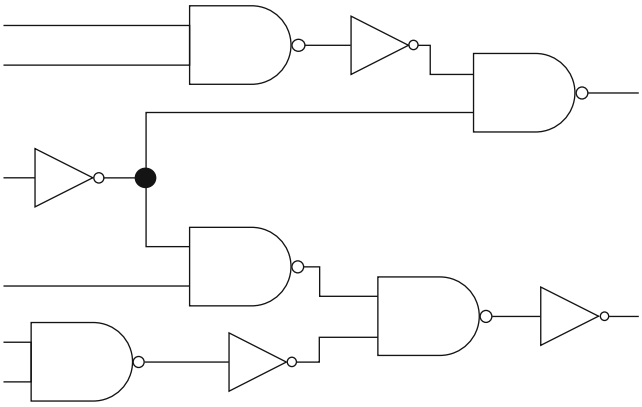
The technology mapping problem can be restated in a more general graph-theoretic setting: *find a minimum cost covering of the subject graph (Boolean circuit) by choosing from the collection of pattern graphs (gates) available in a library.* The inputs to the problem are:

(a) *Subject graph:* This is a directed acyclic graph representation of a Boolean circuit expressed using a set of primitive functions (e.g., 2-input NAND gates and inverters). An example subject graph is shown in Fig. 1.
(b) *Library of pattern graphs:* This is a collection of gates available in the technology library. The pattern graphs are also DAGs expressed using the same primitive functions used to construct the subject graph. Additionally, each gate is annotated with a number of values for different cost functions, such as area, delay, and power. An example library and associated cost model is shown in Fig. 2.

A *valid cover* is a network of pattern graphs implementing the function of the subject graph such that (a) every vertex (i.e., gate) of the subject graph is contained in some pattern graph and (b) each input required by a pattern graph is actually an output of some other pattern graph (i.e., the inputs of a gate must exist as outputs of other gates). Technology mapping can then be viewed as an optimization problem to find a valid cover of minimum cost of the subject graph.

Key Results

To be viable in a realistic design flow, an algorithm for minimum cost graph-covering for technology mapping should ideally possess



Technology Mapping, Fig. 1 Subject graph (DAG) of a Boolean circuit expressed using NAND2 and INVERTER gates

Gate	Cost	Pattern Graph
INVERTER	2	
NAND2	3	
NAND3	4	
AND-OR-INVERT-21	4	
AND-OR-INVERT-22	5	

Technology Mapping, Fig. 2 Library of pattern graphs (composed of NAND2 and INVERTER gates) and associated costs

the following characteristics: (a) the algorithm should be easily adaptable to diverse libraries and cost models; if the library is expanded or replaced, the algorithm must be able to utilize the new gates effectively; (b) it should allow detailed cost models to accurately represent the performance of the gates in the library; and (c) it should be fast and robust on large subject graph instances and large libraries. One technique for solving the minimum cost graph-covering problem is to formulate it as a binate-covering problem, which is a specialized integer linear program [10]. However, binate-covering for a DAG is NP-Hard for any set of primitive functions and is typically unwieldy on large circuits. The DAGON algorithm suggested solving the technology mapping problem through DAG-covering and advanced an alternate approach for DAG-covering based on a *tree-covering* approximation that produced near-optimal solutions for practical circuits and was very fast even for large circuits and large libraries [7].

DAGON was inspired by prevalent techniques for pattern matching employed in the domain of code generation for programming language compilers [1]. The fundamental concept was to partition the subject graph (DAG) into a forest of trees and solve the minimum cost covering problem independently for each tree. The approach was motivated by the existence of efficient dynamic programming algorithms for optimum tree-covering [2]. The three salient components of the DAGON algorithm are (a) subject graph partitioning, (b) pattern matching, and (c) covering.

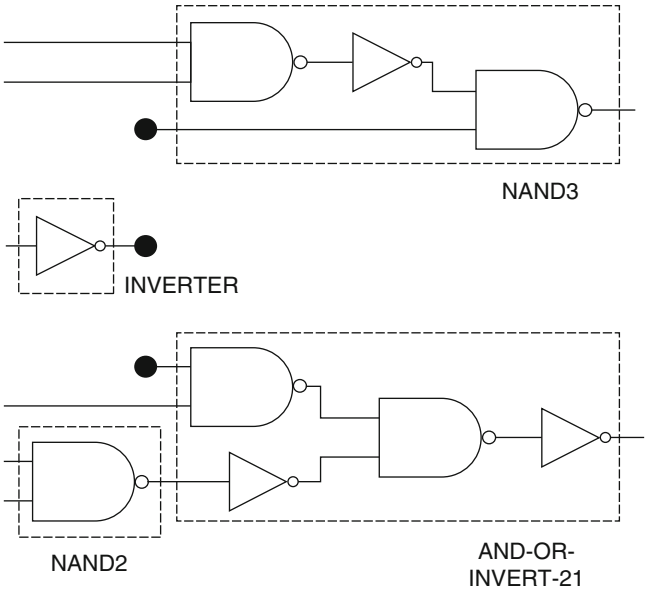
(a) *Subject graph partitioning*: To apply the tree-covering approximation the subject graph is first partitioned into a forest of trees. One approach is to break the graph at each vertex which has an out-degree greater than 1 (multiple fan-out point). The root of each tree is the primary output of the corresponding sub-circuit and the leaves are the primary inputs. Other heuristic partitions of the subject graph that consider duplication of vertices can also be applied to improve the quality of the final cover. Alternate subject graph partitions can also be derived starting from different decompositions of the original Boolean circuit in terms of the primitive functions.

(b) *Pattern matching*: The optimum covering of a tree is determined by generating the complete set of matches for each vertex in the tree (i.e., the set of pattern graphs which are candidates for covering a particular vertex) and then selecting the optimum match from among the candidates. An efficient approach for structural pattern matching is to reduce the tree matching problem to a *string matching* problem [2]. Fast string matching algorithms, such as the Aho-Corasick and the Knuth-Morris-Pratt algorithms, can then be used to find all strings (pattern graphs) which match a given vertex in the subject graph in time proportional to the length of the longest string in the set of pattern graphs. Alternatively, Boolean matching techniques can be used to find matches based on logic functions [5]. Boolean matching is slower than structural string matching, but it can compute matches independent of the actual local decompositions and under different input permutations.

(c) *Covering*: The final step is to generate a valid cover of the subject tree using the pattern graph matches computed at each vertex. Consider the problem of finding a valid cover of minimum area for the subject tree. Every pattern graph in the library has an associated area and the area of a valid cover is the sum of the area of the pattern graphs in the cover. The key property that makes minimum area tree-covering efficient is this: *the minimum area cover of a tree rooted at some vertex v can be computed using only the minimum area covers of vertices below v*. It follows that for every pattern graph that matches the tree rooted at vertex v, the area of the minimum cover containing that match equals the sum of the area of the corresponding match at v and the sum of the areas of the optimal covers of the vertices which are inputs to that match. This property enables a dynamic programming algorithm to compute the minimum area cover of the tree rooted at each vertex of the subject tree. The base case is the minimum area cover of a leaf (primary input) of the subject tree. The area of a match at a leaf is set to 0. A recursive formulation of this dynamic programming concept is summarized in the Algorithm *minimum_area_tree_cover* shown below. As an example, the minimum area cover displayed in Fig. 3 is a result of applying this algorithm to the tree partitions of the subject graph from Fig. 1 using the library from Fig. 2.

Given a vertex v in the subject tree, let $M(v)$ denote the set of candidate matches from the library of pattern graphs for the sub-tree rooted at v.

In this algorithm, each vertex in the tree is visited exactly once. Hence, the complexity of the algorithm is proportional to the number of vertices in the subject tree times the maximum number of pattern matches at any vertex. The maximum number of matches is a function of the pattern graph library and is independent of the subject tree size. As a result, the complexity of computing the minimum cost valid cover of a tree is linear in the size of the subject tree, and the memory requirements are also linear in the size of the subject tree. The algorithm computes the optimum cover when the subject graph is a tree. In the general case of the subject graph being a DAG, empirical results have shown that the tree-covering approximation yields industrial-quality results achieving aggressive area and timing requirements on large real circuit design problems [6, 12].



Technology Mapping, Fig. 3 Result of a minimum area tree-covering of the subject graph in Fig. 1 using the library of pattern graphs in Fig. 2

Applications

Technology mapping is the key link between technology-independent logic synthesis and technology-dependent physical design of VLSI circuits. This motivates the need for efficient and robust algorithms to implement large Boolean circuits in a technology library. Early algorithms for technology mapping were founded on rule-based local transformations [4]. DAGON was the first in advancing an algorithmic foundation in terms of graph transformations that was practicable in the inner loop of iterative procedures in the VLSI design flow [7]. From a theoretical standpoint, the graph-covering formulation provided a formal description of the problem and specified optimality criteria for evaluating solutions. The algorithm was naturally adaptable to diverse libraries and cost models, and was relatively easy to implement and extend. The concept of partitioning the subject graph into trees and covering the trees optimally was effective for varied optimization objectives such as area, delay, and power. The DAGON approach has been incorporated in academic (SIS from the University of California at Berkeley [11]) and industrial (Synopsys™ Design Compiler) tool offerings for logic synthesis and optimization.

The graph-covering formulation has also served as a starting point for advancements in algorithms for technology mapping over the last decade. Decisions related to logic decomposition were integrated in the graph-covering algorithm, which in turn enabled technology independent logic optimizations in the technology mapping phase [9, 14]. Similarly, heuristics were proposed to impose placement constraints and make technology mapping more aware of the physical design and layout of the final circuit [8]. To combat the problem of high power dissipation in modern sub-micron technologies, the graph algorithms were enhanced to minimize power under area and delay constraints [13]. Specializations of these graph algorithms for technology mapping have found successful application in design flows for Field Programmable Gate Array (FPGA) technologies [3, 15]. We recommend the following works for a comprehensive treatment of algorithms for technology mapping and a

```
Algorithm minimum_area_tree_cover (Vertex v) {
// the algorithm minimum_area_tree_cover finds an
optimal cover of the tree rooted at Vertex v
// the algorithm computes best_match(v) and
area_of_best_match(v), which denote the best
pattern graph match at v and the associated area of
the optimal cover of the tree rooted at v, respectively
// check if v is a leaf of the tree
if (v is a leaf) {
area_of_best_match(v) = 0;
best_match(v) = leaf;
return;
}
// compute optimal cover for each input of v
foreach (input of Vertex v) {
minimum_area_tree_cover(input);
}
// each tree rooted at each input of v is now
annotated with its optimal cover
//find the optimal cover of the tree rooted at Vertex v
area_of_best_match(v) = INFINITY;
best_match(v) = NULL;
foreach (Match m in the set of matches M(v)) {
// compute the area of match m at Vertex v
// area_of_match(v,m) denotes the area of the cover
when Match m is selected for v
area_of_match(v,m) = area(m);
foreach input pin vi of match m {
area_of_match (v,m) = area_of_match(v,m) +
area_of_best_match(vi)
}
// update best pattern graph match and associated
area of the optimal cover at Vertex v
if (area_of_match(v,m) < area_of_best_match(v)) {
area_of_best_match(v) = area_of_match(v,m);
best_match(v) = m;
}
}
}
```

survey of new developments and challenges in the design of modern VLSI circuits: [5, 6, 12].

Open Problems

The enduring problem with DAGON-related technology mappers is handling non-tree pattern graphs that arise from modeling circuit elements such as multiplexors, Exclusive-Ors, or memory-elements (e.g., flip-flops) with associated logic (e.g., scan logic). On the other hand, approaches that do not use the tree-covering formulation face challenges in easily representing diverse technology libraries and in matching the subject graph in a computationally efficient manner.

Recommended Reading

1. Aho A, Sethi R, Ullman J (1986) Compilers: principles, techniques and tools. Addison Wesley, Boston, pp 557–584
2. Aho A, Johnson S (1976) Optimal code generation for expression trees. J ACM 23(July):488–501
3. Cong J, Ding Y (1994) FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. IEEE Trans Comput-Aided Des Integr Circuits Syst **13**(1): 1–12
4. Darringer JA, Brand D, Gerbi JV, Joyner WH, Trevillyan LH (1981) LSS: logic synthesis through local transformations. IBM J Res Dev 25:272–280
5. De Micheli G (1994) Synthesis and optimization of digital circuits, 1st edn. McGraw-Hill, New York, pp 504–533
6. Devadas S, Ghosh A, Keutzer K (1994) Logic synthesis. McGraw Hill, New York, pp 185–200
7. Keutzer K (1987) DAGON: technology binding and local optimizations by DAG matching. In: Proceedings of the 24th design automation conference, vol 28(1), Miami Beach, pp 341–347, June 1987
8. Kutzschebauch T, Stok L (2001) Congestion aware layout driven logic synthesis. In: Proceedings of the IEEE/ACM international conference on computer-aided design, Santa Clara, pp 216–223
9. Lehman E, Watanabe Y, Grodstein J, Harkness H (1997) Logic decomposition during technology mapping. IEEE Trans Comput-Aided Des Integr Circuits Syst 16(8):813–834
10. Rudell R (1989) Logic synthesis for VLSI design. Ph.D. thesis, University of California at Berkeley, ERL Memo 89/49, April 1989
11. Sentovich EM, Singh KJ, Moon C, Savoj H, Brayton RK, Sangiovanni-Vincentelli A (1992) Sequential circuit design using synthesis and optimization. In: Proceedings of the IEEE international conference on computer design: VLSI in computers & processors (ICCD), Cambridge, pp 328–333, Oct 1992
12. Stok L, Tiwari V (2002) Technology mapping. In: Hassoun S, Sasou T (eds) Logic synthesis and verification. Kluwer international series in engineering and computer science series. Kluwer, Norwell, pp 115–139
13. Tiwari V, Ashar P, Malik S (1996) Technology mapping for low power in logic synthesis. Integr VLSI J 20(3):243–268
14. Clarke EM, McMillan KL, Zhao X, Fujita M, Yang J (1993) Spectral transforms for large Boolean functions with applications to technology mapping. In: 30th conference on design automation, Dallas, 1993. IEEE, pp 54–60
15. Francis R, Rose J, Vranesic Z (1991) Chortle-crf: fast technology mapping for lookup table-based FPGAs. In: Proceedings of the 28th ACM/IEEE design automation conference, San Francisco, 1991. ACM, pp 227–233

Teleportation of Quantum States

Anurag Anshu[1], Vamsi Krishna Devabathini[1], Rahul Jain[2], and Priyanka Mukhopadhyay[1]
[1]Center for Quantum Technologies, National University of Singapore, Singapore, Singapore
[2]Department of Computer Science, Center for Quantum Technologies, National University of Singapore, Singapore, Singapore

Keywords

Information theory; Quantum communication; Teleportation

Synonyms

Quantum teleportation; Teleportation

Years and Authors of Summarized Original Work

1993; Bennett, Brassard, Crepeau, Jozsa, Peres, Wootters

Problem Definition

Suppose there are two spatially separated parties Alice and Bob and Alice wants to send a quantum state consisting of n quantum bits (qubits) ρ to Bob. Since classical communication is much more reliable, and possibly cheaper, than quantum communication, it is desirable that this task be achieved by communicating just classical bits. Such a procedure is referred to as *teleportation*.

Unfortunately, it is easy to argue that this is in fact not possible if arbitrary quantum states need to be communicated faithfully. However, Bennett, Brassard, Crepeau, Jozsa, Peres, and Wootters [8] presented a nice solution to it by modifying the assumptions about the resources that are available to Alice and Bob.

Key Results

Let $\{|0\rangle, |1\rangle\}$ be the standard basis for the state space of one quantum bit (which is equal to $\mathbb{C}^2$). For simplicity of notation $|0\rangle \otimes |0\rangle$ are represented as $|0\rangle|0\rangle$ or simply $|00\rangle$. An EPR pair is a special two-qubit quantum state defined as $|\psi\rangle \stackrel{\Delta}{=} \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

Alice and Bob are said to share an EPR pair if each holds one qubit of the pair. In this article a standard notation is followed in which classical bits are called "cbits" and shared EPR pairs are called "ebits." Bennett et al. showed the following:

Theorem 1 *Teleportation of an arbitrary n-qubit state can be achieved with $2n$ cbits and n ebits.*

These shared EPR pairs are referred to as *prior entanglement* to the protocol since they are shared at the beginning of the protocol (before Alice gets her input state) and are independent of Alice's input state. This solution is a good compromise since it is conceivable that Alice and Bob share several EPR pairs at the beginning, when they are possibly together, in which case they do not require a quantum channel. Later they can use these EPR pairs to transfer several quantum states when they are spatially separated.

Let us now see how Bennett et al. [8] achieve teleportation. Let us first note that in order to show Theorem 1, it is enough to show that a single qubit, which is possibly a part of a larger state ρ, can be teleported, while preserving its entanglement with the rest of the qubits of ρ, using 2 cbits and 1 ebit. Let us also note that the larger state ρ can now be assumed to be a pure state without loss of generality.

Theorem 2 *Let $|\phi\rangle_{AB} = a_0|\phi_0\rangle_{AB}|0\rangle_A + a_1|\phi_1\rangle_{AB}|1\rangle_A$, where a_0, a_1 are complex numbers with $|a_0|^2 + |a_1|^2 = 1$. Subscripts A, B (representing Alice and Bob, respectively) on qubits signify their owner.*

It is possible for Alice to send two classical bits to Bob such that at the end of the protocol the final state is $a_0|\phi_0\rangle_{AB}|0\rangle_B + a_1|\phi_1\rangle_{AB}|1\rangle_B$.

Proof For simplicity of notation, let us assume below that $|\phi_0\rangle_{AB}$ and $|\phi_1\rangle_{AB}$ do not exist. The proof is easily modified when they do exist by tagging them along. Let an EPR pair $|\psi\rangle_{AB} = \frac{1}{\sqrt{2}}(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B)$ be shared between Alice and Bob. Let us refer to the qubit under concern that needs to be teleported as the input qubit.

The combined starting state of all the qubits is

$$\begin{aligned}|\theta_0\rangle_{AB} &= |\phi\rangle_{AB}|\psi\rangle_{AB} \\ &= (a_0|0\rangle_A + a_1|1\rangle_A)\left(\frac{1}{\sqrt{2}}(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B)\right)\end{aligned}$$

Let CNOT (*controlled-not*) gate be a two-qubit unitary operation described by the operator $|00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 10| + |10\rangle\langle 11|$. Alice now performs a CNOT gate on the input qubit and her part of the shared EPR pair. The resulting state is then

$$\begin{aligned}|\theta_1\rangle_{AB} =& \frac{a_0}{\sqrt{2}}|0\rangle_A(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B) \\ &+ \frac{a_1}{\sqrt{2}}|1\rangle_A(|1\rangle_A|0\rangle_B + |0\rangle_A|1\rangle_B)\end{aligned}$$

Let the Hadamard transform be a single-qubit unitary operation with operator $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\langle 0| + \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\langle 1|$. Alice next performs a Hadamard transform on her input qubit. The resulting state then is

$$\begin{aligned}|\theta_2\rangle_{AB} =& \frac{a_0}{2}(|0\rangle_A + |1\rangle_A)(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B) \\ &+ \frac{a_1}{2}(|0\rangle_A - |1\rangle_A)(|1\rangle_A|0\rangle_B + |0\rangle_A|1\rangle_B) \\ =& \frac{1}{2}(|00\rangle_A(a_0|0\rangle_B + a_1|1\rangle_B) \\ &+ |01\rangle_A(a_0|1\rangle_B + a_1|0\rangle_B)) \\ &+ \frac{1}{2}(|10\rangle_A(a_0|0\rangle_B - a_1|1\rangle_B) \\ &+ |11\rangle_A(a_0|1\rangle_B - a_1|0\rangle_B))\end{aligned}$$

Alice next measures the two qubits in her possession in the standard basis for $\mathbb{C}^4$ and sends the result of the measurement to Bob.

Let the four Pauli gates be the single-qubit unitary operations: identity, $P_{00} = |0\rangle\langle 0| + |1\rangle\langle 1|$; bit flip, $P_{01} = |1\rangle\langle 0| + |0\rangle\langle 1|$; phase flip, $P_{10} = |0\rangle\langle 0| - |1\rangle\langle 1|$; and bit flip together with phase flip, $P_{11} = |1\rangle\langle 0| - |0\rangle\langle 1|$. On receiving the two bits c_0c_1 from Alice, Bob performs the Pauli gate $P_{c_0c_1}$ on his qubit. It is now easily verified that the resulting state of the qubit with Bob would be $a_0|0\rangle_B + a_1|1\rangle_B$. The input qubit is successfully teleported from Alice to Bob! Please refer to Fig. 1 for the overall protocol. ■

Super-Dense Coding

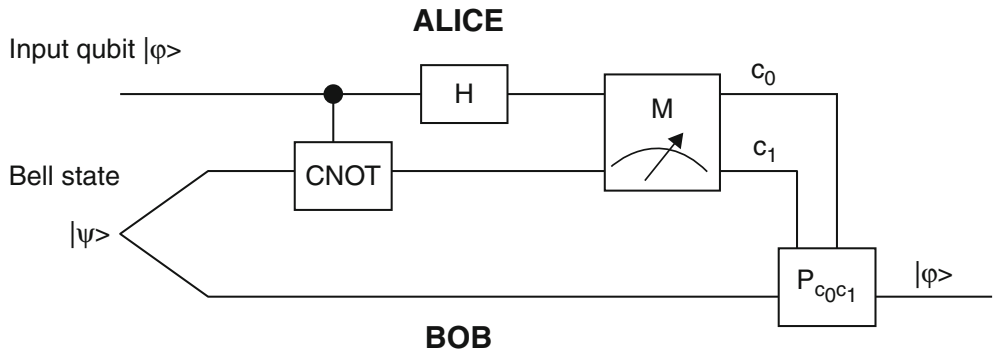
Super-dense coding [22] protocol is a dual to the teleportation protocol. In this protocol, Alice transmits 2 cbits of information to Bob using 1 qubit of communication and 1 shared ebit. It is discussed more elaborately in another article in the encyclopedia.

Lower Bounds on Resources

The above implementation of teleportation requires 2 cbits and 1 ebit for teleporting 1 qubit. It was argued in [8] that these resource requirements are also independently optimal. That is, 2 cbits need to be communicated to teleport a qubit independent of how many ebits are used. Also 1 ebit is required to teleport one qubit independent of how much (possibly two-way) communication is used.

Remote State Preparation

Closely related to the problem of teleportation is the problem of *remote state preparation* (RSP) introduced by Lo [21]. In teleportation Alice is just given the state to be teleported in some input register and has no other information about it. In contrast, in RSP, Alice knows a *complete*



Teleportation of Quantum States, Fig. 1 Teleportation protocol. H represents Hadamard transform and M represents measurement in the standard basis for $\mathbb{C}^4$

description of the input state that needs to be teleported. Also in RSP, Alice is not required to maintain any correlation of the input state with the other parts of a possibly larger state as is achieved in teleportation. The extra knowledge that Alice possesses about the input state can be used to devise protocols for probabilistically exact RSP with one cbit and one ebit per qubit asymptotically [9]. In a probabilistically exact RSP, Alice and Bob can abort the protocol with a small probability; however, when they do not abort, the state produced with Bob at the end of the protocol is exactly the state that Alice intends to send.

Teleportation as a Private Quantum Channel

The teleportation protocol also satisfies an interesting privacy property as follows. If there was a third party, say Eve, having access to the communication channel between Alice and Bob, then Eve learns nothing about the input state of Alice that she is teleporting to Bob. This is because the distribution of the classical messages of Alice is always uniform, independent of her input state. Such a channel is referred to as a *private quantum channel* [2, 11, 18].

Quantum State Redistribution

The teleportation protocol is a part of a wide range of information theoretical tasks. A more general task, referred to as *quantum state redistribution*, is as follows. Three parties, Alice, Bob, and Referee, share a joint quantum state $|\Psi\rangle_{ACBR}$, where A, C registers are with Alice, B is with Bob, and R is with Referee. The task is to transfer register C to Bob. In the *asymptotic setting* (in this limit of infinite copies of the input state), it was shown by [14, 26] that the number of cbits to be transmitted is I(R : C|B) (quantum mutual information between R and C conditioned on B). A sub-task of quantum state redistribution in which register A is not present is referred to as *quantum state merging*, which was used by [17] to give an operational interpretation to negative *quantum conditional entropy*. Another sub-task in which register B is not present is referred to as *quantum state splitting*. These protocols have also been well studied in the *single-shot* setting where a single copy of the quantum state is available [1, 4, 5, 13].

Applications

Apart from the main application of transporting quantum states over large distances using only classical channel, the teleportation protocol finds other important uses as well. A generalization of this protocol to implement unitary operations [12] is used in *fault-tolerant computation* in order to construct an infinite class of fault-tolerant gates in a uniform fashion. In another application, a form of teleportation called as the error correcting teleportation, introduced by

Knill [19], is used in devising quantum circuits that are resistant to very high levels of noise.

Ideas from quantum teleportation form the basis of measurement-based models of quantum computation. Starting from an arbitrary state $|\psi\rangle = a|0\rangle + b|1\rangle$, and an ancilla $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, apply the controlled-Z gate on the two qubits to obtain $a|0,+\rangle + b|1,-\rangle$ (here $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$). Measuring the first qubit in $\{|+\rangle, |-\rangle\}$ basis gives the state $X^m|\psi\rangle$, where $m \in \{0, 1\}$ is the measurement outcome. In particular, if a phase gate Z_θ acted after controlled-Z unitary, then outcome would be $X^m Z_\theta|\psi\rangle$. Thus, the information about the state $|\psi\rangle$ is still preserved after the measurement. By preparing large arrays of standard entangled states, and performing single-qubit unitary measurement, one can hence simulate any quantum circuit, up to unitaries that depend on measurement outcomes. More details can be found in [23].

This protocol can in particular be used to perform a blind quantum computation. Given a state $|\psi\rangle$, Alice can apply a random Pauli operator on it and obtain the state $|\psi'\rangle$. Then the state is sent to Bob, who uses the idea in previous paragraph to realize a desired phase gate Z_θ on the state. The input state to Bob is completely random, yet a unitary upto measurement outcome is realized by Bob, who then sends the state back to Alice. Since Alice knows which Pauli operation she applied, she can recover back the state $Z_\theta|\psi\rangle$. More details can be found in [6, 16].

Experimental Results

Teleportation protocol has been experimentally realized in various different forms, to name a few, by Boschi et al. [3] using optical techniques, by Bouwmeester et al. [10] using photon polarization, by Nielsen et al. [24] using *Nuclear magnetic resonance* (NMR), and by Ursin et al. [25] using photons for long distance.

Krauter et al. [20] have achieved the teleportation of a complicated quantum state: a continuous variable state stored in the collective spin of an atomic ensemble. Unlike qubits that can be measured only in the 0 or 1 state, the outcome of a continuous variable measurement is a real number, like the position and momentum. The quantum states prepared and teleported in the experiment are actually similar to the coherent states of a harmonic oscillator – describing a particle in a harmonic potential well moving under the influence of a classical force.

Majorana bound states are localized zero-energy excitations of a superconductor. An isolated Majorana bound state is an equal superposition of electron and hole excitations and therefore not a fermionic state. Instead, two spatially separated Majorana bound states together make one zero-energy fermion level which can be either occupied or empty. This defines a two-level system which can store quantum information nonlocally, as needed to realize topological quantum computation. In [15] a nonlocal electron transfer process due to Majorana bound states in a mesoscopic superconductor is predicted. An electron which is injected into one Majorana bound state can go out from another one far apart maintaining phase coherence. The transmission phase shift is independent of the distance "traveled." In this sense this phenomenon can be called "electron transportation."

In summary this work reveals a striking nonlocal electron transport phenomenon through Majorana bound states in a finite-sized superconductor with charging energy. Most interestingly, the transmission phase shift detects the state of a qubit made of two spatially separated Majorana bound states.

In Baur et al. [7] have benchmarked a teleportation algorithm by tomographic reconstruction of the three-qubit entangled state generated by the circuit up to the single-qubit measurements. Using an entanglement witness, they showed that this state has genuine tripartite entanglement. This technique presents an important step toward making use of teleportation in quantum processors realized in superconducting circuits.

Cross-References

▸ Quantum Dense Coding

Recommended Reading

1. Anshu A, Devabathini VK, Jain R (2014) Near optimal bounds on quantum communication complexity of single-shot quantum state redistribution. http://arxiv.org/abs/1410.3031
2. Ambainis A, Mosca M, Tapp A, de Wolf R (2000) Private quantum channels. In: Proceedings of the 41st annual IEEE symposium on foundations of computer science, Redondo Beach, pp 547–553
3. Boschi D, Branca S, Martini FD, Hardy L, Popescu S (1998) Experimental realization of teleporting an unknown pure quantum state via dual classical and Einstein-Podolski-Rosen channels. Phys Rev Lett 80:1121–1125
4. Berta M, Christandl M, Touchette D (2014) Smooth entropy bounds on one-shot quantum state redistribution. http://arxiv.org/abs/1409.4338
5. Berta M (2009) Single-shot quantum state merging. Master's thesis, ETH Zurich
6. Broadbent A, Fitzsimons J, Kashefi E (2009) Universal blind quantum computation. In: Proceedings of the 50th annual IEEE symposium on foundations of computer science, Atlanta
7. Baur M, Fedorov A, Steffen L, Filipp S, da Silva MP, Wallraff A (2012) Benchmarking a quantum teleportation protocol in superconducting circuits using tomography and an entanglement witness. Phys Rev Lett 108(4):040502
8. Bennett C, Brassard G, Crepeau C, Jozsa R, Peres R, Wootters W (1993) Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. Phys Rev Lett 70:1895–1899
9. Bennett CH, Hayden P, Leung W, Shor PW, Winter A (2005) Remote preparation of quantum states. IEEE Trans Inf Theory 51:56–74
10. Bouwmeester D, Pan JW, Mattle K, Eible M, Weinfurter H, Zeilinger A (1997) Experimental quantum teleportation. Nature 390(6660):575–579
11. Boykin PO, Roychowdhury V (2003) Optimal encryption of quantum bits. Phys Rev A 67:042317
12. Chaung IL, Gottesman D (1999) Quantum teleportation is a universal computational primitive. Nature 402:390–393
13. Datta N, Hsieh M-H, Oppenheim J (2014) An upper bound on the second order asymptotic expansion for the quantum communication cost of state redistribution. http://arxiv.org/abs/1409.4352
14. Devetak I, Yard J (2008) Exact cost of redistributing multipartite quantum states. Phys Rev Lett 100:230501
15. Fu L (2010) Electron teleportation via Majorana bound states in a mesoscopic superconductor. Phys Rev Lett 104(5):056402
16. Herder C, http://www.scottaaronson.com/showcase2/report/charles-herder.pdf
17. Horodecki M, Oppenheim J, Winter A (2007) Quantum state merging and negative information. Commun Math Phys 269:107–136
18. Jain R (2006) Resource requirements of private quantum channels and consequence for oblivious remote state preparation. Technical report, arXive:quant-ph/0507075
19. Knill E (2005) Quantum computing with realistically noisy devices. Nature 434:39–44
20. Krauter H, Salart D, Muschik CA, Petersen JM, Shen JM, Fernholz T, Polzik ES (2013) Deterministic quantum teleportation between distant atomic objects. Nat Phys 9:400–404
21. Lo H-K (2000) Classical communication cost in distributed quantum information processing – a generalization of quantum communication complexity. Phys Rev A 62:012313
22. Nielsen M, Chuang I (2000) Quantum computation and quantum information. Cambridge University Press, Cambridge/New York
23. Nielsen MA (2005) Cluster-state quantum computation. http://arxiv.org/abs/quant-ph/0504097
24. Nielsen MA, Knill E, Laflamme R (1998) Complete quantum teleportation using nuclear magnetic resonance. Nature 396(6706):52–55
25. Ursin R, Jennewein T, Aspelmeyer M, Kaltenbaek R, Lindenthal M, Zeilinger A (2004) Quantum teleportation link across the danube. Nature 430:849
26. Yard JT, Devetak I (2009) Optimal quantum source coding with quantum side information at the encoder and decoder. IEEE Trans Inf Theory 55:5339–5351

Temperature Programming in Self-Assembly

Scott M. Summers
Department of Computer Science, University of Wisconsin – Oshkosh, Oshkosh, WI, USA

Keywords

Algorithm self-assembly; DNA computing; Kolmogorov complexity; Scaled shapes; Self-assembly; Tile assembly model; Temperature programming; Tile complexity; Tile self-assembly

Years and Authors of Summarized Original Work

2005; Aggarwal, Cheng, Goldwasser, Kao, Moisset de Espanés, Schweller
2006; Kao, Schweller
2012; Summers

Problem Definition

Self-assembly is a process by which a small number of fundamental components automatically coalesce to form a target structure. In 1998, Winfree [10] introduced the abstract Tile Assembly Model (aTAM) as a deliberately oversimplified, discrete mathematical model of the DNA tile self-assembly pioneered by Seeman [6]. The aTAM "effectivizes" classical Wang tiling [9] in the sense that the former augments the latter with a mechanism for sequential "growth" of a tile assembly. Very briefly, in the aTAM, the fundamental components are un-rotatable, translatable square "tile types" whose sides are labeled with (alpha-numeric) glue "colors" and (integer) "strengths." Two tiles that are placed next to each other *bind* if the glues on their abutting sides match in both color and strength, and the common strength is at least a certain (integer) "temperature." Self-assembly starts from a "seed" tile type, typically assumed to be placed at the origin of the coordinate system, and proceeds nondeterministically and asynchronously as tiles bind to the seed-containing assembly one at a time.

The *multiple temperature* model [2, 3, 8] is a natural generalization of the aTAM, where the temperature of a tile system is dynamically adjusted by the experimenter as self-assembly proceeds. In the multiple temperature model, a tile assembly system (TAS) is defined as an ordered triple $\mathcal{T} = \left(T, \sigma, \langle \tau_i \rangle_{i=0}^{k-1}\right)$, where T is a tile set, σ is a "seed assembly," and the third component $\langle \tau_i \rangle_{i=0}^{k-1}$ is a sequence of nonnegative integer temperatures.

Intuitively, self-assembly in the multiple temperature TAS $\mathcal{T}$ is carried out in k phases. In the first temperature phase, tiles are added to the existing assembly as they normally would be in the aTAM until a τ_0-stable terminal assembly is reached. In phase two, tiles can accrete to the existing assembly if they can do so with at least strength τ_1. Also, and at any time during the second temperature phase, if there is ever a cut of the assembly having a strength less than τ_1, then all of the tiles on the side of the cut not containing the seed can be removed from the assembly. When a τ_1-stable terminal assembly is reached in phase two, phase three begins and proceeds in a similar fashion. This process continues through the final temperature phase in which tiles are added or removed with respect to the temperature τ_{k-1} until reaching a τ_{k-1}-stable terminal assembly. See Fig. 1 for an example of this process.

Problem 1 (Reducing tile complexity for the self-assembly of shapes through temperature programming) Given a *shape* $X \subseteq \mathbb{Z}^2$, find a TAS $\mathcal{T} = \left(T, \sigma, \langle \tau_i \rangle_{i=0}^{k-1}\right)$ such that X uniquely self-assembles in $\mathcal{T}$ and $|T|$ and k are minimal.

In some cases, it is sufficient to uniquely self-assemble a scaled-up version of X, i.e., $X^c = \left\{(x, y) \in \mathbb{Z}^2 \mid \left(\left\lfloor \frac{x}{c} \right\rfloor, \left\lfloor \frac{y}{c} \right\rfloor\right) \in X\right\}$. Intuitively, X^c is the shape obtained by replacing each point in X with a $c \times c$ block of points. We refer to the natural number c as the *scaling factor* or *resolution loss*.

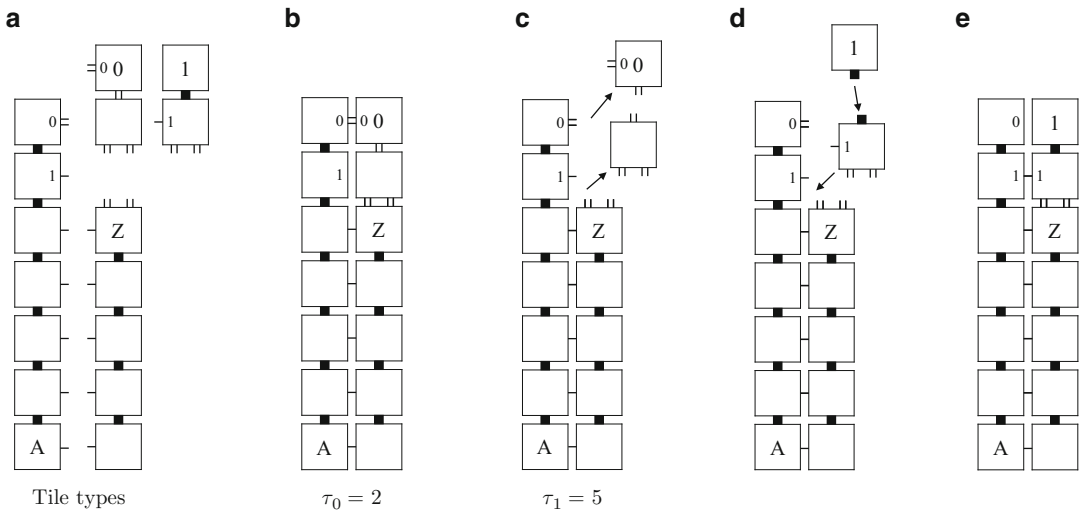
Key Results

Thin Rectangles

Aggarwal, Cheng, Goldwasser, Kao, Moisset de Espanés, and Schweller [2] proved that, in the aTAM, $\Omega\left(\frac{N^{1/k}}{k}\right)$ unique tile types are required to uniquely self-assemble a rectangle of size $k \times N$, where $k < \frac{\log N}{\log\log N - \log\log\log N}$ (this restriction makes the rectangle "thin"). In the same paper, the authors reduced this bound to $O\left(\frac{\log N}{\log\log N}\right)$ in the 2-temperature model. Intuitively, their construction builds a $j \times N$ rectangle for an optimal value of $j \gg k$. Then, when the temperature is raised, the top $j - k$ rows detach, leaving a (stable) $k \times N$ rectangle.

Squares

In the aTAM, the minimum number of unique tile types required to uniquely self-assemble an $N \times N$ square is $O\left(\frac{\log N}{\log\log N}\right)$ [1]. In 2006, Kao and Schweller [3] reduced this bound to $O(1)$ using $O(\log N)$ temperature changes. Their construction relies on a simple yet ingenious gadget called the "bit-flip gadget." Basically, a bit-flip



Temperature Programming in Self-Assembly, Fig. 1 Thick notches are strength 5, and thin notches are strength 1. Not all glue labels are shown

gadget is a constant set of tile types that can be programmed, via a carefully chosen sequence of temperature values, to build a rectangle of constant size that encodes either 0 or 1. Figure 1 gives an example of a simple bit-flip gadget. In their main construction, Kao and Schweller first self-assemble a sequence of $O(\log N)$ bit-flip gadgets, using $O(\log N)$ temperature changes. The result is a rectangle of length $O(\log N)$ that encodes N in binary. Finally, the temperature is lowered to 2, and a standard square-building tile set (i.e., [5], but without the "seed row" tile types) is used to fill in the rest of the square. In the same paper, Kao and Schweller also prove that there is no smooth tradeoff, i.e., for a TAS $\mathcal{T} = \left(T, \sigma, \langle \tau_i \rangle_{i=0}^{k-1}\right)$ that uniquely self-assembles an $N \times N$ square, it cannot be the case that $|T| = o\left(\frac{\log N}{\log \log N}\right)$ and $k = o(\log N)$.

Scaled Finite Shapes

In [3], Kao and Schweller posed the following question: Is it possible to have a tile set of size $O(1)$ that can, via some sequence of temperature values, uniquely self-assemble into an arbitrary finite shape (as specified by the sequence of temperature values)? In 2012, Summers [8] investigated this question and discovered the following:

1. **The answer to the previous question is "NO."** It turns out that a tile set of size $O(1)$ cannot uniquely self-assemble an arbitrary finite shape via (any number of) temperature values. Technically speaking, Summers proved that, for every tile set T, there exists a finite shape $X \subset \mathbb{Z}^2$, such that, for each temperature sequence $\langle \tau_i \rangle_{i=0}^{k-1}$, $\mathcal{T} = \left(T, \sigma, \langle \tau_i \rangle_{i=0}^{k-1}\right)$ does not uniquely self-assemble X. In the proof, X is always a line of length $|T| + 1$.

2. **Short temperature sequence, big scale factor.** On the positive side, there exists a universal tile set that can be programmed via temperature values to build a scaled version of an arbitrary finite shape. For instance, Summers exhibited a construction in which the bit-flip gadget of Kao and Schweller [3] is combined with the non-seed portion of the optimal shape-building construction by Soloveichik and Winfree [7] to get the following result: there exists a tile set T with $|T| = O(1)$, such that, for every finite shape X, there exists $c \in \mathbb{N}$ and a temperature sequence $\langle \tau_i \rangle_{i=0}^{k-1}$ with $m = O(K(X))$, where $K(X)$ is the Kolmogorov complexity of X (see [4]), such that,

$\mathcal{T} = \left(T, \sigma, \langle \tau_i \rangle_{i=0}^{k-1}\right)$ uniquely self-assembles X^c.

3. **Long temperature sequence, small scale factor.** Note that, in the previously mentioned construction, the scaling factor can be quite large. Technically, the scaling factor c depends on the running time of π, whence $c = \text{poly}(\text{time}(\pi))$. In a truly nanoscale setting, it is necessary to have a construction in which the scaling factor is always small or, better yet, bounded by a constant independent of the shape being assembled. Summers gave such a construction: there exists a tile set T with $|T| = O(1)$, such that, for every finite shape X, there exists a temperature sequence $\langle \tau_i \rangle_{i=0}^{k-1}$ with $m = O(|X|)$, such that, $\mathcal{T} = \left(T, \sigma, \langle \tau_i \rangle_{i=0}^{k-1}\right)$ uniquely self-assembles X[22]. This construction utilizes a modified bit-flip gadget, in the form of a bit-flip "square," which is essentially a bit-flip gadget that can be programmed (via temperature values) to follow the directions specified by a Hamiltonian path through a shape (not every shape has a Hamiltonian path, but every shape scaled up by a factor of 2 does).

Open Problems

Does there exist a tile set T, with $|T| = O(1)$ and $c \in \mathbb{N}$, such that, for every finite shape X, there exists a temperature sequence $\langle \tau_i \rangle_{i=0}^{k-1}$, with $m = O(K(X))$, such that, X^c uniquely self-assembles in $\mathcal{T} = \left(T, \sigma, \langle \tau_i \rangle_{i=0}^{k-1}\right)$?

Cross-References

- Combinatorial Optimization and Verification in Self-Assembly
- Intrinsic Universality in Self-Assembly
- Patterned Self-Assembly Tile Set Synthesis
- Randomized Self-Assembly
- Robustness in Self-Assembly
- Self-Assembly at Temperature 1
- Self-Assembly of Fractals
- Self-Assembly of Squares and Scaled Shapes
- Self-Assembly with General Shaped Tiles
- Temperature Programming in Self-Assembly

Recommended Reading

1. Adleman L, Cheng Q, Goel A, Huang MD (2001) Running time and program size for self-assembled squares. In: Proceedings of the 33rd annual ACM symposium on theory of computing, Heraklion, pp 740–748
2. Cheng Q, Aggarwal G, Goldwasser MH, Kao MY, Schweller RT, de Espanés PM (2005) Complexities for generalized models of self-assembly. SIAM J Comput 34:1493–1515
3. Kao MY, Schweller RT (2006) Reducing tile complexity for self-assembly through temperature programming. In: Proceedings of the 17th annual ACM-SIAM symposium on discrete algorithms, Miami, pp 571–580
4. Li M, Vitanyi P (1997) An introduction to Kolmogorov complexity and its applications, 2nd edn. Springer, New York
5. Rothemund PWK, Winfree E (2000) The program-size complexity of self-assembled squares (extended abstract). In: Proceedings of the thirty-second annual ACM symposium on theory of computing, Portland, pp 459–468
6. Seeman NC (1982) Nucleic-acid junctions and lattices. J Theor Biol 99:237–247
7. Soloveichik D, Winfree E (2007) Complexity of self-assembled shapes. SIAM J Comput 36(6):1544–1569
8. Summers SM (2012) Reducing tile complexity for the self-assembly of scaled shapes through temperature programming. Algorithmica 63(1–2):117–136
9. Wang H (1961) Proving theorems by pattern recognition – II. Bell Syst Tech J XL(1):1–41
10. Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology

Testing Bipartiteness in the Dense-Graph Model

Oded Goldreich[1] and Dana Ron[2]
[1]Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel
[2]School of Electrical Engineering, Tel-Aviv University, Ramat-Aviv, Israel

Keywords

Graph properties; Property testing; Randomized algorithms; Sublinear time algorithms

Years and Authors of Summarized Original Work

1998; Goldreich, Goldwasser, Ron
2002; Alon, Krivelevich

Problem Definition

A graph is bipartite (or 2-colorable) if its vertices can be partitioned into two sets such that there are no edges between pairs of vertices that reside in the same set.

Given a (simple) graph, the task is to determine whether it is bipartite or is "far" from being bipartite. Thus, the standard decision problem is relaxed by allowing any answer when the graph is not bipartite but is "close" to some bipartite graph. We focus on dense graphs (i.e., for which the number of edges is quadratic in the number of vertices) and wish to solve the aforementioned "approximate decision" problem in constant time, given access to a data structure that answers adjacency queries in unit time.

To complete the formulation of the problem, we need to define the distance between graphs and describe how the graph is accessed. The distance between the graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ is determined by the symmetric difference between their edge sets (i.e., $E_1 \Delta E_2$), and we say that they are ϵ-close (resp., ϵ-far) if $|E_1 \Delta E_2| \leq \epsilon \cdot |V|^2$ (resp., $|E_1 \Delta E_2| > \epsilon \cdot |V|^2$). Note that this definition is appropriate for dense graphs (i.e., when the number of edges is $\Omega(|V|^2)$), whereas any two sparse graphs are deemed to be close by it. (An alternative model that is more suitable for sparse graphs is presented in this encyclopedia's entry *Testing Bipartiteness of Graphs in Sublinear Time*.) We say that $G = (V, E)$ is ϵ-far from a graph property $\mathcal{P}$ (i.e., a set of graphs that is closed under isomorphism) if for every $G' \in \mathcal{P}$ it holds that G is ϵ-far from G'.

We consider algorithms that make oracle queries to the input graph, denoted $G = (V, E)$. Specifically, the algorithm can perform adjacency queries of the form $(u, v) \in V^2$, which are answered by 1 if $\{u, v\} \in E$ any by 0 otherwise. We can now define property testing in this model.

Definition 1 (testing graph properties in the dense-graph model) A tester for a graph property $\mathcal{P}$ is a randomized algorithm that is given as input a size parameter N and a proximity parameter ϵ as well as access to an adjacency oracle for an N-vertex graph $G = ([N], E)$. The tester should output a binary verdict that satisfies the following two conditions.

1. If $G \in \mathcal{P}$, then the tester accepts with probability at least $2/3$.
2. If G is ϵ-far from $\mathcal{P}$, then the tester accepts with probability at most $1/3$.

A tester has one-sided error if it accepts every graph in $\mathcal{P}$ with probability 1. A tester is nonadaptive if it determines all its queries based solely on its internal coin tosses *(and the parameters N and ϵ)*; otherwise it is adaptive.

Here we are interested in the case that $\mathcal{P}$ is the set of all bipartite graphs, and we seek a tester of time complexity that only depends on the proximity parameter, denoted ϵ, and is independent of the size of the graph.

Key Results

Goldreich, Goldwasser, and Ron [9] showed that there exists a tester for bipartiteness that, given a proximity parameter ϵ and access to an N-vertex graph, runs in time $\text{poly}(1/\epsilon)$. Furthermore, the tester is nonadaptive and has one-sided error; that is, it always accepts bipartite graphs.

The fact that there exist properties that can be tested in time that only depends on the proximity parameter should not come as a surprise. It is well known that the average value of a (bounded) function defined over a huge domain can be approximated up to a factor of $1 \pm \epsilon$ by taking $O(1/\epsilon^2)$ samples. This approximation problem

can be cast as a property testing problem (even as one that refers to graphs in the current model, by considering the edge density of a graph). But the foregoing approximation problem is highly unstructured (i.e., it merely refers to the average of values regardless of anything else), whereas bipartite graphs are highly structured (and the edge density in them is almost arbitrary).

Turning back to bipartiteness, this property is a special case of k-colorability, where $k = 2$. (A graph is k-colorable if its vertices can be partitioned into k sets such that there are no edges between pairs of vertices that reside in the same set.) The tester for bipartiteness, and more generally for k-colorability (for any $k \geq 2$), is very simple. It merely selects a sample of $\text{poly}(1/\epsilon)$ random vertices and accepts if and only if the subgraph induced by the selected vertices is k-colorable. In fact, as shown by Alon and Krivelevich [1] (improving on the bounds obtained in [9]), in the case of $k = 2$ (bipartiteness), a sample of $\tilde{O}(1/\epsilon)$ vertices suffices, and in general a sample of size $\tilde{O}(k/\epsilon^2)$ is sufficient. (The notation $\tilde{O}(q)$ "hides" polylogarithmic factors in q.)

Clearly, the algorithm always accepts k-colorable graphs, and its running time is $\text{poly}(1/\epsilon)$ if $k = 2$ and exponential in $\text{poly}(1/\epsilon)$ (the sample size) otherwise. The analysis boils down to proving that if the graph G is ϵ-far from being k-colorable, then it is rejected with probability at least $2/3$. Below we shall sketch the argument for the case of $k = 2$ and when one uses a sample of $\tilde{O}(1/\epsilon^2)$ random vertices.

We view the random sample (of vertices) as a union of two disjoint sets, denoted U and S, where $t \stackrel{\text{def}}{=} |U| = \tilde{O}(1/\epsilon)$ and $m \stackrel{\text{def}}{=} |S| = O(t/\epsilon)$. We consider all possible (2-way) partitions of U and associate a partial partition of V with each such partition of U. Specifically, given a partition of U, denoted (U_1, U_2), we place all neighbors of U_1 (resp., of U_2) opposite to U_1 (resp., U_2). Indeed, such a placing is forced if we seek a partition of V that is consistent with the given partition (U_1, U_2) of U. One may show that, with high probability, most high-degree vertices in V have at least one neighbor in U, and so the partition of these vertices is forced by the partition of U. Since there are relatively few edges incident to vertices that do not neighbor U, there must be many edges that violate the partition induced by (U_1, U_2) (i.e., their endpoints are forced to be on the same side of the induced partition). It follows that when we take the additional sample S and perform all queries on pairs in $U \times S$ and $S \times S$, with high probability, we detect a violating edge with respect to each of the $2^{|U|}$ induced partitions, thus ruling out all potential partitions of U. This implies that with high probability, the subgraph induced by $U \cup S$ is not bipartite. Let us stress the key observation: *It suffices to rule out relatively few* (partial) *partitions of* V (i.e., these $2^{|U|}$ partitions induced by partitions of U) rather than all ($2^{|V|}$) possible partitions of V.

Applications

The procedure employed in the above analysis yields a randomized $\text{poly}(1/\epsilon) \cdot N$-time algorithm for 2-partitioning a bipartite graph such that (with high probability) at most ϵN^2 edges lie within the same side. This is done by running the tester, determining a partition of U (defined as in the proof) that is consistent with the bipartite partition of $U \cup S$, and partitioning V as done in the proof (with vertices that do not neighbor U, or neighbor both U_1 and U_2, placed arbitrarily). Thus, the placement of each vertex is determined by inspecting at most $\tilde{O}(1/\epsilon)$ entries of the adjacency matrix. Furthermore, the aforementioned partition of U constitutes a succinct representation of the 2-partition of the entire graph. All this is a typical consequence of the fact that the analysis of the tester follows the "enforce-and-test" paradigm (see the survey of Ron [12, Sec. 4]).

Open Problems

As stated above, a more refined analysis yields a nonadaptive tester that inspects the subgraph induced by $\tilde{O}(1/\epsilon)$ random vertices. One can easily show that this result is almost optimal with respect to testers that inspect an induced subgraph. Furthermore, as Bogdanov and Trevisan show [3], a query complexity of $O(1/\epsilon^2)$ is optimal for any nonadaptive

tester for bipartiteness, whereas any adaptive tester requires $\Omega(\epsilon^{-3/2})$ queries. This raises the question of *what is the query complexity of adaptive testers for bipartiteness*.

While Goldreich and Trevisan [8] have shown that the gap between adaptive and nonadaptive testers (in the dense-graph model) is at most quadratic, the above question falls within the uncertainly left open by the quadratic upper bound. Furthermore, Gonen and Ron [10] showed that N-vertex graphs of maximum degree $O(\epsilon N)$ can be tested for bipartiteness in time $\tilde{O}(\epsilon^{-3/2})$, whereas the aforementioned lower bound of [3] holds also for such graphs.

The general question of the gap between adaptive and nonadaptive testers was studied by Goldreich and Ron [7]. They showed that there exist graph properties that have an adaptive tester that runs in time $\tilde{O}(1/\epsilon)$, for which any nonadaptive tester requires $\Omega(\epsilon^{-3/2})$ queries. They conjectured that there exist graph properties that have an adaptive tester that runs in time $\tilde{O}(1/\epsilon)$, for which any nonadaptive tester requires $\Omega(\epsilon^{-2})$ queries.

Cross-References

- Testing Bipartiteness of Graphs in Sublinear Time

Comments for the Recommended Reading

The current entry falls within the scope of property testing (see the surveys [5, 6, 11, 12]). A general definition of this setting was first put forward by Rubinfeld and Sudan [13]. This definition was further generalized and systematically investigated by Goldreich, Goldwasser, and Ron [9], who focused on testing graph properties in the dense-graph model. Alternative models for testing graph properties are discussed in this encyclopedia's entry cited above.

As already noted, the tester for k-colorability described above was suggested and analyzed in [9]. A tighter analysis that yields the best bounds known was subsequently provided in [1]. Testers for any "graph partition property" (including testing that a graph contains a clique of certain density or has a bisection of certain density) were also presented in [9].

The fact that all these (nonadaptive) testers operate by inspecting a random induced subgraph was shown to be no coincidence in [8]. We also mention that the class of graph properties that can be tested (in the dense-graph model) within complexity that is independent of the size of the graph was characterized by Alon et al. [2]. Their characterization is related to Szemerédi's Regular Partitions [14]. A different characterization, based on graph limits, was proved independently by Borgs et al. [4].

Recommended Reading

1. Alon N, Krivelevich M (2002) Testing k-colorability. SIAM J Discret Math 15:211–227
2. Alon N, Fischer E, Newman I, Shapira A (2009) A combinatorial characterization of the testable graph properties: it's all about regularity. SIAM J Comput 39(1):143–167
3. Bogdanov A, Trevisan L (2004) Lower bounds for testing bipartiteness in dense graphs. In: Proceedings of the ninteenth IEEE annual conference on computational complexity (CCC), Amherst, pp 75–81
4. Borgs C, Chayes J, Lovász L, Sós VT, Szegedy B, Vesztergombi K (2006) Graph limits and parameter testing. In: Proceedings of the thirty-eighth annual ACM symposium on the theory of computing (STOC), Seattle, pp 261–270
5. Goldreich O (2010) Introduction to testing graph properties, in [6]
6. Goldreich O (ed) (2010) Property testing: current research and surveys. LNCS, vol 6390. Springer, Heidelberg
7. Goldreich O, Ron D (2011) Algorithmic aspects of property testing in the dense graphs model. SIAM J Comput 40(2):376–445
8. Goldreich O, Trevisan L (2003) Three theorems regarding testing graph properties. Random Struct Algorithms 23(1):23–57
9. Goldreich O, Goldwasser S, Ron D (1998) Property testing and its connections to learning and approximation. J ACM 45:653–750
10. Gonen M, Ron D (2010) On the benefits of adaptivity in property testing of dense graphs. Algorithmica 58(4):811–830
11. Ron D (2008) Property testing: a learning theory perspective. Found Trends Mach Learn 1(3):307–402

12. Ron D (2010) Algorithmic and analysis techniques in property testing. Found Trends Theor Comput Sci 5:73–205
13. Rubinfeld R, Sudan M (1996) Robust characterization of polynomials with applications to program testing. SIAM J Comput 25(2):252–271
14. Szemerédi E (1978) Regular partitions of graphs. In: Proceedings, Colloque Inter. CNRS, Paris, pp 399–401

Testing Bipartiteness of Graphs in Sublinear Time

Oded Goldreich[1] and Dana Ron[2]
[1]Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel
[2]School of Electrical Engineering, Tel-Aviv University, Ramat-Aviv, Israel

Keywords

Graph properties; Property testing; Randomized algorithms; Sublinear time algorithms

Years and Authors of Summarized Original Work

1998; Goldreich, Ron
2004; Kaufman, Krivelevich, Ron

Problem Definition

A graph is bipartite (or 2-colorable) if its vertices can be partitioned into two sets such that there are no edges between pairs of vertices that reside in the same set.

Given a (simple) graph, the task is to determine whether it is bipartite or is "far" from being bipartite. Thus, the standard decision problem is relaxed by allowing any answer when the graph is not bipartite but is "close" to some bipartite graph. We wish to solve this "approximate decision" problem in sublinear time, given access to a data structure that answers adjacency and incidence queries in unit time.

To complete the formulation of the problem, we need to define the distance between graphs and describe how the graph is accessed. The distance between two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ is determined by the symmetric difference between their edge sets (i.e., $E_1 \Delta E_2$), where they are ϵ-close (resp., ϵ-far) if $|E_1 \Delta E_2| \leq \epsilon \cdot (|E_1| + |E_2|)$ (resp., $|E_1 \Delta E_2| > \epsilon \cdot (|E_1| + |E_2|)$). A *graph property* is a set of graphs that is closed under isomorphism, and we say that $G = (V, E)$ is ϵ-far from a graph property $\mathcal{P}$ if for every $G' \in \mathcal{P}$ it holds that G is ϵ-far from G'.

We consider algorithms that make oracle queries to the input graph, denoted $G = (V, E)$. Specifically, an algorithm can perform adjacency queries of the form $(u, v) \in V^2$, which are answered by 1 if $\{u, v\} \in E$ and by 0 otherwise, and incidence queries of the form $(u, i) \in V \times [|V| - 1]$, which are answered by v if v is the ith neighbor of u and by $\perp$ if u has less than i neighbors. (Note that adjacency queries may be quite useless when the graph is very sparse.) We now define property testing in this model.

Definition 1 (testing graph properties with adjacency and incidence queries) A tester for a graph property $\mathcal{P}$ is a randomized algorithm that is given as input a size parameter N and a proximity parameter ϵ as well as access to adjacency and incidence oracles for an N-vertex graph $G = ([N], E)$. The tester should output a binary verdict that satisfies the following two conditions.

1. If $G \in \mathcal{P}$, then the tester accepts with probability at least $2/3$.
2. If G is ϵ-far from $\mathcal{P}$, then the tester accepts with probability at most $1/3$.

A tester has one-sided error if it accepts every graph in $\mathcal{P}$ with probability 1.

Here we are interested in the case that $\mathcal{P}$ is the set of all bipartite graphs, and we seek a tester of time complexity that is sublinear in the size of the graph. The dependence of the running time on the proximity parameter, denoted ϵ, is of secondary concern.

Key Results

Building on the work of Goldreich and Ron [6] on testing bipartiteness of bounded-degree graphs, Kaufman, Krivelevich, and Ron [9] presented a tester for bipartiteness that, given a proximity parameter ϵ and access to an N-vertex graph, runs in time $\sqrt{N} \cdot \text{poly}(\log(N), 1/\epsilon)$. This algorithm uses only incidence queries. In case the number of edges, denoted M, is larger than $N^{3/2}$, the running time can be reduced to $(N^2/M) \cdot \text{poly}(\log N, 1/\epsilon)$ by also using adjacency queries. Furthermore, in both cases, the testers have one-sided error; that is, they always accept bipartite graphs.

From now on, we focus on the special case that $M = O(N)$ and further assume that the graph has constant maximum degree, denoted d.

As a warm-up, we note that the case of $d = 2$ is easy. In this case, we are guaranteed that the graph consists of a collection of paths and cycles, and we only need to check that it does not have short cycles of odd length. Note that such an N-vertex graph is ϵ-far from being bipartite if and only if it contains more than ϵN cycles of odd length, where most of these cycles must have length at most $2/\epsilon$. Hence, in this case, testing bipartiteness can be performed by selecting $O(1/\epsilon)$ random vertices and exploring their neighborhoods up to distance $1/\epsilon$.

In contrast, in the case that $d \geq 3$, any tester for bipartiteness must perform $\Omega(\sqrt{N})$ queries. As shown by Goldreich and Ron [7], this can be proved by considering the following two families of N-vertex graphs (for any even N):

1. The first family, denoted $\mathcal{G}_1^N$, consists of all degree-3 graphs that are composed of the union of a Hamiltonian cycle and a perfect matching. That is, there are N edges connecting the vertices in a cycle, and the other $N/2$ edges form a perfect matching.
2. The second family, denoted $\mathcal{G}_2^N$, is the same as the first, *except* that the perfect matchings allowed are restricted as follows. The distance on the cycle between every two vertices that are connected by a perfect matching edge must be odd.

Clearly, all graphs in $\mathcal{G}_2^N$ are bipartite. It can be shown that almost all graphs in $\mathcal{G}_1^N$ are far from being bipartite. On the other hand, one can prove that an algorithm that performs $o(\sqrt{N})$ queries cannot distinguish between a graph chosen randomly from $\mathcal{G}_2^N$ (which is always bipartite) and a graph chosen randomly from $\mathcal{G}_1^N$ (which with high probability is far from bipartite). Loosely speaking, this follows from the fact that in both cases the algorithm is unlikely to encounter a cycle (among the vertices that it has inspected).

The algorithm itself is based on taking many (i.e., $\text{poly}(1/\epsilon) \cdot \tilde{O}(N^{1/2})$) *random walks* from few (i.e., $O(1/\epsilon)$) randomly selected start vertices, where each walk has length $\text{poly}(\epsilon^{-1} \log N)$. Specifically, given as input N, d, ϵ as well as access to an incidence oracle for an N-vertex graph, $G = (V, E)$, of degree bound d, the algorithm repeats the following steps $T \stackrel{\text{def}}{=} \Theta(\frac{1}{\epsilon})$ times:

1. Uniformly select a vertex s in V.
2. Try to find an odd-length cycle through s:
   (a) Perform $K \stackrel{\text{def}}{=} \text{poly}((\log N)/\epsilon) \cdot \sqrt{N}$ random walks starting from s, each of length $L \stackrel{\text{def}}{=} \text{poly}((\log N)/\epsilon)$.
   (b) Let R_0 (respectively, R_1) denote set of vertices reached from s in an even (respectively, odd) number of steps in any of these walks.
   (c) If $R_0 \cap R_1$ is not empty, then reject.

If the algorithm did not reject in any of the foregoing T iterations, then it accepts.

Clearly, the algorithm always accepts bipartite graphs. Hence, the analysis boils down to proving that if the graph G is ϵ-far from being bipartite, then it is rejected with probability at least $2/3$.

The analysis is quite involved. We confine ourselves to the special case where the graph has a "rapid mixing" feature. It is convenient to modify the random walks so that at each step each neighbor is selected with probability $1/2d$, and otherwise (with probability at least $1/2$) the walk remains at the present vertex. Furthermore, we will consider a single execution of Step (2) starting from an arbitrary vertex, s, which is fixed

in the rest of the discussion. The rapid mixing feature we assume is that, for every vertex v, a (modified) random walk of length L starting at s reaches v with probability approximately $1/N$ (say, up to a factor of 2). Note that if the graph is an expander, then this is certainly the case (since $L = \omega(\log N)$).

The key quantities in the analysis are the following probabilities, referring to the parity *of the length of a path obtained from the random walk by omitting the self-loops* (transitions that remain at current vertex). Let $p^0(v)$ (respectively, $p^1(v)$) denote the probability that a (modified) *random walk of length L, starting at s, reaches v while making an even* (respectively, *odd*) *number of real* (i.e., non-self-loop) *steps*. By the rapid mixing assumption (for every $v \in V$), it holds that

$$\frac{1}{2N} \;<\; p^0(v) + p^1(v) \;<\; \frac{2}{N}\,. \qquad (1)$$

We consider two cases regarding the sum $\sum_{v \in V} p^0(v)p^1(v)$: If the sum is (relatively) "small," we show that V can be 2-partitioned so that there are relatively few edges between vertices that are placed in the same part, which implies that G is close to being bipartite. Otherwise (i.e., when the sum is not "small"), we show that with significant probability, when Step (2) is started at vertex s, it is completed by rejecting G.

In general, the input graph may not be "rapidly mixing," and so the actual analysis, which appears in [6], is far more complex. Another layer of complexity is added when we move from the case of constant degree bound (i.e., d) to the case where the vertex degrees may vary significantly; see [9].

Applications

The foregoing algorithm can be used to find odd-length cycles (of polylogarithmic length) in graphs that are far from lacking such cycles. In general, any one-sided error tester for a property $\mathcal{P}$ finds subgraphs that are inconsistent with the property when invoked on a graph that is far from having property $\mathcal{P}$. Thus, the fact that the bipartite tester finds odd cycles (when invoked on graphs that are far from lacking such cycles) follows directly from its definition, but the fact that these cycles are short is a feature of the specific tester presented above.

Cross-References

▸ Testing Bipartiteness in the Dense-Graph Model

Comments for the Recommended Reading

The current entry falls within the scope of property testing (see [4,5,11,12]). A general definition of this setting was first put forward by Rubinfeld and Sudan [13]. This definition was further generalized and systematically investigated by Goldreich, Goldwasser, and Ron [8], who focused on testing graph properties in the dense-graph model (see this Encyclopedia's entry cited above). The model considered in this entry was suggested by Kaufman, Krivelevich, and Ron [9]. It generalizes a model proposed by Parnas and Ron [10], which in turn generalizes the bounded-degree model of Goldreich and Ron [7]. The latter paper focuses on testers of complexity that only depends on the proximity parameter (i.e., independent of the size of the graph). Among these testers is a two-sided error tester of cycle-freeness; a one-sided error tester for this problem is presented in [3], but its complexity depends on the size of the graph (where this dependence is unavoidable).

As already noted, the bipartiteness tester for the bounded-degree model is due to Goldreich and Ron [6], and it was extended to the general model by Kaufman, Krivelevich, and Ron [9]. In contrast to these results, Bogdanov, Obata, and Trevisan [2] proved that 3-colorability cannot be tested with sublinear query complexity, even in the bounded-degree model. The problem of testing colorability of general graphs was further studied by Ben-Eliezer et al. [1].

Recommended Reading

1. Ben-Eliezer I, Kaufman T, Krivelevich M, Ron D (2012) Comparing the strength of query types in

property testing: the case of testing k-colorability. Comput Complex 22(1):89–135
2. Bogdanov A, Obata K, Trevisan L (2002) A lower bound for testing 3-colorability in bounded-degree graphs. In: Proceedings of the forty-third annual symposium on foundations of computer science (FOCS), Los Alamitos, pp 93–102
3. Czumaj A, Goldreich O, Ron D, Seshadhri C, Shapira A, Sohler C (2012) Finding cycles and trees in sublinear time. Technical Report TR12-035, Electronic Colloquium on Computational Complexity (ECCC), to appear in Random Structures and Algorithms
4. Goldreich O (2010) Introduction to testing graph properties, in [5]
5. Goldreich O (ed) (2010) Property testing: current research and surveys. LNCS, vol 6390. Springer, Heidelberg
6. Goldreich O, Ron D (1999) A sublinear bipartite tester for bounded degree graphs. Combinatorica 19(3):335–373
7. Goldreich O, Ron D (2002) Property testing in bounded degree graphs. Algorithmica 32(2): 302–343
8. Goldreich O, Goldwasser S, Ron D (1998) Property testing and its connections to learning and approximation. J ACM 45:653–750
9. Kaufman T, Krivelevich M, Ron D (2004) Tight bounds for testing bipartiteness in general graphs. SIAM J Comput 33(6):1441–1483
10. Parnas M, Ron D (2002) Testing the diameter of graphs. Random Struct Algorithms 20(2):165–183
11. Ron D (2008) Property testing: a learning theory perspective. Found Trends Mach Learn 1(3):307–402
12. Ron D (2010) Algorithmic and analysis techniques in property testing. Found Trends Theor Comput Sci 5:73–205
13. Rubinfeld R, Sudan M (1996) Robust characterization of polynomials with applications to program testing. SIAM J Comput 25(2):252–271

Testing if an Array Is Sorted

Sofya Raskhodnikova
Computer Science and Engineering Department, Pennsylvania State University, University Park, State College, PA, USA

Keywords

Monotonicity; Property testing; Sorted arrays; Sublinear-time algorithms

Years and Authors of Summarized Original Work

2000; Ergün, Kannan, Kumar, Rubinfeld, Viswanathan
2014; Berman, Raskhodnikova, Yaroslavtsev

Problem Definition

Suppose we would like to check whether a given array of real numbers is sorted (say, in nondecreasing order). Performing this task exactly requires reading the entire array. Here we consider the approximate version of the problem: testing whether an array is sorted or "far" from sorted. We consider two natural definitions of the distance of a given array from a sorted array. Intuitively, we would like to measure how much the input array must change to become sorted. We could measure the change by:

1. The number of entries changed
2. The sum of the absolute values of changes in all entries

It is not hard to see that looking at the number of entries that must be deleted in an array to make it sorted is equivalent to the measure in item 1.

To define the two distance measures formally, let $a = (a_1, \ldots, a_n)$ be the input array and $\mathcal{S}$ be the set of all sorted arrays of length n. We denote by $[n]$ the set $\{1, 2, \ldots, n\}$. The *Hamming distance* from a to $\mathcal{S}$, denoted $\mathrm{dist}(a, \mathcal{S})$, is $\min_{b \in \mathcal{S}} |\{i \in [n] : a_i \neq b_i\}|$. The L_1 *distance* from a to $\mathcal{S}$, denoted $\mathrm{dist}_1(a, \mathcal{S})$, is $\min_{b \in \mathcal{S}} \sum_{i \in [n]} |a_i - b_i|$. Given a parameter $\epsilon \in (0, 1)$, an array is ϵ-far from sorted with respect to the Hamming distance or, respectively, L_1 distance, if the corresponding distance from a to $\mathcal{S}$ is at least ϵn.

A *tester* for sortedness is a randomized algorithm that is given parameters $\epsilon \in (0, 1)$ and n and direct access to an input array a. It is required to accept with probability at least 2/3 if the array is sorted and reject with probability at least 2/3 if the array is ϵ-far from sorted. We consider two types of testers, Hamming and L_1, corresponding to the two distance measures we defined. The

query complexity of a tester is the number of array entries it reads. The goal is to design testers for sortedness with the smallest possible query complexity and running time.

There are two special cases of testers we will discuss. A tester is *nonadaptive* if it makes all queries in advance, before receiving any query answers. A tester has *1-sided error* if it always accepts all sorted arrays.

Bibliographical Notes

The Hamming testers for sortedness were first studied by Ergün et al. [7]. The L_1-testers (and, more generally, L_p-testers, which use the L_p distance for some $p \geq 1$) were introduced by Berman, Raskhodnikova, and Yaroslavtsev [2]. The two distance measures we discussed, dist and dist_1, are identical for arrays with 0/1 entries, which we call Boolean arrays. The L_1-tester in [2] builds on the sortedness tester for Boolean arrays by Dodis et al. [6].

Observe that an array $(a_1, a_2, \ldots, a_n)$ of real numbers can be represented by a function $f : [n] \to \mathbb{R}$ defined by $f(i) = a_i$ for all $i \in [n]$. The formulated problem is equivalent to testing if a function f over an ordered finite domain is monotone. In fact, the L_1-tester we will discuss can be easily adapted to work for functions over infinite domains (specifically, bounded intervals), because its complexity is independent of the domain size. The problem of Hamming testing monotonicity of functions over domain $[n]^d$ was first investigated by Goldreich et al. [11]; general partially ordered domains were studied by Fischer et al. [10]. These problems are discussed in the encyclopedia entry "Monotonicity Testing."

Key Results

Ergün et al. [7] designed two Hamming testers for sortedness that run in time $O\left(\frac{\log n}{\epsilon}\right)$. Later, Bhattacharyya et al. [3] and Chakrabarty and Seshadhri [5] gave different testers with the same complexity, with additional features that made them useful as subroutines in testing monotonicity of high-dimensional functions. Fischer [9] proved that the running time of these testers is optimal. Berman, Raskhodnikova, and Yaroslavtsev [2] gave an L_1-tester for sortedness with running time $O(1/\epsilon)$, which is also optimal.

Here we present two Hamming testers from [3, 7] and the L_1-tester from [2].

Hamming Testers for Sortedness

A Tester Based on Binary Search [7]

We present and analyze the first tester for sortedness (Algorithm 1) with the assumption that all entries in the array a are distinct. This assumption can be removed by treating element a_i as $\langle a_i, i \rangle$ for all $i \in [n]$.

Algorithm 1: Hamming Tester for Sortedness Based on Binary Search

input : parameters n and ϵ; direct access to array a.

1 **repeat** $\lceil \frac{\ln 3}{\epsilon} \rceil$ times:
2 pick $i \in [n]$ uniformly at random;
3 perform a binary search for the value a_i in the array a;
4 **if** a_i is not located by the binary search,
   `// it leads to another position`
5 **reject**;
6 **accept**

Analysis of the First Tester

The tester always accepts all sorted arrays. Now consider an array that is ϵ-far from sorted (in Hamming distance). We say that a position $i \in [n]$ is *searchable* if a_i can be found by a binary search in Step 3 and *not searchable* otherwise. If positions i and j such that $i < j$ are both searchable, then $a_i < a_j$, because both a_i and a_j are in the correct position with respect to their common ancestor in the binary search tree. Thus, all numbers in searchable positions are sorted. Since the array is ϵ-far from sorted, at least ϵn positions must be unsearchable. If the tester picks an unsearchable position in Step 2, it rejects. The probability that it happens in one trial is at least ϵ. Therefore, the probability that it fails to happen in $\lceil \frac{\ln 3}{\epsilon} \rceil$ trials is at most

$$(1-\epsilon)^{\lceil \frac{\ln 3}{\epsilon} \rceil} \leq \exp\left(-\epsilon \cdot \frac{\ln 3}{\epsilon}\right) = 1/3. \quad (1)$$

Thus, the tester rejects an array that is ϵ-far from sorted with probability at least 2/3.

A Tester Based on Graph Spanners [3]

The next tester we discuss is based on graph spanners. We can represent the requirement that the array is sorted as a directed graph G, where nodes are positions in $[n]$, and there is an edge (i, j) for all $i < j$. That is, an edge (i, j) represents that $a_i \leq a_j$. A *2-spanner* of G is a subgraph H of G with vertex set $[n]$ such that for every edge (i, j) in G, there is a path of length at most 2 from i to j in H. It is not hard to construct a 2-spanner of G with at most $n \log n$ edges[3, 12]. (e.g., it can be done using divide-and-conquer as follows: connect all nodes to the one in the middle, orienting the edges towards the nodes with larger indices; remove the middle node; and recurse on the two resulting sublists.)

The tester simply repeats the following step $\lceil \frac{(2 \ln 3) \log n}{\epsilon} \rceil$ times: pick a uniformly random edge (i, j) of the 2-spanner H, and reject if this edge is *violated*, namely, if $a_i > a_j$. If the tester does not find a violated edge, it accepts.

Analysis of the Second Tester

If the input array is sorted, it does not have any violated edges, and the tester always accepts. Now consider an array that is ϵ-far from sorted (in Hamming distance). We call a position $i \in [n]$ *bad* if node i is an endpoint of a violated edge in the 2-spanner H; otherwise, i is *good*. Note that any two good positions i, j such that $i < j$ are connected by a path of length at most 2 of non-violated edges in H. If this path is (i, j), it implies that $a_i \leq a_j$. If this path is (i, k, j) for some node k, it implies that $a_i \leq a_k \leq a_j$. Consequently, for any two good positions i, j such that $i < j$, the numbers a_i and a_j are in the correct order. That is, all numbers in good positions are sorted. As in the analysis of Algorithm 1, we can conclude that there are at least ϵn bad positions. But each bad position is adjacent to a violated edge. Each violated edge can contribute at most two new bad positions. Thus, there are at least $\epsilon n/2$ violated edges. By a simple calculation similar to (1), the second algorithm rejects an array that is ϵ-far from sorted with probability at least 2/3.

L_1-Tester for Sortedness

The L_1-tester for sortedness [2] requires only a uniform sample from the input (as opposed to the ability to query an arbitrary position). It picks $\lceil \frac{2 \ln 6}{\epsilon} \rceil$ positions uniformly and independently at random and accepts iff the numbers in these positions are sorted.

The main ingredient in the analysis of the tester is a reduction to the case of Boolean arrays. It states that if the tester is nonadaptive and has 1-sided error, it suffices to show that it works for Boolean arrays. We omit the proof of the reduction.

Clearly, the L_1-tester is nonadaptive and always accepts sorted arrays. Now consider a Boolean array a which is ϵ-far from sorted. It remains to show that it is rejected with probability at least 2/3. Let X_0 be the set of the $\epsilon n/2$ largest indices i for which $a_i = 0$. Similarly, let X_1 be the set of the $\epsilon n/2$ smallest indices i for which $a_i = 1$. It is easy to show that $i < j$ for all $i \in X_1$ and $j \in X_0$, because a is ϵ-far from sorted. The L_1-tester samples no index from X_0 with probability at most 1/6. The same holds for X_1. Thus, by a union bound, with probability at least 2/3, it samples an index from X_0 and an index from X_1 and detects a violation.

Running time

We explained why the algorithm that samples $\lceil \frac{2 \ln 6}{\epsilon} \rceil$ positions uniformly and independently at random is an L_1-tester for sortedness. Now we analyze its running time for the case of general arrays. The L_1-tester makes $O(1/\epsilon)$ queries. To determine whether the elements in these positions are sorted, the tester can use bucket sort to sort the sampled positions and then simply check if the sequence of queried elements is nondecreasing. Since the positions are sampled uniformly at random, the bucket sort can be implemented to run in expected time $O(1/\epsilon)$, where the expectation is taken over the choice of the samples. By standard methods, the algorithm can be modified to run in $O(1/\epsilon)$ time in the worst case. Observe that the running time does not depend on the length of the input. This is impossible for Hamming testers for sortedness, which, as we mentioned, must query $\Omega(\log n)$ positions [9].

Applications

Testers for sortedness are used as subroutines in other property testers, e.g., for monotonicity of high-dimensional functions [2, 5, 6] and for the property that given points represent ordered vertices of a convex polygon [7]. They are also used to construct fast approximate probabilistically checkable proofs for different optimization problems [8]. Ben-Moshe et al. [1] employed sortedness testers (with additional features) to speed up query evaluation in databases.

Open Problem

Consider the case when all numbers in the input array lie in some specified small set such as $[r]$ for some integer r. As we discussed, for Boolean arrays, testing sortedness can be done in $O(1/\epsilon)$ time [2, 6]. It is not hard to see that for larger ranges, it can be done in $O(r/\epsilon)$ time. When $r \ll n$, can one test sortedness it time polylogarithmic in r? Is $O\left(\frac{\log r}{\epsilon}\right)$ running time achievable?

Fischer's lower bound for testing sortedness [9] applies only to $n \ll r$. The best known lower bound that takes into account both parameters is $\Omega(\min(\log r, \log n))$, due to [4], but it applies only to nonadaptive testers.

Cross-References

▸ Monotonicity Testing

Acknowledgments The author was supported in part by NSF CAREER award CCF-0845701 and Boston University's Hariri Institute for Computing and Center for Reliable Information Systems and Cyber Security.

Recommended Reading

1. Ben-Moshe S, Kanza Y, Fischer E, Matsliah A, Fischer M, Staelin C (2011) Detecting and exploiting near-sortedness for efficient relational query evaluation. In: ICDT, Uppsala, pp 256–267
2. Berman P, Raskhodnikova S, Yaroslavtsev G (2014) L_p-testing. In: Shmoys DB (ed) STOC, New York. ACM, pp 164–173
3. Bhattacharyya A, Grigorescu E, Jung K, Raskhodnikova S, Woodruff DP (2012) Transitive-closure spanners. SIAM J Comput 41(6):1380–1425
4. Blais E, Raskhodnikova S, Yaroslavtsev G (2014) Lower bounds for testing properties of functions over hypergrid domains. In: IEEE 29th conference on computational complexity (CCC) 2014, Vancouver, 11–13 June 2014, pp 309–320
5. Chakrabarty D, Seshadhri C (2013) Optimal bounds for monotonicity and Lipschitz testing over hypercubes and hypergrids. In: STOC, Palo Alto, pp 419–428
6. Dodis Y, Goldreich O, Lehman E, Raskhodnikova S, Ron D, Samorodnitsky A (1999) Improved testing algorithms for monotonicity. In: RANDOM, Berkeley, pp 97–108
7. Ergün F, Kannan S, Kumar R, Rubinfeld R, Viswanathan M (2000) Spot-checkers. J Comput Syst Sci 60(3):717–751
8. Ergün F, Kumar R, Rubinfeld R (2004) Fast approximate probabilistically checkable proofs. Inf Comput 189(2):135–159
9. Fischer E (2004) On the strength of comparisons in property testing. Inf Comput 189(1):107–116
10. Fischer E, Lehman E, Newman I, Raskhodnikova S, Rubinfeld R, Samorodnitsky A (2002) Monotonicity testing over general poset domains. In: STOC, Montreal, pp 474–483
11. Goldreich O, Goldwasser S, Lehman E, Ron D, Samorodnitsky A (2000) Testing monotonicity. Combinatorica 20(3):301–337
12. Raskhodnikova S (2010) Transitive-closure spanners: a survey. In: Goldreich O (ed) Property testing. Lecture notes in computer science, vol 6390. Springer, Berlin, pp 167–196

Testing Juntas and Related Properties of Boolean Functions

Eric Blais
University of Waterloo, Waterloo, ON, Canada

Keywords

Dimension reduction; Juntas; Property testing; Sublinear-time algorithms

Years and Authors of Summarized Original Work

2004; Fischer, Kindler, Ron, Safra, Samorodnitsky
2009; Blais

Problem Definition

Fix positive integers n and k with $n \geq k$. The function $f : \{0,1\}^n \to \{0,1\}$ is a *k-junta* if it depends on at most k of the input coordinates. Formally, f is a k-junta if there exists a set $J \subseteq \{1,2,\ldots,n\}$ of size $|J| \leq k$ such that for all inputs $x, y \in \{0,1\}^n$ that satisfy $x_i = y_i$ for each $i \in J$, we have $f(x) = f(y)$. Juntas play an important role in different areas of computer science. In machine learning, juntas provide an elegant framework for studying the problem of learning with datasets that contain many irrelevant attributes [9, 10]. In the analysis of Boolean functions, they essentially capture the set of functions of low complexity under natural measures such as total influence [19] and noise sensitivity [12].

How efficiently can we distinguish k-juntas from functions that are far from being k-juntas? We can formalize this question in the setting of property testing. Define the distance between two functions $f, g : \{0,1\}^n \to \{0,1\}$ to be the fraction of inputs on which f and g take different values: $\mathrm{dist}(f,g) := \frac{1}{2^n}|\{x \in \{0,1\}^n : f(x) \neq g(x)\}$. When $\mathrm{dist}(f,g) \geq \epsilon$ for every k-junta g, we say that f is *ϵ-far* from being a k-junta; otherwise we say that f is *ϵ-close* to being a k-juntas. An *ϵ-test* for k-juntas is a randomized algorithm that queries the value of $f : \{0,1\}^n \to \{0,1\}$ on some of its inputs and then with probability at least $\frac{2}{3}$

1. accepts if f is a k-junta, and
2. rejects if f is ϵ-far from being a k-junta.

(The algorithm is free to output anything when f is not a k-junta but is ϵ-close to being a k-junta.)

Problem 1 What is the minimum number of queries to $f : \{0,1\}^n \to \{0,1\}$ required to ϵ-test if f is a k-junta?

Key Results

Testing 1-Juntas

One important class of functions related to junta testing is *dictator* functions – the functions $f : \{0,1\}^n \to \{0,1\}$ of the form $f(x) = x_i$ for some $i \in [n]$. Bellare, Goldreich, and Sudan [3], in a work that was stated in terms of testing the *long code* and part of their analysis of probabilistically checkable proofs (PCPs), showed that dictator functions can be ϵ-tested with $O(1/\epsilon)$ queries. (See the ▸ Locally Testable Codes entry for more details.) This result was later extended by Parnas, Ron, and Samorodnitsky [21]. The class of 1-juntas includes dictator functions, their negations (known as *anti-dictator* functions), and the constant functions; using the algorithms in [3, 21], we can test 1-juntas with $O(1/\epsilon)$ queries.

Testing k-Juntas

The first result on testing k-juntas for values $k > 1$ followed from related work on the problem of *learning* juntas. Blum, Hellerstein, and Littlestone [11] introduced an algorithm that queries a k-junta $f : \{0,1\}^n \to \{0,1\}$ on $O(k \log n + k/\epsilon + 2^k)$ inputs and with probability at least $\frac{5}{6}$ returns a k-junta $h : \{0,1\}^n \to \{0,1\}$ such that $\mathrm{dist}(f,h) \leq \epsilon$. Shortly afterward, Goldreich, Goldwasser, and Ron [20] gave a general reduction showing that a proper learning algorithm with query complexity q for a class C of functions can be used to ϵ-test the class C with $q + O(1/\epsilon)$ queries. This result, combined with the Blum–Hellerstein–Littlestone algorithm, shows that k-juntas can be tested with $O(k \log n + 2^k + 1/\epsilon)$ queries.

Fischer, Kindler, Ron, Safra, and Samorodnitsky [18] showed that, remarkably, it is possible to test k-juntas with a number of queries that is *independent* of n. Specifically, they introduced ϵ-tests for k-juntas with query complexity $O(k^2/\epsilon^2)$. This result was sharpened in [4, 5], leading to the following theorem.

Theorem 1 ([5]) *It is possible to ϵ-test if $f : \{0,1\}^n \to \{0,1\}$ is a k-junta with $O(k \log k + k/\epsilon)$ queries.*

Chockler and Gutfreund [16] showed that $\Omega(k)$ queries are required to test k-juntas, so the bound in Theorem 1 is nearly optimal. (See also [4, 7, 13] for related lower bounds.)

Theorem 1 can be generalized to apply to the setting where $X_1, \ldots, X_n$, and Y are arbitrary

finite sets, and we wish to test whether a function $f : X_1 \times \cdots \times X_n \to Y$ is a k-junta. Interestingly, the query complexity of the k-junta test remains unchanged in this general setting as well. See [5] for the details.

Junta-Testing Algorithm

The proof of Theorem 1 contains two main ingredients.

The first ingredient is a simple modification of the Blum–Hellerstein–Littlestone learning algorithm. The original learning algorithm proceeds in two stages: first, the algorithm learns the k relevant coordinates of the junta; then, it queries f for all 2^k different values of the k relevant coordinates. When we test k-juntas, the second stage is unnecessary and can be replaced with a simpler test that checks whether the (at most) k relevant coordinates that have been identified completely determine the value of f or not. With this modification, we obtain an ϵ-test for k-juntas with query complexity $O(k \log n + k/\epsilon)$. Note that this result already yields the desired bound in Theorem 1 when $n = \text{poly}(k)$.

The second ingredient in the proof of Theorem 1 is a dimension reduction argument. Consider a random partition of the n coordinates into $m = \text{poly}(k)$ parts $S_1, \ldots, S_m$. A function $f : \{0,1\}^n \to \{0,1\}$ is isomorphic to a function $f' : X_1 \times \cdots \times X_m \to \{0,1\}$ where $X_i = \{0,1\}^{|S_i|}$. The function f' is defined over a domain with much smaller dimension, and it satisfies two useful properties. First, when f is a k-junta, then so is f'. Second, when f is ϵ-far from k-juntas and $m = \Omega(k^2)$, then with high probability f' is $\frac{\epsilon}{2}$-far from k-juntas as well. The second fact is far from obvious. It was established in [5] using Fourier analysis and in [8] using a combinatorial argument. These two properties let us complete the algorithm for testing k-juntas by applying the modified Blum–Hellerstein–Littlestone algorithm on the function f'. More details on the algorithm itself can be found in the original papers [5, 18] and the survey [6].

Applications

Feature Selection

Feature selection is the general machine learning task of identifying the features (also known as *attributes* or *variables*) in a dataset that suffice to describe the model being studied. This task is formalized within the junta framework as follows: given a function $f : \{0,1\}^n \to \{0,1\}$, the algorithm seeks to identify a set $J \subseteq [n]$ of size $|J| = k$ where (i) k is as small as possible, and (ii) there is a k-junta $h : \{0,1\}^n \to \{0,1\}$ on the set J that is close to f.

The junta testing algorithm can be used to approximate the minimal value of k for which these two conditions can be satisfied. For example, by executing the junta testing algorithm with $k = 1, 2, 4, 8, \ldots$ until it accepts, we obtain the following estimation result.

Corollary 1 *There is an algorithm that, given query access to $f : \{0,1\}^n \to \{0,1\}$, outputs an estimate $\hat{k}$ such that f is ϵ-close to a k-junta and such that f is not an ℓ-junta for any $\ell < k/2$. Furthermore, this algorithm makes $O(k \log k + k/\epsilon)$ queries to f.*

Testing by Implicit Learning

Let $\mathcal{C}$ be any class (i.e., family) of Boolean functions where every function in $\mathcal{C}$ is close to a being a k-junta. Many natural classes of Boolean functions that have been studied in learning theory and computational complexity fall into this framework. For example, functions with bounded, decision tree complexity, DNF complexity, circuit complexity, and sparse polynomial representation all satisfy this condition. (See the) Diakonikolas et al. [17] gave a general result showing that for each of these classes $\mathcal{C}$, we can ϵ-test the property of being in the class $\mathcal{C}$ efficiently. This result has since been sharpened by Chakraborty et al. [14], yielding the following bounds.

Theorem 2 ([14]) *Fix $s > 0$ and $\epsilon > 0$. We can ϵ-test whether $f : \{0,1\}^n \to \{0,1\}$ can be represented by*

1. *a DNF with s terms,*
2. *a size-s Boolean formula,*
3. *an s-sparse polynomial over* $\mathbb{F}_2^n$*, or*
4. *a decision tree of size s*

with $O(s/\epsilon^2 \cdot \text{polylog}(s/\epsilon))$ *queries.*

The proof of Theorem 2 is remarkable in that the ϵ-test algorithm in [14,17] learns the function $f : \{0,1\}^n \to \{0,1\}$ when f is a k-junta, but without identifying which of the k coordinates of f are part of the junta. This technique is called *testing by implicit learning*, and it is obtained by using and building on the junta testing algorithm.

Testing Function Isomorphism

Two functions $f, g : \{0,1\}^n \to \{0,1\}$ are *isomorphic* to each other when they are identical up to relabeling of the input variables. In the *function isomorphism testing* problem, we are given query access to (an unknown function) f and must determine whether it is isomorphic to (the known function) g or whether it is ϵ-far from being so. How many queries to f do we need to perform this task? The answer, it turns out, depends on the choice of the function g. The functions g for which we can test isomorphism to g with a constant number of queries are called *efficiently isomorphism testable*.

Every symmetric function is efficiently isomorphism testable. Using the junta testing algorithm, Fischer et al. [18] showed that for any constant $k \geq 0$, every k-junta is also efficiently isomorphism testable. An important open problem in property testing is to characterize the set of functions that are efficiently isomorphism testable. The state of the art on this question is a recent result – also building on the junta testing algorithm – showing that every partially symmetric function is also efficiently isomorphism testable. A function $f : \{0,1\}^n \to \{0,1\}$ is *k-partially symmetric* if there is a function $g : \{0,1\}^k \times \{0,1,2,\ldots,n\} \to \{0,1\}$ and a mapping $\rho : [k] \to [n]$ such that $f(x) = g(x_{\rho(1)}, \ldots, x_{\rho(k)}, \|x\|)$ where $\|x\| = \sum_i x_i$ the Hamming weight of x.

Theorem 3 ([8, 15]) *For every constant $k \geq 0$, every k-partially symmetric function is efficiently isomorphism testable.*

Open Problems

There are two particularly appealing open problems related to the junta testing problem that are motivated by its application to the feature selection problem.

Distance Approximation

Theorem 1 shows that we can distinguish k-juntas from functions that are ϵ-far from k-juntas with few queries. Can we also approximate the distance of a function to its closest k-junta with a small number of queries?

Problem 2 What is the minimum number of queries to $f : \{0,1\}^n \to \{0,1\}$ required to approximate the distance of f to its closest k-junta within an additive error of $\pm\epsilon$, where $\epsilon \in [0, \frac{1}{2}]$ is a parameter given to the algorithm?

In some cases, property testing algorithms can also be used directly for the corresponding distance approximation problem. This is the case, for example, for the BLR linearity test in the ▸ Linearity Testing/Testing Hadamard Codes chapter. But it is currently not known whether the junta testing algorithms in [18] or [5] can be extended to yield distance approximators or not.

Testing with Random Samples

The query model we have discussed throughout this chapter – where the algorithm is free to query the target function on any input of its choosing – is known as the *membership query model* in machine learning. In some applications, however, we must consider weaker query models where we restrict the queries that the algorithm can make in some ways. Can we also test k-juntas efficiently in restricted query models?

Problem 3 In which restricted query models can we test whether $f : \{0,1\}^n \to \{0,1\}$ is a k-junta

with a number of queries that is asymptotically smaller than the number of queries required to learn k-juntas in the same settings?

Two examples of restricted query models include the passive sampling model (where each query is drawn independently at random from some fixed distribution) and the active query model (where the algorithm can choose its queries from a larger set of inputs drawn from some distribution). Some initial results on this problem can be found in [1, 2].

Cross-References

▸ Linearity Testing/Testing Hadamard Codes

Recommended Reading

1. Alon N, Hod R, Weinstein A (2013) On active and passive testing. arXiv preprint arXiv:13077364
2. Balcan MF, Blais E, Blum A, Yang L (2012) Active property testing. In: IEEE 53rd annual symposium on foundations of computer science (FOCS'12), New Brunswick. IEEE, pp 21–30
3. Bellare M, Goldreich O, Sudan M (1998) Free bits, PCPs, and nonapproximability—towards tight results. SIAM J Comput 27(3):804–915
4. Blais E (2008) Improved bounds for testing juntas. In: Goel A, Jansen K, Rolim JDP, Rubinfeld R (eds) Approximation, randomization and combinatorial optimization. Algorithms and techniques. Springer, Boston, pp 317–330
5. Blais E (2009) Testing juntas nearly optimally. In: Proceedings of the 2009 ACM international symposium on theory of computing (STOC'09). ACM, New York, pp 151–157
6. Blais E (2010) Testing juntas: a brief survey. In: Goldreich O (ed) Property testing – current research and surveys. Springer, Berlin/Heidelberg, pp 32–40
7. Blais E, Brody J, Matulef K (2012) Property testing lower bounds via communication complexity. Comput Complex 21(2):311–358
8. Blais E, Weinstein A, Yoshida Y (2012) Partially symmetric functions are efficiently isomorphism-testable. In: IEEE 53rd annual symposium on foundations of computer science (FOCS'12), New Brunswick, pp 551–560
9. Blum A (1994) Relevant examples and relevant features: thoughts from computational learning theory. In: AAAI fall symposium on 'Relevance', New Orleans
10. Blum A, Langley P (1997) Selection of relevant features and examples in machine learning. Artif Intell 97(2):245–271
11. Blum A, Hellerstein L, Littlestone N (1995) Learning in the presence of finitely or infinitely many irrelevant attributes. J Comput Syst Sci 50(1): 32–40
12. Bourgain J (2002) On the distribution of the fourier spectrum of boolean functions. Isr J Math 131(1):269–276
13. Buhrman H, García-Soriano D, Matsliah A, de Wolf R (2013) The non-adaptive query complexity of testing k-parities. Chic J Theor Comput Sci 2013: Article 6, 11
14. Chakraborty S, García-Soriano D, Matsliah A (2011) Efficient sample extractors for juntas with applications. In: Aceto L, Henzinger M, Sgall J (eds) Automata, languages and programming. Springer, Zurich, pp 545–556
15. Chakraborty S, Fischer E, García-Soriano D, Matsliah A (2012) Junto-symmetric functions, hypergraph isomorphism, and crunching. In: 2012 IEEE 27th conference on computational complexity (CCC'12). IEEE Computer Society, Los Alamitos, pp 148–158
16. Chockler H, Gutfreund D (2004) A lower bound for testing juntas. Inf Process Lett 90(6):301–305
17. Diakonikolas I, Lee HK, Matulef K, Onak K, Rubinfeld R, Servedio RA, Wan A (2007) Testing for concise representations. In: 48th annual IEEE symposium on foundations of computer science (FOCS'07), Providence. IEEE, pp 549–558
18. Fischer E, Kindler G, Ron D, Safra S, Samorodnitsky A (2004) Testing juntas. J Comput System Sci 68(4):753–787
19. Friedgut E (1998) Boolean functions with low average sensitivity depend on few coordinates. Combinatorica 18(1):27–35
20. Goldreich O, Goldwasser S, Ron D (1998) Property testing and its connection to learning and approximation. J ACM 45(4):653–750
21. Parnas M, Ron D, Samorodnitsky A (2002) Testing basic boolean formulae. SIAM J Discret Math 16(1):20–46

Text Indexing

Srinivas Aluru
Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA

Keywords

String indexing

Years and Authors of Summarized Original Work

1993; Manber, Myers

Problem Definition

Text or string data naturally arises in many contexts including document processing, information retrieval, natural and computer language processing, and describing molecular sequences. In broad terms, the goal of text indexing is to design methodologies to store text data so as to significantly improve the speed and performance of answering queries. While text indexing has been studied for a long time, it shot into prominence during the last decade due to the ubiquity of web-based textual data and search engines to explore it, design of digital libraries for archiving human knowledge, and application of string techniques to further understanding of modern biology. Text indexing differs from the typical indexing of keys drawn from an underlying total order – text data can have varying lengths, and queries are often more complex and involve substrings, partial matches, or approximate matches.

Queries on text data are as varied as the diverse array of applications they support. Consequently, numerous methods for text indexing have been developed and this continues to be an active area of research. Text indexing methods can be classified into two categories: (i) methods that are generalizations or adaptations of indexing methods developed for an ordered set of one-dimensional keys, and (ii) methods that are specifically designed for indexing text data. The most classic query in text processing is to find all occurrences of a pattern P in a given text T (or equivalently, in a given collection of strings). Important and practically useful variants of this problem include finding all occurrences of P subject to at most k mismatches, or at most k insertions/deletions/mismatches. The focus in this entry is on these two basic problems and remarks on generalizations of one-dimensional data structures to handle text data.

Key Results

Consider the problem of finding a given pattern P in text T, both strings over alphabet Σ. The case of a collection of strings can be trivially handled by concatenating the strings using a unique end of string symbol, not in Σ, to create text T. It is worth mentioning the special case where T is structured – i.e., T consists of a sequence of words and the pattern P is a word. Consider a total order of characters in Σ. A string (or word) of length k can be viewed as a k-dimensional key and the order on Σ can be naturally extended to lexicographic order between multidimensional keys of variable length. Any one-dimensional search data structure that supports $O(\log n)$ search time can be used to index a collection of strings using lexicographic order such that a string of length k can be searched in $O(k \log n)$ time. This can be considerably improved as below [8]:

Theorem 1 *Consider a data structure on one-dimensional keys that relies on constant-time comparisons among keys (e.g., binary search trees, red-black trees etc.) and the insertion of a key identifies either its predecessor or successor. Let $O(\mathcal{F}(n))$ be the search time of the data structure storing n keys (e.g., $O(\log n)$ for red-black trees). The data structure can be converted to index n strings using O(n) additional space such that the query for a string s can be performed in $O(\mathcal{F}(n))$ time if s is one of the strings indexed, and in $O(\mathcal{F}(n) + |s|)$ otherwise.*

A more practical technique that provides $O(\mathcal{F}(n) + |s|)$ search time for a string s under more restrictions on the underlying one-dimensional data structure is given in [9]. The technique is nevertheless applicable to several classic one-dimensional data structures, in particular binary search trees and its balanced variants. For a collection of strings that share long common prefixes such as IP addresses and XML path strings, a faster search method is described in [5].

When answering a sequence of queries, significant savings can be obtained by promoting

frequently searched strings so that they are among the first to be encountered in a search path through the indexing data structure. Ciriani et al. [4] use self-adjusting skip lists to derive an expected bound for a sequence of queries that matches the information-theoretic lower bound.

Theorem 2 *A collection of n strings of total length N can be indexed in optimal O(N) space so that a sequence of m string queries, say* $s_1,\cdots,s_m$, *can be performed in* $O(\sum_{j=1}^{m}|s_j| + \sum_{i=1}^{n} n_i \log(m/n_i))$ *expected time, where* n_i *is the number of times the ith string is queried.*

Notice that the first additive term is a lower bound for reading the input, and the second additive term is a standard information-theoretic lower bound denoting the entropy of the query sequence. Ciriani et al. also extended the approach to the external memory model, and to the case of dynamic sets of strings. More recently, Ko and Aluru developed a self-adjusting tree layout for dynamic sets of strings in secondary storage that provides optimal number of disk accesses for a sequence of string or substring queries, thus providing a deterministic algorithm that matches the information-theoretic lower bound [4].

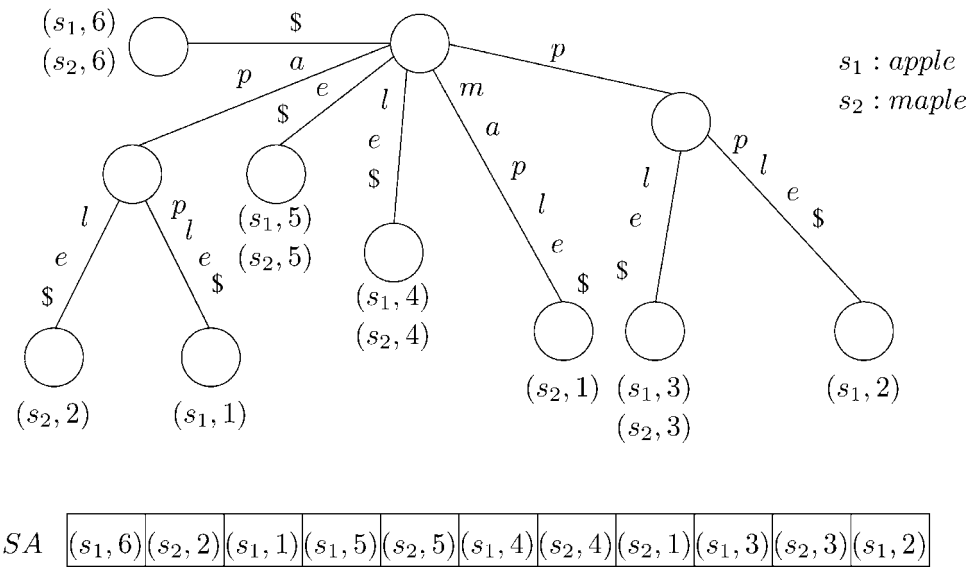
The next part of this entry deals with some of the widely used data structures specifically designed for string data, suffix trees, and suffix arrays. These are particularly suitable for querying unstructured text data, such as the genomic sequence of an organism. The following notation is used: Let $s[i]$ denote the ith character of string s, $s[i \ldots j]$ denote the substring $s[i]s[i+1]\ldots s[j]$, and $S_i = s[i]s[i+1]\ldots s[|s|]$ denote the suffix of s starting at ith position. The suffix S_i can be uniquely described by the integer i. In case of multiple strings, the suffix of a string can be described by a tuple consisting of the string number and the starting position of the suffix within the string. Consider a collection of strings over Σ, having total length n, each extended by adding a unique termination symbol $\$ \notin \Sigma$. The suffix tree of the strings is a compacted trie of all suffixes of these extended strings. The suffix array of the strings is the lexicographic sorted order of all suffixes of these extended strings. For convenience, we list '$', the last suffix of each string, just once. The suffix tree and suffix array of strings 'apple' and 'maple' are shown in Fig. 1. Both these data structures take $O(n)$ space and can be constructed in $O(n)$ time [11, 13], both directly and from each other.

Without loss of generality, consider the problem of searching for a pattern P as a substring of a single string T. Assume the suffix tree ST of T is available. If P occurs in T starting from position i, then P is a prefix of suffix $T_i = T[i]T[i+1]\ldots T[|T|]$ in T. It follows that P matches the path from root to leaf labeled i in ST. This property results in the following simple algorithm: Start from the root of ST and follow the path matching characters in P, until P is completely matched or a mismatch occurs. If P is not fully matched, it does not occur in T. Otherwise, each leaf in the subtree below the matching position gives an occurrence of P. The positions can be enumerated by traversing the subtree in $O(occ)$ time, where occ denotes the number of occurrences of P. If only one occurrence is desired, ST can be preprocessed in $O(|T|)$ time such that each internal node contains the suffix at one of the leaves in its subtree.

Theorem 3 *Given a suffix tree for text T and a pattern P, whether P occurs in T can be answered in* $O(|P|)$ *time. All occurrences of P in T can be found in* $O(|P| + occ)$ *time, where occ denotes the number of occurrences.*

Now consider solving the same problem using the suffix array SA of T. All suffixes prefixed by P appear in consecutive positions in SA. These can be found using binary search in SA. Naively performed, this would take $O(|P| * \log|T|)$ time. It can be improved to $O(|P| + \log|T|)$ time as follows [15]:

Let $SA[L \ldots R]$ denote the range in the suffix array where the binary search is focused. To begin with, $L = 1$ and $R = |T|$. Let $\prec$ denote "lexicographically smaller", $\preceq$ denote "lexicographically smaller or equal", and



Text Indexing, Fig. 1 Suffix tree and suffix array of strings *apple* and *maple*

$lcp(\alpha, \beta)$ denote the length of the longest common prefix between strings α and β. At the beginning of an iteration, $T_{SA[L]} \preceq P \preceq T_{SA[R]}$. Let $M = \lceil (L + R)/2 \rceil$. Let $l = lcp(P, T_{SA[L]})$ and $r = lcp(P, T_{SA[R]})$. Because SA is lexicographically ordered, $lcp(P, T_{SA[M]}) \geq \min(l, r)$. If $l = r$, then compare P and $T_{SA[M]}$ starting from the (l+1)th character. If $l \neq r$, consider the case when $l > r$.

Case I: $l < lcp(T_{SA[L]}, T_{SA[M]})$. In this case, $T_{SA[M]} \prec P$ and $lcp(P, T_{SA[M]}) = lcp(P, T_{SA[L]})$. Continue search in $SA[M \ldots R]$. No character comparisons required.

Case II: $l > lcp(T_{SA[L]}, T_{SA[M]})$. In this case, $P \prec T_{SA[M]}$ and $lcp(P, T_{SA[M]}) = lcp(T_{SA[L]}, T_{SA[M]})$. Continue search in $SA[L \ldots M]$. No character comparisons required.

Case III: $l = lcp(T_{SA[L]}, T_{SA[M]})$. In this case, $lcp(P, T_{SA[M]}) \geq l$. Compare P and $T_{SA[M]}$ beyond lth character to determine their relative order and *lcp*.

Similarly, the case when $r > l$ can be handled such that comparisons between P and $T_{SA[M]}$, if at all needed, start from $(r + 1)$th character. To start the execution of the algorithm, $lcp(P, T_{SA[1]})$ and $lcp(P, T_{SA[|T|]})$ are computed directly using at most $2|P|$ character comparisons. It remains to be described how the $lcp(T_{SA[L]}, T_{SA[M]})$ and $lcp(T_{SA[R]}, T_{SA[M]})$ values required in each iteration are computed. Let $Lcp[1 \ldots |T| - 1]$ be an array such that $Lcp[i] = lcp(SA[i], SA[i + 1])$. The *Lcp* array can be computed from *SA* in $O(|T|)$ time [12]. For any $1 \leq i < j \leq n$, $lcp(T_{SA[i]}, T_{SA[j]}) = \min_{k=i}^{j-1} Lcp[k]$. In order to find the *lcp* values required by the algorithm in constant time, note that the binary search can be viewed as traversing a path in the binary tree corresponding to all possible search intervals used by any execution of the binary search algorithm [15]. The root of the tree denotes the interval $[1 \ldots n]$. If $[i \ldots j]$ $(j - i \geq 2)$ is the interval at an internal node of the tree, its left child is given by $[i \ldots \lceil (i + j)/2 \rceil]$ and its right child is given by $[\lceil (i + j)/2 \rceil \ldots j]$. The *lcp* value for each interval in the tree is precomputed and recorded in $O(n)$ time and space.

Theorem 4 *Given the suffix array SA of text T and a pattern P, the existence of P in T can be checked in* $O(|P| + \log |T|)$ *time. All occurrences of P in T can be found in O(occ) additional time, where occ denotes their number.*

Proof The algorithm makes at most $2|P|$ comparisons in determining $lcp(P, T_{SA[1]})$ and $lcp(P, T_{SA[n]})$. A comparison made in an iteration to determine $lcp(P, T_{SA[M]})$ is categorized *successful* if it contributes the *lcp*, and categorized *failed* otherwise. There is at most one failed comparison per iteration. As for successful comparisons, note that the comparisons start with $(\max(l, r) + 1)^{\text{th}}$ character of P, and each successful comparison increases the value of $\max(l, r)$ for the next iteration. Thus, each character of P is involved only once in a successful comparison. The total number of character comparisons is at most $3|P| + \log|T| = O(|P| + \log|T|)$. □

Abouelhoda et al. [1] reduce this time further to $O(|P|)$ by mimicking the suffix tree algorithm on a suffix array with some auxiliary information. The strategy is useful in other applications based on top-down traversal of suffix trees. At this stage, the distinction between suffix trees and suffix arrays is blurred as the auxiliary information stored makes the combined data structure equivalent to a suffix tree. Using clever implementation techniques, the space is reduced to approximately $6n$ bytes. A major advantage of the suffix tree and suffix array based methods is that the text T is often large and relatively static, while it is queried with several short patterns. With suffix trees and enhanced suffix arrays [1], once the text is preprocessed in $O(|T|)$ time, each pattern can be queried in $O(|P|)$ time for constant size alphabet. For large alphabets, the query can be answered in $O(|P| * \log|\Sigma|)$ time using $O(n|\Sigma|)$ space (by storing an ordered array of $|\Sigma|$ pointers to potential children of a node), or in $O(|P| * |\Sigma|)$ time using $O(n)$ space (by storing pointers to first child and next sibling). (Recently, Cole et al. (2006) showed how to further reduce the search time to $O(|P| + \log|\Sigma|)$ while still keeping the optimal $O(|T|)$ space). For indexing in various text-dynamic situations, see [3, 7] and references therein. The problem of compressing suffix trees and arrays is covered in more detail in other entries.

While exact pattern matching has many useful applications, the need for approximate pattern matching arises in several contexts ranging from information retrieval to finding evolutionary related biomolecular sequences. The classic approximate pattern matching problem is to find substrings in the text T that have an edit distance of k or less to the pattern P, i.e., the substring can be converted to P with at most k insert/delete/substitute operations. This problem is covered in more detail in other entries. Also see [16], the references therein, and Chapter 36 of [2].

Applications

Text indexing has many practical applications – finding words or phrases in documents under preparation, searching text for information retrieval from digital libraries, searching distributed text resources such as the web, processing XML path strings, searching for longest matching prefixes among IP addresses for internet routing, to name just a few. The reader interested in further exploring text indexing is referred to the book by Crochemore and Rytter [6], and to other entries in this Encyclopedia. The last decade of explosive growth in computational biology is aided by the application of string processing techniques to DNA and protein sequence data. String indexing and aggregate queries to uncover mutual relationships between strings are at the heart of important scientific challenges such as sequencing genomes and inferring evolutionary relationships. For an in depth study of such techniques, the reader is referred to Parts I and II of [10] and Parts II and VIII of [2].

Open Problems

Text indexing is a fertile research area, making it impossible to cover many of the research results or actively pursued open problems in a short amount of space. Providing better algorithms and data structures to answer a flow of string-search queries when caches or other query models are taken into account, is an interesting research issue [4].

Cross-References

- Compressed Suffix Array
- Compressed Text Indexing
- Indexed Approximate String Matching
- Indexed Two-Dimensional String Matching
- Suffix Array Construction
- Suffix Tree Construction in Hierarchical Memory
- Suffix Tree Construction

Recommended Reading

1. Abouelhoda M, Kurtz S, Ohlebusch E (2004) Replacing suffix trees with enhanced suffix arrays. J Discret Algorithms 2:53–86
2. Aluru S (ed) (2005) Handbook of computational molecular biology, Computer and Information Science Series. Chapman and Hall/CRC, Boca Raton
3. Amir A, Kopelowitz T, Lewenstein M, Lewenstein N (2005) Towards real-time suffix tree construction. In: Proceedings of the string processing and information retrieval symposium (SPIRE), pp 67–78
4. Ciriani V, Ferragina P, Luccio F, Muthukrishnan S (2007) A data structure for a sequence of string accesses in external memory. ACM Trans Algorithms 3
5. Crescenzi P, Grossi R, Italiano G (2003) Search data structures for skewed strings. In: International workshop on experimental and efficient algorithms (WEA). Lecture notes in computer science, vol 2. Springer, Berlin, pp 81–96
6. Crochemore M, Rytter W (2002) Jewels of stringology. World Scientific Publishing Company, Singapore
7. Ferragina P, Grossi R (1998) Optimal on-line search and sublinear time update in string matching. SIAM J Comput 3:713–736
8. Franceschini G, Grossi R (2004) A general technique for managing strings in comparison-driven data structures. In: Annual international colloquium on automata, languages and programming (ICALP)
9. Grossi R, Italiano G (1999) Efficient techniques for maintaining multidimensional keys in linked data structures. In: Annual international colloquium on automata, languages and programming (ICALP), pp 372–381
10. Gusfield D (1997) Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge University Press, New York
11. Karkkainen J, Sanders P, Burkhardt S (2006) Linear work suffix arrays construction. J ACM 53:918–936
12. Kasai T, Lee G, Arimura H et al (2001) Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Proceedings of the 12th annual symposium, combinatorial pattern matching (CPM), pp 181–192
13. Ko P, Aluru S (2005) Space efficient linear time construction of suffix arrays. J Discret Algorithms 3:143–156
14. Ko P, Aluru S (2007) Optimal self-adjustring tree for dynamic string data in secondary storage. In: Proceedings of the string processing and information retrieval symposium (SPIRE), Santiago. Lecture notes in computer science, vol 4726, pp 184–194
15. Manber U, Myers G (1993) Suffix arrays: a new method for on-line search. SIAM J Comput 22:935–948
16. Navarro G (2001) A guided tour to approximate string matching. ACM Comput Surv 33:31–88

Three-Dimensional Graph Drawing

David R. Wood
School of Mathematical Sciences, Monash University, Melbourne, VIC, Australia

Keywords

Track layout; Three-dimensional straight-line grid drawing; Treewidth

Years and Authors of Summarized Original Work

2005; Dujmović, Morin, Wood

Problem Definition

A *three-dimensional straight-line grid drawing* of a graph, henceforth called a *3D drawing*, represents the vertices by distinct grid-points in $\mathbb{Z}^3$ and represents each edge by the line segment between its end vertices, such that no two edges cross. In contrast to the case in the plane, it is folklore that every graph has a 3D drawing. For example, the "moment curve" algorithm places the ith vertex at (i, i^2, i^3). It is easily seen that no four vertices are coplanar, and thus no two edges cross. Since every graph has a 3D drawing, we are interested in optimizing certain measures of their aesthetic quality. If a 3D drawing is contained in

an axis-aligned box with side lengths $X-1$, $Y-1$, and $Z-1$, then we speak of an $X \times Y \times Z$ drawing with *volume* $X \cdot Y \cdot Z$. This entry considers the problem of producing a 3D drawing of a given graph with small volume.

Key Results

Observe that the drawings produced by the moment curve algorithm have $\mathcal{O}(n^6)$ volume, where n is the number of vertices. Cohen et al. [2] improved this bound, by proving that if p is a prime with $n < p \leq 2n$, and the ith vertex is at $(i, i^2 \bmod p, i^3 \bmod p)$, then there is still no crossing. The resulting $\mathcal{O}(n^3)$ volume bound is optimal for the complete graph K_n since each grid plane may contain at most four vertices. It is therefore of interest to identify fixed graph parameters that allow for 3D drawings with small volume, as summarized in the following table.

Graph family	Min. volume	Reference
Arbitrary	$\Theta(n^3)$	[2]
Bounded chromatic number	$\Theta(n^2)$	[19]
Bounded maximum degree	$\mathcal{O}(n^{3/2})$	[7]
Bounded degeneracy	$\mathcal{O}(n^{3/2})$	[9]
H-minor-free (H fixed)	$n \log^{\mathcal{O}(1)} n$	[12]
Bounded genus	$\mathcal{O}(n \log n)$	[12]
Apex-minor-free	$\mathcal{O}(n \log n)$	[12]
Planar	$\mathcal{O}(n \log n)$	[6]
Bounded treewidth	$\Theta(n)$	[11]

The first such parameter to be studied was the chromatic number. Pach et al. [19] proved that graphs of bounded chromatic number have 3D drawings with $\mathcal{O}(n^2)$ volume. If p is a suitably chosen prime, the main step of their algorithm represents the vertices in the ith color class by grid-points in the set $\{(i, t, it) : t \equiv i^2 \pmod p\}$. It follows that the volume bound is $\mathcal{O}(k^2 n^2)$ for k-colorable graphs.

Pach et al. [19] also proved an $\Omega(n^2)$ lower bound for the volume of 3D drawings of the complete bipartite graph $K_{n,n}$. This lower bound was generalized for all graphs by Bose et al. [1], who proved that every 3D drawing of an n-vertex m-edge graph has volume at least $\frac{1}{8}(n + m)$. In particular, the maximum number of edges in an $X \times Y \times Z$ drawing is exactly $(2X-1)(2Y-1)(2Z-1) - XYZ$.

Graphs with bounded maximum degree have bounded chromatic number and, thus, by the result of Pach et al. [19], have 3D drawings with $\mathcal{O}(n^2)$ volume. Pach et al. [19] conjectured that such graphs have 3D drawings with $o(n^2)$ volume, which was verified by Dujmović and Wood [7], who proved a $\mathcal{O}(n^{3/2})$ bound. The best lower bound is $\Omega(n)$. Determining the optimal volume for 3D drawings of bounded degree graphs is a challenging open problem; see [13]. The $\mathcal{O}(n^{3/2})$ upper bound for bounded degree graphs was generalized for graphs with bounded degeneracy [9].

The first nontrivial $\mathcal{O}(n)$ volume bound was established by Felsner et al. [15] for outerplanar graphs. Their elegant algorithm "wraps" a 2D drawing around a triangular prism to obtain a 3D drawing. This result naturally led to the following open problem due to Felsner et al. [15], which motivated much subsequent research: does every planar graph have a 3D drawing with $\mathcal{O}(n)$ volume?

For some time, the $\mathcal{O}(n^2)$ bound for 2D drawings was the best known bound in 3D. Then Dujmović and Wood [7] proved that every planar graph has a 3D drawing with $\mathcal{O}(n^{3/2})$ volume. A breakthrough came with the $\mathcal{O}(n \log^8 n)$ bound of Di Battista et al. [4], which was improved to $\mathcal{O}(n \log n)$ by Dujmović [6] (with a much simpler proof). The most recent work in this direction, by Dujmović et al. [12], extended this $\mathcal{O}(n \log n)$ bound to all graphs of bounded Euler genus and more generally proved that every graph excluding a fixed minor has a 3D drawing with $n \log^{\mathcal{O}(1)} n$ volume.

The $\mathcal{O}(n)$ volume bound for outerplanar graphs mentioned above was generalized by Dujmović et al. [11] as follows:

Theorem 1 ([11]) *Graphs with bounded treewidth have 3D drawings with $\mathcal{O}(n)$ volume.*

This result is the focus of the remainder of this entry. Treewidth is a measure of the similarity of a graph to a tree. It can be defined as follows. A graph is *chordal* if every induced cycle is a triangle. The *treewidth* of a graph G is the minimum

integer k such that G is a spanning subgraph of a chordal graph with no $(k+2)$-clique. Many graphs arising in applications of graph drawing have small treewidth. Trees have treewidth 1, while outerplanar and series-parallel graphs have treewidth 2. Another example arises in software engineering applications. Thorup [20] proved that the control-flow graphs of go-to free programs in many programming languages have treewidth bounded by a small constant, in particular, 3 for Pascal and 6 for C.

Reference [11] is also important because it discovered the connection between 3D drawings, track layouts, and queue layouts; also see [10, 16].

Track Layouts

Track layouts are a combinatorial tool that effectively eliminates the geometry from 3D drawings and exposes the underlying combinatorial structure. They were introduced in [11] although they are implicit in some previous work [15, 16].

Let $V_1, \ldots, V_t$ be the color classes in a (proper) vertex t-coloring of a graph G. Suppose that each color class V_i is equipped with a total order, denoted by $\preceq$. Call V_i a *track* and $V_1, \ldots, V_t$ a *t-track assignment*. An *X-crossing* in $V_1, \ldots, V_t$ consists of two edges vw and xy such that $v \prec x$ in some track V_i and $y \prec w$ in some other track V_j. A t-track assignment with no X-crossing is called a *t-track layout*.

One can produce a track layout from an $A \times B \times C$ drawing of a graph G as follows. Let $V_{x,y}$ be the set of vertices of G with an X-coordinate of x and a Y-coordinate of y. Order each set $V_{x,y}$ by the corresponding Z-coordinates. We obtain an AB-track layout of G, except that consecutive vertices in each track might be adjacent. Doubling each track and putting alternate vertices in $V_{x,y}$ on distinct tracks gives a $2AB$-track layout of G. Most interestingly, a converse result is also true.

Theorem 2 ([11]) *If an n-vertex graph has a t-track layout, then G has a $\mathcal{O}(t) \times \mathcal{O}(t) \times \mathcal{O}(n)$ drawing with $\mathcal{O}(t^2 n)$ volume.*

The proof of Theorem 2 is inspired by the generalizations of the moment curve algorithm by Cohen et al. [2] and Pach et al. [19]. Loosely speaking, Cohen et al. [2] allow three "free" dimensions, whereas Pach et al. [19] use a coloring to "fix" one dimension with two dimensions free. Theorem 2 uses a track layout to fix two dimensions with one dimension free; see Fig. 1. In particular, say $(V_1, \ldots, V_t)$ is the given t-track layout. Let p be the smallest prime such that $p > k$. Then $p \leq 2k$ by Bertrand's postulate. For $1 \leq i \leq k$, represent the vertices in V_i by the grid-points $\{(i, i^2 \bmod p, t) : 1 \leq t \leq p \cdot |V_i|,\ t \equiv i^3 \pmod p\}$, such that the Z-coordinates respect the given total order of V_i.

Note that Dujmović and Wood [7] combined the method of Pach et al. [19] with the proof of Theorem 2 to conclude a $\mathcal{O}(tn)$ volume bound of 3D drawings of t-track graphs with bounded chromatic number.

As an example of how to construct a track layout, we now show that every tree T has a 3-track layout (which is implicitly proved in [15]). Let r be a vertex of T. Let V_i be the vertices at distance i from r. Note that $(V_0, V_1, \ldots)$ is a coloring of T. Clearly, each color class V_i can be ordered so that there is no X-crossing; see Fig. 2a. Hence $(V_0, V_1, \ldots)$ is a track layout. Note that, working from the root down, the child nodes of each node can be ordered arbitrarily. This will be important later. Now, imagine wrapping this track layout around a prism; see Fig. 2b. That is, for $0 \leq i \leq 2$, group tracks $V_i \prec V_{3+i} \prec V_{6+i} \prec \ldots$ to obtain a 3-track layout of T.

An Algorithm for Graphs of Bounded Treewidth

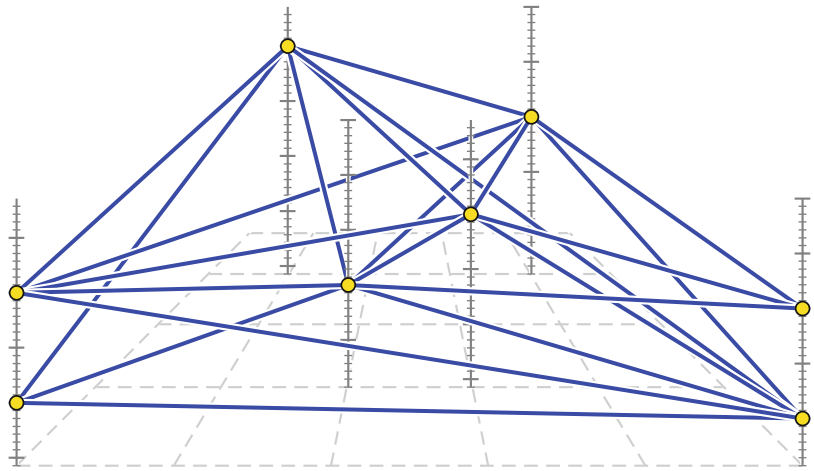
Theorem 1 is an immediate consequence of Theorem 2 and the following claim, which we prove by induction on $k \geq 0$: for each integer $k \geq 0$, there is an integer t_k such that every k-tree has a t_k-track layout. A 0-tree has no edges and thus has a 1-track layout. A 1-tree is a tree which has a 3-track layout. Thus the result holds with $t_0 = 1$ and $t_1 = 3$. Let G be a k-tree. Various authors have proved that G can be decomposed as follows [11, 18]. There is a tree T rooted at some node r and a partition $\{B_x : x \in V(T)\}$ of $V(G)$ indexed by the nodes of T with the following properties:

- For each edge vw of G, there is a node x of T such that $v, w \in B_x$, or there is an edge xy of T such that $v \in B_x$ and $w \in B_y$.
- For each node x of T, the induced subgraph $G[B_x]$ is a $(k-1)$-tree.
- For each non-root node y of T, if x is the parent node of y, and C_y is the set of vertices in B_x adjacent to some vertex in B_y, then C_y is a clique in G called the *parent clique* of y.

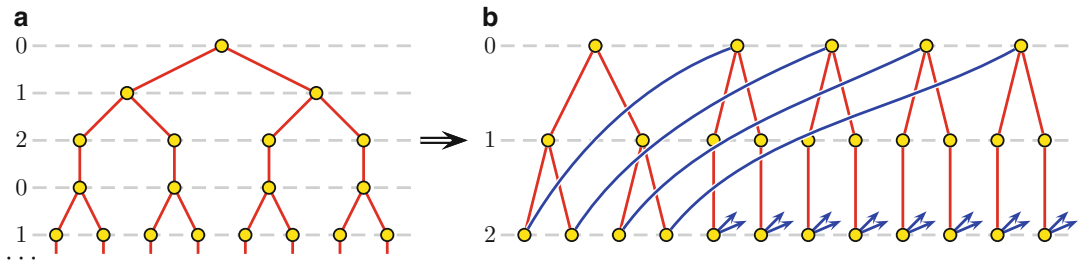
By induction, for each node x of T, there is a t_{k-1}-track layout of $G[B_x]$. Each clique C in $G[B_x]$ has size at most k. Define the *signature* of C to be the set of (at most k) tracks that contain C. Since there is no X-crossing, the set of cliques of $G[B_x]$ with the same signature can be linearly ordered $C_1 \prec \cdots \prec C_p$, such that if v and w are vertices in the same track, and in distinct cliques C_i and C_j with $i < j$, then $v \prec w$ in that track. Call this a *clique ordering*.

Let T_0, T_1, T_2 be a 3-track layout of T described above. Replace each track T_i by t_{k-1} subtracks, and replace each node $x \in T_i$ by the t_{k-1}-track layout of $G[B_x]$. This defines a $3 \cdot t_{k-1}$ track assignment for G. Clearly an edge in some $G[B_x]$ is in no X-crossing with any other edge. There is no X-crossing between two edges between a parent bag B_x and some same child bag B_y, since the end points in B_x of such edges form a clique (the parent clique of y) and therefore are in distinct tracks. The only possible X-crossing is between edges ab and cd, where a and c are in some parent bag B_x and b and d are in distinct child bags B_y and B_z, respectively.

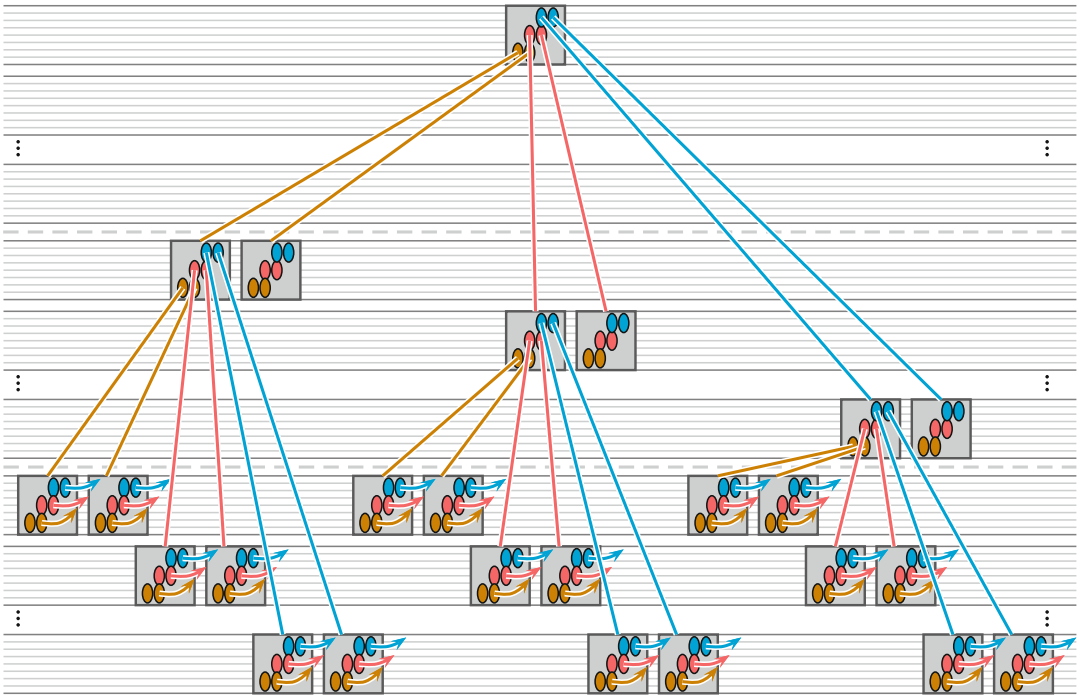
To solve this problem, when determining the 3-track layout of T, the child nodes of each node x are ordered in their track so that $y \prec z$ whenever the parent cliques C_y and C_z have the same signature and $C_y \prec C_z$ in the clique ordering. Then group the child nodes of x according to



Three-Dimensional Graph Drawing, Fig. 1 A 3D drawing produced from a track layout



Three-Dimensional Graph Drawing, Fig. 2 A 3-track layout of a tree



Three-Dimensional Graph Drawing, Fig. 3 Final track layout with $3(t_{k-1})^k$ groups of t_{k-1} tracks

the signatures of their parent cliques, and for each signature σ, use a distinct set of t_{k-1} tracks for the child bags whose parent cliques have signature σ. Now the ordering of the child bags with the same signature agrees with the clique ordering of their parent cliques and therefore agrees with the ordering of any neighbors in the parent bag. It follows that there is no X-crossing, as illustrated in Fig. 3. The number of tracks is at most $3t_{k-1}$ times the number of signatures, which is at most $\sum_{i=1}^{k} \binom{t_{k-1}}{i} \leq (t_{k-1})^k$. This completes the proof with $t_k := 3(t_{k-1})^{k+1}$.

This proof makes no effort to reduce the bound on t_k. The recurrence roughly solves to $3^{(k+2)!}$. The original proof by Dujmović et al. [11] reduces this bound to a doubly exponential function in k. Further improvements were made by Di Giacomo et al. [5], but the bound is still doubly exponential. The best lower bound, due to Dujmović et al. [11], is $\Omega(k^2)$. For $k = 2$, the best upper bound is 15, due to Di Giacomo et al. [5].

Other Models for 3D Graph Drawing

- Polyline grid drawings, where bends in the edges are allowed (at grid-points) [3,8]
- Orthogonal 3D drawings, where the edges are routed along the grid-lines [14,21]
- Upward 3D drawings of directed acyclic graphs [5,9]
- Symmetrical 3D drawings with vertices in $\mathbb{R}^3$ [17]

Recommended Reading

1. Bose P, Czyzowicz J, Morin P, Wood DR (2004) The maximum number of edges in a three-dimensional grid-drawing. J Graph Algorithms Appl 8(1):21–26
2. Cohen RF, Eades P, Lin T, Ruskey F (1996) Three-dimensional graph drawing. Algorithmica 17(2):199–208
3. Devillers O, Everett H, Lazard S, Pentcheva M, Wismath S (2006) Drawing K_n in three dimensions with one bend per edge. J Graph Algorithms Appl 10(2):287–295
4. Di Battista G, Frati F, Pách J (2013) On the queue number of planar graphs. SIAM J Comput 42(6):2243–2285
5. Di Giacomo E, Liotta G, Meijer H, Wismath SK (2009) Volume requirements of 3D upward drawings. Discret Math 309(7):1824–1837
6. Dujmović V (2015) Graph layouts via layered separators. J Comb Theory Ser B 110:79–89

7. Dujmović V, Wood DR (2004) Three-dimensional grid drawings with sub-quadratic volume. In: Pach J (ed) Towards a theory of geometric graphs. Contemporary mathematics, vol 342. American Mathematical Society, Providence, pp 55–66
8. Dujmović V, Wood DR (2005) Stacks, queues and tracks: layouts of graph subdivisions. Discret Math Theor Comput Sci 7:155–202
9. Dujmović V, Wood DR (2006) Upward three-dimensional grid drawings of graphs. Order 23(1): 1–20
10. Dujmović V, Pór A, Wood DR (2004) Track layouts of graphs. Discret Math Theor Comput Sci 6(2):497–522
11. Dujmović V, Morin P, Wood DR (2005) Layout of graphs with bounded tree-width. SIAM J Comput 34(3):553–579
12. Dujmović V, Morin P, Wood DR (2013) Layered separators for queue layouts, 3D graph drawing and nonrepetitive coloring. In: Proceedings of 54th Annual Symposium on Foundations of Computer Science (FOCS '13), Berkeley. IEEE, pp 280–289. http://arxiv.org/abs/1306.1595
13. Dujmović V, Morin P, Sheffer A (2014) Crossings in grid drawings. Electron J Combin 21(1):#P1.41
14. Eades P, Symvonis A, Whitesides S (2000) Three dimensional orthogonal graph drawing algorithms. Discret Appl Math 103:55–87
15. Felsner S, Liotta G, Wismath SK (2003) Straight-line drawings on restricted integer grids in two and three dimensions. J Graph Algorithms Appl 7(4):363–398
16. Heath LS, Leighton FT, Rosenberg AL (1992) Comparing queues and stacks as mechanisms for laying out graphs. SIAM J Discret Math 5(3):398–412
17. Hong SH, Eades P (2003) Drawing trees symmetrically in three dimensions. Algorithmica 36(2):153–178
18. Kündgen A, Pelsmajer MJ (2008) Nonrepetitive colorings of graphs of bounded tree-width. Discret Math 308(19):4473–4478
19. Pach J, Thiele T, Tóth G (1999) Three-dimensional grid drawings of graphs. In: Chazelle B, Goodman JE, Pollack R (eds) Advances in discrete and computational geometry. Contemporary mathematics, vol 223. American Mathematical Society, Providence, pp 251–255
20. Thorup M (1998) All structured programs have small tree-width and good register allocation. Inf Comput 142(2):159–181
21. Wood DR (2003) Optimal three-dimensional orthogonal graph drawing in the general position model. Theor Comput Sci 299(1–3):151–178

Thresholds of Random k-SAT

Alexis Kaporis[1] and Lefteris Kirousis[2]
[1]Department of Information and Communication Systems Engineering, University of the Aegean, Karlovasi, Samos, Greece
[2]Department of Computer Engineering and Informatics, University of Patras, Patras, Greece

Keywords

Phase transitions; Probabilistic analysis of a Davis-Putnam heuristic

Years and Authors of Summarized Original Work

2002; Kaporis, Kirousis, Lalas

Problem Definition

Consider n Boolean variables $V = \{x_1, \ldots, x_n\}$ and the corresponding set of $2n$ literals $L = \{x_1\overline{x}_1 \ldots, x_n, \overline{x}_n\}$. A k-clause is a disjunction of k literals of distinct underlying variables. A random formula $\phi_{n,m}$ in k conjunctive normal form (k-CNF) is the conjunction of m clauses, each selected in a uniformly random and independent way among the $2^k \binom{n}{k}$ possible k-clauses on n variables in V. The density r_k of a k-CNF formula $\phi_{n,m}$ is the clauses-to-variables ratio m/n.

It was conjectured that for each $k \geq 2$ there exists a critical density ${r_k}^*$ such that asymptotically almost all (a.a.a.) k-CNF formulas with density $r < {r_k}^*$ ($r > {r_k}^*$) are satisfiable (unsatisfiable, respectively). So far, the conjecture has been proved only for $k = 2$ [3, 11]. For $k \geq 3$, the conjecture still remains open but is supported by experimental evidence [14] as well as by theoretical, but non-rigorous, work based on statistical physics [15]. The value of the putative threshold ${r_3}^*$ is estimated to be around 4.27. Approximate values of the putative threshold for larger values of k have also been computed.

As far as rigorous results are concerned, Friedgut [10] proved that for each $k \geq 3$, there exists a sequence $r_k^*(n)$ such that for any $\epsilon > 0$, a.a.a. k-CNF formulas ϕ_n, $\lfloor (r_k^*(n) - \epsilon)n \rfloor$ (ϕ_n, $\lfloor (r_k^*(n) + \epsilon)n \rfloor$) are satisfiable (unsatisfiable, respectively). The convergence of the sequence $r_k^*(n), n = 0, 1, \ldots$ for $k \geq 3$ remains open.

Let now

$$r_k^{*-} = \underline{\lim}_{n\to\infty} r_k^*(n)$$
$$= \sup\{r_k : \Pr[\phi_{n,\lfloor r_k n \rfloor} \text{ is satisfiable} \to 1]\}$$

and

$$r_k^{*+} = \overline{\lim}_{n\to\infty} r_k^*(n)$$
$$= \inf\{r_k : \Pr[\phi_{n,\lceil r_k n \rceil} \text{ is satisfiable} \to 0]\}.$$

Obviously, $r_k{}^{*-} \leq r_k{}^{*+}$. Bounding from below (from above) $r_k{}^{*-}$ ($r_k{}^{*+}$, respectively) with an as large as possible (as small as possible, respectively) bound has been the subject of intense research work in the past decade.

Upper bounds to $r_k{}^{*+}$ are computed by counting arguments. To be specific, the standard technique is to compute the expected number of satisfying truth assignments of a random formula with density r_k and find an as small as possible value of r_k for which this expected value approaches zero. Then, by Markov's inequality, it follows that for such a value of r_k, a random formula $\phi_{n,\lceil rkn \rceil}$ is unsatisfiable asymptotically almost always. This argument has been refined in two directions: First, consider not all satisfying truth assignments but a subclass of them with the property that a satisfiable formula always has a satisfying truth assignment in the subclass considered. The restriction to a judiciously chosen such subclass forces the expected value of the number of satisfying truth assignments to get closer to the probability of satisfiability and thus leads to a better (smaller) upper bound r_k. However, it is important that the subclass should be such that the expected value of the number of satisfying truth assignments can be computable by the available probabilistic techniques.

Second, make use in the computation of the expected number of satisfying truth assignments of *typical* characteristics of the random formula, i.e., characteristics shared by a.a.a. formulas. Again this often leads to an expected number of satisfying truth assignments that is closer to the probability of satisfiability (nontypical formulas may contribute to the increase of the expected number). Increasingly better upper bounds to $r_3{}^{*+}$ have been computed using counting arguments as above (see the surveys [6, 13]). Dubois, Boufkhad, and Mandler [7] proved $r_3{}^{*+} < 4.506$. The latter remains the best upper bound to date.

On the other hand, for fixed and small values of k (especially for $k = 3$), lower bounds to $r_k{}^{*-}$ are usually computed by algorithmic methods. To be specific, one designs an algorithm that for an as large as possible r_k it returns a satisfying truth assignment for a.a.a. formulas $\phi_{n,\lfloor rkn \rfloor}$. Such an r_k is obviously a lower bound to $r_k{}^{*-}$. The simpler the algorithm, the easier to perform the probabilistic analysis of returning a satisfying truth assignment for a given r_k, but the smaller the r_k's for which a satisfying truth assignment is returned asymptotically almost always. In this context, backtrack-free DPLL algorithms [4, 5] of increasing sophistication were rigorously analyzed (see the surveys [1, 9]). At each step of such an algorithm, a literal is set to TRUE and then a *reduced* formula is obtained by (i) deleting clauses where this literal appears and by (ii) deleting the negation of this literal from the clauses it appears. At steps at which 1-clauses exist (known as forced steps), the selection of the literal to be set to TRUE is made so as a 1-clause becomes satisfied. At the remaining steps (known as free steps), the selection of the literal to be set to TRUE is made according to a heuristic that characterizes the particular DPLL algorithm. A free step is followed by a round of consecutive forced steps. To facilitate the probabilistic analysis of DPLL algorithms, it is assumed that they never backtrack: if the algorithm ever hits a contradiction, i.e., a 0-clause is generated, it stops and reports failure; otherwise, it returns a satisfying truth assignment. The previously best lower bound for the satisfiability threshold obtained by such an analysis was $3.26 < r_3{}^{*-}$ [2].

The previously analyzed such algorithms (with the exception of the *Pure Literal* algorithm [8]) at a free step take into account only the clause size where the selected literal appears. Due to this limited information exploited on selecting the literal to be set, the reduced formula in each step remains random conditional only on the current numbers of 3- and 2-clauses and the number of yet unassigned variables. This retention of "strong" randomness permits a successful probabilistic analysis of the algorithm in a not very complicated way. However, for $k = 3$, it succeeds to show satisfiability only for densities up to a number slightly larger than 3.26. In particular, in [2] it is shown that this is the optimal value that can be attained by such algorithms.

Key Results

In [12], a DPLL algorithm is described (and then probabilistically analyzed) such that each free step selects the literal to be set to TRUE, taking into account its *degree* (i.e., its number of occurrences) in the current formula.

Algorithm *Greedy*

The first variant of the algorithm is very simple: At each free step, a literal with the maximum number of occurrences is selected and set to TRUE (Section 4.A in [12]). Notice that in this greedy variant, a literal is selected irrespectively of the number of occurrences of its negation. This algorithm successfully returns a satisfying truth assignment for a.a.a. formulas with density up to a number slightly larger than 3.42, establishing that ${r_3}^{*-} > 3.42$. Its simplicity, contrasted with the improvement over the previously obtained lower bounds, suggests the importance of analyzing heuristics that take into account degree information of the current formula.

Algorithm *CL*

In the second variant, at each free step t, the degree of the negation $\overline{\tau}$ of the literal τ that is set to TRUE is also taken into account (Section 5.A in [12]). Specifically, the literal to be set to TRUE is selected so as upon the completion of the round of forced steps that follow the free step t, the marginal expected increase of the flow from 2-clauses to 1-clauses per unit of expected decrease of the flow from 3-clauses to 2-clauses is minimized. The marginal expectation corresponding to each literal can be computed from the numbers of its positive and negative occurrences. More specifically, if $m_i, i = 2, 3$ equals the expected flow of i-clauses to $(i-1)$-clauses at each step of a round, and τ is the literal set to TRUE at the beginning of the round, then τ is chosen so as to minimize the ratio $|\frac{\Delta m_2}{\Delta m_3}|$ of the differences Δm_2 and Δm_3 between the beginning and the end of the round. This has an effect to the bounding of the rate of generation of 1-clauses by the smallest possible number throughout the algorithm. For the probabilistic analysis to go through, we need to know for each i, j the number of literals with degree i whose negation has degree j. This heuristic succeeds in returning a satisfying truth assignment for a.a.a. formulas with density up to a number slightly larger than 3.52, establishing that ${r_3}^{*-} > 3.52$.

Applications

Some applications of SAT solvers include sequential circuit verification, artificial intelligence, automated deduction and planning, VLSI, CAD, model-checking, and other types of formal verification. Recently, automatic SAT-based model-checking techniques were used to effectively find attacks on security protocols.

Open Problems

The main open problem in the area is to formally show the existence of the threshold ${r_k}^*$ for all (or at least some) $k \geq 3$. To rigorously compute upper and lower bounds better than the ones mentioned here still attracts some interest. Related results and problems arise in the framework of variants of the satisfiability problem and also the problem of colorability.

Cross-References

- Backtracking Based k-SAT Algorithms
- Exact Algorithms for k SAT Based on Local Search
- Exact Algorithms for Maximum Two-Satisfiability
- Tail Bounds for Occupancy Problems

Recommended Reading

1. Achlioptas D (2001) Lower bounds for random 3-SAT via differential equations. Theor Comput Sci 265(1–2):159–185
2. Achlioptas D, Sorkin GB (2000) Optimal myopic algorithms for random 3-SAT. In: 41st annual symposium on foundations of computer science, Redondo Beach. IEEE Computer Society, Washington, DC, pp 590–600
3. Chvátal V, Reed B (1992) Mick gets some (the odds are on his side). In: 33rd annual symposium on foundations of computer science, Pittsburgh. IEEE Computer Society, pp 620–627
4. Davis M, Putnam H (1960) A computing procedure for quantification theory. J Assoc Comput Mach 7(4):201–215
5. Davis M, Logemann G, Loveland D (1962) A machine program for theoremproving. Commun ACM 5:394–397
6. Dubois O (2001) Upper bounds on the satisfiability threshold. Theor Comput Sci 265:187–197
7. Dubois O, Boufkhad Y, Mandler J (2000) Typical random 3-SAT formulae and the satisfiability threshold. In: 11th ACM-SIAM symposium on discrete algorithms, San Francisco. Society for Industrial and Applied Mathematics, pp 126–127
8. Franco J (1984) Probabilistic analysis of the pure literal heuristic for the satisfiability problem. Ann Oper Res 1:273–289
9. Franco J (2001) Results related to threshold phenomena research in satisfiability: lower bounds. Theor Comput Sci 265:147–157
10. Friedgut E (1997) Sharp thresholds of graph properties, and the k-SAT problem. J AMS 12:1017–1054
11. Goerdt A (1996) A threshold for unsatisfiability. J Comput Syst Sci 33:469–486
12. Kaporis AC, Kirousis LM, Lalas EG (2006) The probabilistic analysis of a greedy satisfiability algorithm. Random Struct Algorithms 28(4):444–480
13. Kirousis L, Stamatiou Y, Zito M (2006) The unsatisfiability threshold conjecture: the techniques behind upper bound improvements. In: Percus A, Istrate G, Moore C (eds) Computational complexity and statistical physics. Santa Fe Institute studies in the sciences of complexity. Oxford University Press, New York, pp 159–178
14. Mitchell D, Selman B, Levesque H (1992) Hard and easy distribution of SAT problems. In: 10th national conference on artificial intelligence, San Jose. AAAI Press, Menlo Park, pp 459–465
15. Monasson R, Zecchina R (1997) Statistical mechanics of the random k-SAT problem. Phys Rev E 56:1357–1361

Topology Approach in Distributed Computing

Maurice Herlihy
Department of Computer Science, Brown University, Providence, RI, USA

Keywords

Wait-free renaming

Years and Authors of Summarized Original Work

1999; Herlihy Shavit

Problem Definition

The application of techniques from Combinatorial and Algebraic Topology has been successful at solving a number of problems in distributed computing. In 1993, three independent teams [3, 15, 17], using different ways of generalizing the classical graph-theoretical model of distributed computing, were able to solve *set agreement* a long-standing open problem that had eluded the standard approaches. Later on, in 2004, journal articles by Herlihy and Shavit [15] and by Saks and Zaharoglou [17] were to win the prestigious Gödel prize. This paper describes the approach taken by the Herlihy/Shavit paper, which was the first draw the connection between Algebraic and Combinatorial Topology and Distributed Computing.

Pioneering work in this area, such as by Biran, Moran, and Zaks [2] used graph-theoretic notions to model uncertainty, and were able to express

certain lower bounds in terms of graph connectivity. This approach, however, had limitations. In particular, it proved difficult to capture the effects of multiple failures or to analyze decision problems other then consensus.

Combinatorial topology generalizes the notion of a graph to the notion of a *simplicial complex*, a structure that has been well-studied in mainstream mathematics for over a century. One property of central interest to topologists is whether a simplicial complex has no "holes" below a certain dimension k, a property known as *k-connectivity*. Lower bounds previously expressed in terms of connectivity of graphs can be generalized by recasting them in terms of k-connectivity of simplicial complexes. By exploiting this insight, it was possible to solve some open problems (k-set agreement, renaming), to pose and solve some new problems ([13]), and to unify a number of disparate results and models [14].

Key Results

A *vertex* $\vec{v}$ is a point in a high-dimensional Euclidean space. Vertexes $\vec{v}_0, \ldots, \vec{v}_n$ are *affinely independent* if $\vec{v}_1 - \vec{v}_0, \ldots, \vec{v}_n - \vec{v}_0$ are linearly independent. An *n-dimensional simplex* (or *n-simplex*) $S^n = (\vec{s}_0, \ldots, \vec{s}_n)$ is the convex hull of a set of $n+1$ affinely-independent vertexes. For example, a 0-simplex is a vertex, a 1-simplex a line segment, a 2-simplex a solid triangle, and a 3-simplex a solid tetrahedron. Where convenient, superscripts indicate dimensions of simplexes. The $\vec{s}_0, \ldots, \vec{s}_n$ are said to *span* S^n. By convention, a simplex of dimension $d < 0$ is an empty simplex.

A *simplicial complex* (or complex) is a set of simplexes closed under containment and intersection. The *dimension* of a complex is the highest dimension of any of its simplexes. $\mathcal{L}$ is a *subcomplex* of $\mathcal{K}$ if every simplex of $\mathcal{L}$ is a simplex of $\mathcal{K}$. A map $\mu: \mathcal{K} \to \mathcal{L}$ carrying vertexes to vertexes is *simplicial* if it also induces a map of simplexes to simplexes.

Definition 1 A complex $\mathcal{K}$ is *k-connected* if every continuous map of the k-sphere to $\mathcal{K}$ can be extended to a continuous map of the $(k+1)$-disk. By convention, a complex is (-1) *-connected* if and only if it is nonempty, and every complex is *k-connected* for $k < -1$.

A complex is 0-connected if it is connected in the graph-theoretic sense, and a complex is k-connected if it has no holes in dimensions k or less. The definition of k-connectivity may appear difficult to use, but fortunately reasoning about connectivity can be done in a combinatorial way, using the following elementary consequence of the Mayer–Vietoris sequence.

Theorem 2 *If $\mathcal{K}$ and $\mathcal{L}$ are complexes such that $\mathcal{K}$ and $\mathcal{L}$ are k-connected, and $\mathcal{K} \cap \mathcal{L}$ is $(k-1)$-connected, then $\mathcal{K} \cup \mathcal{L}$ is k-connected.*

This theorem, plus the observation that any non-empty simplex is k-connected for all k, allows reasoning about a complex's connectivity inductively in terms of the connectivity of its components.

A set of $n+1$ sequential *processes* communicate either by sending messages to one another or by applying operations to shared objects. At any point, a process may *crash*: it stops and takes no more steps. There is a bound f on the number of processes that can fail. Models differ in their assumptions about timing. At one end of the spectrum is the *synchronous model* in which computation proceeds in a sequence of rounds. In each round, a process sends messages to the other processes, receives the messages sent to it by the other processes in that round, and changes state. (Or it applies operations to shared objects.) All processes take steps at exactly the same rate, and all messages are delivered with exactly the same message delivery time. At the other end is the *asynchronous model* in which there is no bound on the amount of time that can elapse between process steps, and there is no bound on the time it can take for a message to be delivered. Between these extremes is the *semi-synchronous model* in which process step times and message delivery times can vary, but are bounded between constant upper and lower bounds. Proving a lower bound in any of these models requires a deep

understanding of the global states that can arise in the course of a protocol's execution, and of how these global states are related.

Each process starts with an *input value* taken from a set V, and then executes a deterministic *protocol* in which it repeatedly receives one or more messages, changes its local state, and sends one or more messages. After a finite number of steps, each process chooses a *decision value* and halts.

In the k-set agreement task [5], processes are required to (1) choose a decision value after a finite number of steps, (2) choose as their decision values some process's input value, and (3) collectively choose no more than k distinct decision values. When $k = 1$, this problem is usually called *consensus* [16].

Here is the connection between topological models and computation. An initial local state of process P is modeled as a vertex $\vec{v} = \langle P, v \rangle$ labeled with P's process id and initial value v. An initial global state is modeled as an n-simplex $S^n = (\langle P_0, v_0 \rangle, \ldots, \langle P_n, v_n \rangle)$, where the P_i are distinct. The term $ids(S^n)$ denotes the set of process ids associated with S^n, and $vals(S^n)$ the set of values. The set of all possible initial global states forms a complex, called the *input complex*.

Any protocol has an associated *protocol complex* $\mathcal{P}$, defined as follows. Each vertex is labeled with a process id and a possible local state for that process. A set of vertexes $\langle P_0, v_0 \rangle, \ldots, \langle P_d, v_d \rangle$ spans a simplex of $\mathcal{P}$ if and only if there is some protocol execution in which $P_0, \ldots, P_d$ finish the protocol with respective local states $v_0, \ldots, v_d$. Each simplex thus corresponds to an equivalence class of executions that "look the same" to the processes at its vertexes. The term $\mathcal{P}(S^m)$ to denote the subcomplex of $\mathcal{P}$ corresponding to executions in which only the processes in $ids(S^m)$ participate (the rest fail before sending any messages). If $m < n - f$, then there are no such executions, and $\mathcal{P}(S^m)$ is empty. The structure of the protocol complex $\mathcal{P}$ depends both on the protocol and on the timing and failure characteristics of the model. $\mathcal{P}$ often refers to both the protocol and its complex, relying on context to disambiguate.

A protocol *solves* k-set agreement if there is a simplicial map δ, called *decision map*, carrying vertexes of $\mathcal{P}$ to values in V such that if $\vec{p} \in \mathcal{P}(S^n)$ then $\delta(\vec{p}) \in vals(S^n)$, and δ maps the vertexes of any given simplex in $\mathcal{P}(S^n)$ to at most k distinct values.

Applications

The renaming problem is a key tool for understanding the power of various asynchronous models of computation.

Open Problems

Characterizing the full power of the topological approach to proving lower bounds remains an open problem.

Cross-References

▸ Asynchronous Consensus Impossibility
▸ Renaming

Recommended Reading

Perhaps the first paper to investigate the solvability of distributed tasks was the landmark 1985 paper of Fischer, Lynch, and Paterson [6] which showed that *consensus*, then considered an abstraction of the database commitment problem, had no 1-resilient message-passing solution. Other tasks that attracted attention include *renaming* [1, 12, 15] and *set agreement* [3, 5, 12, 10, 15, 17].

In 1988, Biran, Moran, and Zaks [2] gave a graph-theoretic characterization of decision problems that can be solved in the presence of a single failure in a message-passing system. This result was not substantially improved until 1993, when three independent research teams succeeded in applying combinatorial techniques to protocols that tolerate delays by more than one processor: Borowsky and Gafni [3], Saks and Zaharoglou [17], and Herlihy and Shavit [15].

Later, Herlihy and Rajsbaum used homology theory to derive further impossibility results for set agreement and to unify a variety of known impossibility results in terms of the theory of chain maps and chain complexes [12]. Using the same simplicial model.

Biran, Moran, and Zaks [2] gave the first decidability result for decision tasks, showing that tasks are decidable in the 1-resilient message-passing model. Gafni and Koutsoupias [7] were the first to make the important observation that the contractibility problem can be used to prove that tasks are undecidable, and suggest a strategy to reduce a specific wait-free problem for three processes to a contractibility problem. Herlihy and Rajsbaum [11] provide a more extensive collection of decidability results.

Borowsky and Gafni [3], define an iterated immediate snapshot model that has a recursive structure. Chaudhuri, Herlihy, Lynch, and Tuttle [4] give an inductive construction for the synchronous model, and while the resulting "Bermuda Triangle" is visually appealing and an elegant combination of proof techniques from the literature, there is a fair amount of machinery needed in the formal description of the construction. In this sense, the formal presentation of later constructions is substantially more succinct.

More recent work in this area includes separation results [8] and complexity lower bounds [9].

1. Attiya H, Bar-Noy A, Dolev D, Peleg D, Reischuk R (1990) Renaming in an asynchronous environment. J ACM 37(3):524–548
2. Biran O, Moran S, Zaks S (1990) A combinatorial characterization of the distributed 1-solvable tasks. J Algorithms 11(3):420–440
3. Borowsky E, Gafni E (1993) Generalized FLP impossibility result for t-resilient asynchronous computations. In: Proceedings of the 25th ACM symposium on theory of computing, May 1993
4. Chaudhuri S, Herlihy M, Lynch NA, Tuttle MR (2000) Tight bounds for k-set agreement. J ACM 47(5):912–943
5. Chaudhuri S (1993) More choices allow more faults: set consensus problems in totally asynchronous systems. Inf Comput 105(1):132–158, A preliminary version appeared in ACM PODC 1990
6. Fischer MJ, Lynch NA, Paterson MS (1985) Impossibility of distributed consensus with one faulty processor. J ACM 32(2):374–382
7. Gafni E, Koutsoupias E (1999) Three-processor tasks are undecidable. SIAM J Comput 28(3):970–983
8. Gafni E, Rajsbaum S, Herlihy M (2006) Subconsensus tasks: renaming is weaker than set agreement. In: Lecture notes in computer science. pp 329–338
9. Guerraoui R, Herlihy M, Pochon B (2006) A topological treatment of early-deciding set-agreement. OPODIS, pp 20–35
10. Herlihy M, Rajsbaum S (1994) Set consensus using arbitrary objects. In: Proceedings of the 13th annual ACM symposium on principles of distributed computing, Aug 1994, pp 324–333
11. Herlihy M, Rajsbaum S (1997) The decidability of distributed decision tasks (extended abstract). In: STOC '97: proceedings of the twenty-ninth annual ACM symposium on theory of computing. ACM, New York, pp 589–598
12. Herlihy M, Rajsbaum S (2000) Algebraic spans. Math Struct Comput Sci 10(4):549–573
13. Herlihy M, Rajsbaum S (2003) A classification of wait-free loop agreement tasks. Theor Comput Sci 291(1):55–77
14. Herlihy M, Rajsbaum S, Tuttle MR (1998) Unifying synchronous and asynchronous message-passing models. In: PODC '98: proceedings of the seventeenth annual ACM symposium on principles of distributed computing. ACM, New York, pp 133–142
15. Herlihy M, Shavit N (1999) The topological structure of asynchronous computability. J ACM 46(6):858–923
16. Pease M, Shostak R, Lamport L (1980) Reaching agreement in the presence of faults. J ACM 27(2):228–234
17. Saks M, Zaharoglou F (2000) Wait-free k-set agreement is impossible: the topology of public knowledge. SIAM J Comput 29(5):1449–1483

Trade-Offs for Dynamic Graph Problems

Camil Demetrescu[1,2] and Giuseppe F. Italiano[1,2]

[1]Department of Computer and Systems Science, University of Rome, Rome, Italy

[2]Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords

Trading off update time for query time in dynamic graph problems

Years and Authors of Summarized Original Work

2005; Demetrescu, Italiano

Problem Definition

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. A typical definition is given below:

Definition 1 (Dynamic graph algorithm) Given a graph and a graph property $\mathcal{P}$, a *dynamic graph algorithm* is a data structure that supports any intermixed sequence of the following operations:

`insert`(u, v): insert edge (u, v) into the graph.
`delete`(u, v): delete edge (u, v) from the graph.
`query(...)`: answer a query about property $\mathcal{P}$ of the graph.

A graph algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions and *partially dynamic* if it can handle either edge insertions or edge deletions, but not both: it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only. Some papers study variants of the problem where more than one edge can be deleted of inserted at the same time, or edge weights can be changed. In some cases, an update may be the insertion or deletion of a node along with all edges incident to them. Some other papers only deal with specific classes of graphs, e.g., planar graphs, directed acyclic graphs (DAGs), etc.

There is a vast literature on dynamic graph algorithms. Graph problems for which efficient dynamic solutions are known include graph connectivity, minimum cut, minimum spanning tree, transitive closure, and shortest paths (see, e.g., [3] and the references therein). Many of them update explicitly the property $\mathcal{P}$ after each update in order to answer queries in optimal time. This may be a good choice in scenarios where there are few updates and many queries. In applications where the numbers of updates and queries are comparable, a better approach would be to try to reduce the update time, possibly at the price of increasing the query time. This is typically achieved by relaxing the assumption that the property $\mathcal{P}$ should be maintained explicitly.

This entry focuses on algorithms for dynamic graph problems that maintain the graph property implicitly, and thus require non-constant query time while supporting faster updates. In particular, it considers two problems: *dynamic transitive closure* (also known as *dynamic reachability*) and *dynamic all-pairs shortest paths*, defined below.

Definition 2 (Fully dynamic transitive closure) The *fully dynamic transitive closure problem* consists of maintaining a directed graph under an intermixed sequence of the following operations:

`insert`(u, v): insert edge (u, v) into the graph.
`delete`(u, v): delete edge (u, v) from the graph.
`query`(x, y): return *true* if there is a directed path from vertex x to vertex y, and *false* otherwise.

Definition 3 (Fully dynamic all-pairs shortest paths) The *fully dynamic transitive closure problem* consists of maintaining a weighted directed graph under an intermixed sequence of the following operations:

`insert`(u, v): insert edge (u, v) into the graph with weight w.
`delete`(u, v): delete edge (u, v) from the graph.
`query`(x, y): return the distance from x to y in the graph, or $+\infty$ if there is no directed path from x to y.

Recall that the distance from a vertex x to a vertex y is the weight of a minimum-weight path from x

to y, where the weight of a path is defined as the sum of edge weights in the path.

Key Results

This section presents a survey of query/update tradeoffs for dynamic transitive closure and dynamic all-pairs shortest paths.

Dynamic Transitive Closure

The first query/update tradeoff for this problem was devised by Henzinger and King [6], who proved the following result:

Theorem 1 (Henzinger and King 1995 [6]) *Given a general directed graph, there is a randomized algorithm with one-sided error for the fully dynamic transitive closure that supports a worst-case query time of $O(n/\log n)$ and an amortized update time of $O(m\sqrt{n}\log^2 n)$.*

The first subquadratic algorithm for this problem is due to Demetrescu and Italiano for the case of directed acyclic graphs [4, 5]:

Theorem 2 (Demetrescu and Italiano 2000 [4, 5]) *Given a directed acyclic graph with n vertices, there is a randomized algorithm with one-sided error for the fully dynamic transitive closure problem that supports each query in $O(n^\epsilon)$ time and each insertion/deletion in $O(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon})$, for any $\epsilon \in [0,1]$, where $\omega(1,\epsilon,1)$ is the exponent of the multiplication of an $n \times n^\epsilon$ matrix by an $n^\epsilon \times n$ matrix.*

Notice that the dependence of the bounds upon parameter ε leads to a full range of query/update tradeoffs. Balancing the two terms in the update bound of Theorem 2 yields that ε must satisfy the equation $\omega(1,\epsilon,1) = 1 + 2\epsilon$. The current best bounds on $\omega(1,\epsilon,1)$ [2, 7] imply that $\epsilon < 0.575$. Thus, the smallest update time is $O(n^{1.575})$, which gives a query time of $O(n^{0.575})$ (Table 1):

Corollary 1 (Demetrescu and Italiano 2000 [4, 5]) Given a directed acyclic graph with n vertices, there is a randomized algorithm with one-sided error for the fully dynamic transitive closure problem that supports each query in $O(n^{0.575})$ time and each insertion/deletion in $O(n^{1.575})$ time.

This result has been generalized to the case of general directed graphs by Sankowski [13]:

Theorem 3 (Sankowsk 2004 [13]) *Given a general directed graph with n vertices, there is a randomized algorithm with one-sided error for the fully dynamic transitive closure problem that supports each query in $O(n)$ time and each insertion/deletion in $O(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon})$, for any $\epsilon \in [0,1]$, where $\omega(1,\epsilon,1)$ is the exponent of the multiplication of an $n \times n^\epsilon$ matrix by an $n^\epsilon \times n$ matrix.*

Corollary 2 (Sankowski 2004 [13]) *Given a general directed graph with n vertices, there is a randomized algorithm with one-sided error for the fully dynamic transitive closure problem that supports each query in $O(n^{0.575})$ time and each insertion/deletion in $O(n^{1.575})$ time.*

Sankowski has also shown how to achieve an even faster update time of $O(n^{1.495})$ at the expense of a much higher $O(n^{1.495})$ query time:

Theorem 4 (Sankowski 2004 [13]) *Given a general directed graph with n vertices, there is a randomized algorithm with one-sided error for the fully dynamic transitive closure problem that supports each query and each insertion/deletion in $O(n^{1.495})$ time.*

Roditty and Zwick presented algorithms designed to achieve better bounds in the case of sparse graphs:

Theorem 5 (Roditty and Zwick 2002 [10]) *Given a general directed graph with n vertices and m edges, there is a deterministic algorithm for the fully dynamic transitive closure problem that supports each insertion/deletion in $O(m\sqrt{n})$ amortized time and each query in $O(\sqrt{n})$ worst-case time.*

Theorem 6 (Roditty and Zwick 2004 [11]) *Given a general directed graph with n vertices and m edges, there is a deterministic algorithm for the fully dynamic transitive closure problem that supports each insertion/deletion in $O(m + n\log n)$ amortized time and each query in $O(n)$ worst-case time.*

Trade-Offs for Dynamic Graph Problems, Table 1 Fully dynamic transitive closure algorithms with implicit solution representation

Type of graphs	Type of algorithm	Update time	Query time	Reference
General	Monte Carlo	$O(m\sqrt{n}\log^2 n)$*amort*	$O(n/\log n)$	HK [6]
DAG	Monte Carlo	$O(n^{1.575})$	$O(n^{0.575})$	DI [4]
General	Monte Carlo	$O(n^{1.575})$	$O(n^{0.575})$	Sank. [13]
General	Monte Carlo	$O(n^{1.495})$	$O(n^{1.495})$	Sank. [13]
General	Deterministic	$O(m\sqrt{n})$*amort*	$O(\sqrt{n})$	RZ [10]
General	Deterministic	$O(m + n\log n)$*amort*	$O(n)$	RZ [11]

Observe that the results of Theorem 5 and Theorem 6 are subquadratic for $m = o(n^{1.5})$ and $m = o(n^2)$, respectively. Moreover, they are not based on fast matrix multiplication, which is theoretically efficient but impractical.

Dynamic Shortest Paths

The first effective tradeoff algorithm for dynamic shortest paths is due to Roditty and Zwick in the special case of sparse graphs with unit edge weights [12]:

Theorem 7 (Roditty and Zwick 2004 [12]) *Given a general directed graph with n vertices, m edges, and unit edge weights, there is a randomized algorithm with one-sided error for the fully dynamic all-pairs shortest paths problem that supports each distance query in* $O(t + \frac{n\log n}{k})$ *worst-case time and each insertion/deletion in* $O(\frac{mn^2\log n}{t^2} + km + \frac{mn\log n}{k})$ *amortized time.*

By choosing $k = (n\log n)^{1/2}$ and $(n\log n)^{1/2} \leq t \leq n^{3/4}(\log n)^{1/4}$ in Theorem 7, it is possible to obtain an amortized update time of $O(\frac{mn^2\log n}{t^2})$ and a worst-case query time of $O(t)$. The fastest update time of $O(m\sqrt{n\log n})$ is obtained by choosing $t = n^{3/4}(\log n)^{1/4}$.

Later, Sankowski devised the first subquadratic algorithm for dense graphs based on fast matrix multiplication [14]:

Theorem 8 (Sankowski 2005 [14]) *Given a general directed graph with n vertices and unit edge weights, there is a randomized algorithm with one-sided error for the fully dynamic all-pairs shortest paths problem that supports each distance query in* $O(n^{1.288})$ *time and each insertion/deletion in* $O(n^{1.932})$ *time.*

Applications

The transitive closure problem studied in this entry is particularly relevant to the field of databases for supporting transitivity queries on dynamic graphs of relations [16]. The problem also arises in many other areas such as compilers, interactive verification systems, garbage collection, and industrial robotics.

Application scenarios of dynamic shortest paths include network optimization [1], document formatting [8], routing in communication systems, robotics, incremental compilation, traffic information systems [15], and dataflow analysis. A comprehensive review of real-world applications of dynamic shortest path problems appears in [9].

Open Problems

It is a fundamental open problem whether the fully dynamic all pairs shortest paths problem of Definition 3 can be solved in subquadratic time per operation in the case of graphs with real-valued edge weights.

Cross-References

- All Pairs Shortest Paths in Sparse Graphs
- All Pairs Shortest Paths via Matrix Multiplication

- ▶ Decremental All-Pairs Shortest Paths
- ▶ Fully Dynamic All Pairs Shortest Paths
- ▶ Fully Dynamic Transitive Closure
- ▶ Single-Source Fully Dynamic Reachability

Recommended Reading

1. Ahuja R, Magnanti T, Orlin J (1993) Network flows: theory, algorithms and applications. Prentice Hall, Englewood Cliffs
2. Coppersmith D, Winograd S (1990) Matrix multiplication via arithmetic progressions. J Symb Comput 9:251–280
3. Demetrescu C, Finocchi I, Italiano G (2005) Dynamic graphs. In: Mehta D, Sahni S (eds) Handbook on data structures and applications. CRC Press series, in computer and information science, chap. 36. CRC, Boca Raton
4. Demetrescu C, Italiano G (2000) Fully dynamic transitive closure: breaking through the $O(n^2)$ barrier. In: Proceedings of the 41st IEEE annual symposium on foundations of computer science (FOCS'00), Redondo Beach, pp 381–389
5. Demetrescu C, Italiano G (2005) Trade-offs for fully dynamic reachability on dags: breaking through the $O(n2)$ barrier. J ACM 52:147–156
6. Henzinger M, King V (1995) Fully dynamic biconnectivity and transitive closure. In: Proceedings of the 36th IEEE symposium on foundations of computer science (FOCS'95), Milwaukee, pp 664–672
7. Huang X, Pan V (1998) Fast rectangular matrix multiplication and applications. J Complex 14:257–299
8. Knuth D, Plass M (1981) Breaking paragraphs into lines. Softw Pract Exp 11:1119–1184
9. Ramalingam G (1996) Bounded incremental computation. Lecture notes in computer science, vol 1089. Springer, New York
10. Roditty L, Zwick U (2002) Improved dynamic reachability algorithms for directed graphs. In: Proceedings of 43th annual IEEE symposium on foundations of computer science (FOCS), Vancouver, pp 679–688
11. Roditty L, Zwick U (2004) A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In: Proceedings of the 36th annual ACM symposium on theory of computing (STOC), Chicago, pp 184–191
12. Roditty L, Zwick U (2004) On dynamic shortest paths problems. In: Proceedings of the 12th annual European symposium on algorithms (ESA), Bergen, pp 580–591
13. Sankowski P (2004) Dynamic transitive closure via dynamic matrix inverse. In: FOCS '04: proceedings of the 45th annual IEEE symposium on foundations of computer science (FOCS'04). IEEE Computer Society, Washington, DC, pp 509–517
14. Sankowski P (2005) Subquadratic algorithm for dynamic shortest distances. In: 11th annual international conference on computing and combinatorics (COCOON'05), Kunming, pp 461–470
15. Schulz F, Wagner D, Weihe K (1999) Dijkstra's algorithm on-line: an empirical case study from public railroad transport. In: Proceedings of the 3rd workshop on algorithm engineering (WAE'99), London, pp 110–123
16. Yannakakis M (1990) Graph-theoretic methods in database theory. In: Proceedings of the 9-th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, Nashville, pp 230–242

Transactional Memory

Nir Shavit[1,2] and Alexander Matveev[1]
[1]Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA
[2]School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel

Keywords

Atomic operations; Hardware transactional memory; Multiprocessor synchronization; Software transactional memory

Years and Authors of Summarized Original Work

1993; Herlihy, Moss

Problem Definition

A *transactional memory* (TM) is a concurrency control mechanism for executing accesses to memory shared by multiple processes. A *transaction*, in this context, is a section of code that executes a series of reads and writes to the shared memory as one *atomic* indivisible unit. As a result, intermediate states of a transaction are hidden from other concurrent transactions, and it is only possible to see either all of the modifications of a transaction or none of them.

The goal of transactional memory is to provide an alternative to lock-based concurrency control. A programmer can replace the use of lock-based critical sections with transactions and rely on the TM system to execute these sections concurrently while preserving their atomicity. During the execution, the TM system tracks the reads and writes to the shared memory by the different transactions and, in this way, is able to detect conflicts: situations in which transactions are executing operations to the same memory location. Most TM systems are optimistic, executing with the expectation that there will be few conflicts or none. When a conflict is detected, the TM system may have to abort and restart the transaction. The modifications to memory performed by a transaction must thus be reversible.

The concept of transactional memory and a pure hardware implementation of it (HTM) were proposed by Herlihy and Moss [9] in 1993. Two years later, Shavit and Touitou proposed a pure software implementation (STM) [16], and since then HTM and STM systems have been in the focus of intensive research efforts to make them simple and practical for general use. Today's TM systems are not pure hardware or software, but rather a hybrid of HTM and STM.

```
int red_black_tree_contains(node *root, int value) {
    node *cur_node = root;

  __transaction_atomic
  {
     while (cur_node != NULL)
     {
        If (cur_node.value == value) {
           return true;
        }
        If (cur_node.value < value) {
          cur_node = cur_node.left;
        } else {
          cur_node = cur_node.right;
        }
     }
     return false;
  }
}
```

Transactional Memory, Fig. 1 An example of using the GCC TM mechanism to define the red-black tree contains(. . .) operation as a transaction

Key Results

TM C/C++ Specification and Compiler Support

Transactional memory became an industry standard with the addition of transactional language constructs into the C++ specification [1]. The latest GNU C/C++ compiler implements these TM constructs and provides runtime support for state-of-the-art TM algorithms. Figure 1 shows an example of a GCC TM transaction that is defined by using the new *__transaction_atomic* keyword.

HTM in Mainstream Processors

The latest commodity Intel and IBM processors provide support for hardware transactions by leveraging the processor's hardware cache-coherence protocol to track transactional reads and writes and detect conflicts. They unfortunately provide no progress guarantee for hardware transactions: a transaction may fail due to a hardware-related reason (like an L1 cache capacity overflow or an interrupt), and this can happen repeatedly so the transaction may never succeed. To overcome this limitation and provide a progress guarantee, researchers have developed hybrid TM systems [5, 10, 11] that execute failed hardware transactions in an all-software fallback path.

STM Implementations

Software transactions have become much faster and more practical since their introduction by Shavit and Touitou. The state-of-the-art TL2/LSA style STM designs [6, 7] provide software transactions with a guarantee of opacity [8]: the transaction always executes on a consistent memory state. Opacity enables simple STM runtime implementations, since it effectively eliminates the need to detect and handle any runtime errors that could be generated by inconsistent executions.

The TL2/LSA style STMs use a global clock and per object metadata to coordinate transactions, which introduce high constant overheads for reads and writes compared to the pure execution of those reads and writes in hardware. As a result, the TL2/LSA STMs usually perform

well at high concurrency levels, but exhibit poor results for low concurrency. Unlike hardware transactions, they provide a progress guarantee. An alternative design to TL2/LSA is the NORec STM [4] that has no per object metadata and only uses a single global clock to coordinate transactions. The overheads of NORec are very low and operate well at low concurrency levels.

Hardware Lock Elision

Hardware lock elision (HLE) [14] is a mechanism provided by the HTM systems of Intel and IBM and used to optimize lock-based critical sections. The idea of HLE is simple: try to execute the lock-based critical sections concurrently, by using hardware transactions, and if there is a conflict, then fall back to the serial lock-based execution. In this way, the HLE can automatically introduce concurrency into non-conflicting lock-based critical sections, without the need to modify the existing application's code.

Hybrid TM

In order to provide both the performance of hardware and the guarantee of progress of the software implementations, recent TM systems are a hybrid of HTM and STM. A typical hybrid TM first tries to execute transactions in hardware, and if the transaction fails to commit, then it falls back to execute it in software. The key feature of a good hybrid TM is that it provides concurrency between transactions, some of which are executing in hardware and some in software. Recent research shows that it is challenging to provide hardware-software coordination to make hybrid TMs work efficiently [2, 3, 12, 13, 15].

TM Applications

It is still not clear how exactly TM will be used. The intention is that TM will replace the use of locks in application code. Replacing locks in existing code is proving to be a complex task. The main issues arise from the fact that transactions must be able to abort. This means that any side effect or update of a transaction must be reversible, and this constrains the programmer to use only functions that can be undone. The main problem now is that the standard libraries and the C++ STL do not provide full support for TM. Providing such support would be a major step forward toward simple applicability.

The hope going forward is that new programming languages will include transactional memory mechanisms in the language itself and thus allow future code to be written a priori in a transactional fashion without the use of locks.

Cross-References

- ▸ Linearizability
- ▸ Snapshots in Shared Memory

Recommended Reading

1. Adl-Tabatabai A, Shpeisman T, Gottschlich J (2012) Draft specification of transactional language constructs for C++. https://sites.google.com/site/tmforcplusplus
2. Calciu I, Gottschlich J, Shpeisman T, Pokam G, Herlihy M (2014) Invyswell: a hybrid transactional memory for Haswell's restricted transactional memory. In: International conference on parallel architectures and compilation, PACT '14, Edmonton, 24–27 Aug 2014, pp 187–200
3. Dalessandro L, Carouge F, White S, Lev Y, Moir M, Scott ML, Spear MF (2011) Hybrid norec: a case study in the effectiveness of best effort hardware transactional memory. SIGPLAN Not 46(3):39–52
4. Dalessandro L, Spear MF, Scott ML (2010) Norec: streamlining STM by abolishing ownership records. In: Proceedings of the 15th ACM SIGPLAN symposium on principles and practice of parallel programming, PPoPP '10, Bangalore. ACM, New York, pp 67–78
5. Damron P, Fedorova A, Lev Y, Luchangco V, Moir M, Nussbaum D (2006) Hybrid transactional memory. SIGPLAN Not 41(11):336–346
6. Dice D, Shalev O, Shavit N (2006) Transactional locking II. In: Proceedings of the 20th international symposium on distributed computing (DISC 2006), Stockholm, pp 194–208
7. Felber P, Riegel T, Fetzer C (2006) A lazy snapshot algorithm with eager validation. In: 20th international symposium on distributed computing (DISC), Stockholm, Sept 2006.
8. Guerraoui R, Kapalka M (2008) On the correctness of transactional memory. In: Proceedings of the 13th ACM SIGPLAN symposium on principles and practice of parallel programming, PPoPP '08, Salt Lake City. ACM, New York, pp 175–184
9. Herlihy M, Moss E (1993) Transactional memory: architectural support for lock-free data structures. In: Proceedings of the twentieth annual international symposium on computer architecture, San Diego

10. Kumar S, Chu M, Hughes C, Kundu P, Nguyen A (2006, to appear) Hybrid transactional memory. In: Proceedings of the ACM SIGPLAN symposium on principles and practice of parallel programming, PPoPP 2006, New York
11. Lev Y, Moir M, Nussbaum D (2007) PhTM: phased transactional memory. In: Workshop on transactional computing (Transact), 2007. research.sun.com/scalable/pubs/TRANSACT2007PhTM.pdf
12. Matveev A, Shavit N (2013) Reduced hardware transactions: a new approach to hybrid transactional memory. In: 25th ACM symposium on parallelism in algorithms and architectures, SPAA '13, Montreal, pp 11–22
13. Matveev A, Shavit N (2015) Reduced hardware norec: a safe and scalable hybrid transactional memory. In: Proceedings of the twentieth international conference on architectural support for programming languages and operating systems, ASPLOS '15, Istanbul. ACM, New York, pp 59–71
14. Rajwar R, Goodman J (2001) Speculative lock elision: enabling highly concurrent multithreaded execution. In: Proceedings of the 34th annual international symposium on microarchitecture, MICRO, Austin. ACM/IEEE, pp 294–305
15. Riegel T, Marlier P, Nowack M, Felber P, Fetzer C (2011) Optimizing hybrid transactional memory: the importance of nonspeculative operations. In: Proceedings of the 23rd ACM symposium on parallelism in algorithms and architectures, SPAA '11, San Jose. ACM, New York, pp 53–64
16. Shavit N, Touitou D (1997) Software transactional memory. Distrib Comput 10(2):99–116

Traveling Sales Person with Few Inner Points

Yoshio Okamoto
Department of Information and Computer Sciences, Toyohashi University of Technology, Toyohashi, Japan

Keywords

Minimum-cost Hamiltonian circuit problem; Minimum-cost Hamiltonian cycle problem; Minimum-weight Hamiltonian circuit problem; Minimum-weight Hamiltonian cycle problem; Traveling salesman problem; Traveling salesperson problem

Years and Authors of Summarized Original Work

2004; Deĭneko, Hoffmann, Okamoto, Woeginger

Problem Definition

In the *traveling salesman problem* (TSP) n cities 1, 2, …, n together with all the pairwise distances $d(i, j)$ between cities i and j are given. The goal is to find the shortest tour that visits every city exactly once and in the end returns to its starting city. The TSP is one of the most famous problems in combinatorial optimization, and it is well-known to be NP-hard. For more information on the TSP, the reader is referred to the book by Lawler, Lenstra, Rinnooy Kan, and Shmoys [14].

A special case of the TSP is the so-called *Euclidean TSP*, where the cities are points in the Euclidean plane, and the distances are simply the Euclidean distances. A special case of the Euclidean TSP is the *convex Euclidean TSP*, where the cities are further restricted so that they lie in convex position. The Euclidean TSP is still NP-hard [4, 17], but the convex Euclidean TSP is quite easy to solve: Running along the boundary of the convex hull yields a shortest tour. Motivated by these two facts, the following natural question is posed: What is the influence of the number of inner points on the complexity of the problem? Here, an *inner point* of a finite point set P is a point from P which lies in the interior of the convex hull of P. Intuition says that "Fewer inner points make the problem easier to solve."

The result below answers this question and supports the intuition above by providing simple exact algorithms.

Key Results

Theorem 1 *The special case of the Euclidean TSP with few inner points can be solved in the*

following time and space complexity. Here, n denotes the total number of cities and k denotes the number of cities in the interior of the convex hull. 1. In time $O(k!kn)$ *and space* $O(k)$*. 2. In time* $O(2^k k^2 n)$ *and space* $O(2^k kn)$ *[1].*

Here, assume that the convex hull of a given point set is already determined, which can be done in time $O(n \log n)$ and space $O(n)$. Further, note that the above space bounds do not count the space needed to store the input but they just count the space in working memory (as usual in theoretical computer science).

Theorem 1 implies that, from the viewpoint of parameterized complexity [2, 3, 16], these algorithms are fixed-parameter algorithms, when the number k of inner points is taken as a parameter, and hence the problem is fixed-parameter tractable (FPT). (A *fixed-parameter algorithm* has running time $O(f(k)\text{poly}(n))$, where n is the input size, k is a parameter and $f : \mathbb{N} \to \mathbb{N}$ is an arbitrary computable function. For example, an algorithm with running time $O(5^k n)$ is a fixed-parameter algorithm whereas one with $O(n^k)$ is not.) Observe that the second algorithm gives a polynomial-time exact solution to the problem when $k = O(\log n)$.

The method can be extended to some generalized versions of the TSP. For example, Deĭneko et al. [1] stated that the prize-collecting TSP and the partial TSP can be solved in a similar manner.

Applications

The theorem is motivated more from a theoretical side rather than an application side. No real-world application has been assumed.

As for the theoretical application, the viewpoint (introduced in the problem definition section) has been applied to other geometric problems. Some of them are listed below.

The Minimum Weight Triangulation Problem: Given n points in the Euclidean plane, the problem asks to find a triangulation of the points which has minimum total length. The problem is now known to be NP-hard [15]. Hoffmann and Okamoto [10] proved that the problem is fixed-parameter tractable with respect to the number k of inner points. The time complexity they gave is $O(6^k n^5 \log n)$. This is subsequently improved by Grantson, Borgelt, and Levcopoulos [6] to $O(4^k kn^4)$ and by Spillner [18] to $O(2^k kn^3)$. Yet other fixed-parameter algorithms have also been proposed by Grantson, Borgelt, and Levcopoulos [7, 8]. The currently best time complexity was given by Knauer and Spillner [13] and it is $O(2^{c\sqrt{k}\log k} k^{3/2} n^3)$ where $c = (2+\sqrt{2})/(\sqrt{3}-\sqrt{2}) < 11$.

The Minimum Convex Partition Problem: Given n points in the Euclidean plane, the problem asks to find a partition of the convex hull of the points into the minimum number of convex regions having some of the points as vertices.

Grantson and Levcopoulos [9] gave an algorithm running in $O(k^{6k-5} 2^{16k} n)$ time. Later, Spillner [19] improved the time complexity to $O(2^k k^3 n^3)$.

The Minimum Weight Convex Partition Problem: Given n points in the Euclidean plane, the problem asks to find a convex partition of the points with minimum total length.

Grantson [5] gave an algorithm running in $O(k^{6k-5} 2^{16k} n)$ time. Later, Spillner [19] improved the time complexity to $O(2^k k^3 n^3)$.

The Crossing Free Spanning Tree Problem: Given an n-vertex geometric graph (i.e., a graph drawn on the Euclidean plane where every edge is a straight line segment connecting two distinct points), the problem asks to determine whether it has a spanning tree without any crossing of the edges. Jansen and Woeginger [11] proved this problem is NP-hard.

Knauer and Spillner [12] gave algorithms running in $O(175^k k^2 n^3)$ time and $O(2^{33\sqrt{k}\log k} k^2 n^3)$ time.

The method proposed by Knauer and Spillner [12] can be adopted to the TSP as well. According to their result, the currently best time complexity for the TSP is $2^{O(\sqrt{k}\log k)}\text{poly}(n)$.

Open Problems

Currently, no lower bound result for the time complexity seems to be known. For example, is it possible to prove under a reasonable complexity-theoretic assumption the impossibility for the existence of an algorithm running in $2^{O(\sqrt{k})}\text{poly}(n)$ for the TSP?

Cross-References

On the traveling salesman problem:

- ▸ Euclidean Traveling Salesman Problem
- ▸ Hamilton Cycles in Random Intersection Graphs
- ▸ Implementation Challenge for TSP Heuristics
- ▸ Metric TSP

On fixed-parameter algorithms:

- ▸ Closest Substring
- ▸ Parameterized SAT
- ▸ Vertex Cover Kernelization
- ▸ Vertex Cover Search Trees

On others:

- ▸ Minimum Weight Triangulation

Recommended Reading

1. Dĕıneko VG, Hoffmann M, Okamoto Y, Woeginger GJ (2006) The traveling salesman problem with few inner points. Oper Res Lett 31:106–110
2. Downey RG, Fellows MR (1999) Parameterized complexity. Monographs in computer science. Springer, New York
3. Flum J, Grohe M (2006) Parameterized complexity theory. Texts in theoretical computer science an EATCS series. Springer, Berlin
4. Garey MR, Graham RL, Johnson DS (1976) Some NP-complete geometric problems. In: Proceedings of 8th annual ACM symposium on theory of computing (STOC '76). Association for Computing Machinery, New York, pp 10–22
5. Grantson M (2004) Fixed-parameter algorithms and other results for optimal partitions. Lecentiate thesis, Department of Computer Science, Lund University
6. Grantson M, Borgelt C, Levcopoulos C (2005) A fixed parameter algorithm for minimum weight triangulation: analysis and experiments. Technical report 154, Department of Computer Science, Lund University
7. Grantson M, Borgelt C, Levcopoulos C (2005) Minimum weight triangulation by cutting out triangles. In: Deng X, Du D-Z (eds) Proceedings of the 16th annual international symposium on algorithms and computation (ISAAC). Lecture notes in computer science, vol 3827. Springer, New York, pp 984–994
8. Grantson M, Borgelt C, Levcopoulos C (2006) Fixed parameter algorithms for the minimum weight triangulation problem. Technical report 158, Department of Computer Science, Lund University
9. Grantson M, Levcopoulos C (2005) A fixed parameter algorithm for the minimum number convex partition problem. In: Akiyama J, Kano M, Tan X (eds) Proceedings of Japanese conference on discrete and computational geometry (JCDCG 2004). Lecture notes in computer science, vol 3742. Springer, New York, pp 83–94
10. Hoffmann M, Okamoto Y (2006) The minimum weight triangulation problem with few inner points. Comput Geom Theor Appl 34:149–158
11. Jansen K, Woeginger GJ (1993) The complexity of detecting crossingfree configurations in the plane. BIT 33:580–595
12. Knauer C, Spillner A (2006) Fixed-parameter algorithms for finding crossing-free spanning trees in geometric graphs. Technical report 06–07, Department of Computer Science, Friedrich-Schiller-Universität Jena
13. Knauer C, Spillner A (2006) A fixed-parameter algorithm for the minimum weight triangulation problem based on small graph separators. In: Proceedings of the 32nd international workshop on graph-theoretic concepts in computer science (WG). Lecture notes in computer science, vol 4271. Springer, New York, pp 49–57
14. Lawler E, Lenstra J, Rinnooy Kan A, Shmoys D (eds) (1985) The traveling salesman problem: a guided tour of combinatorial optimization. Wiley, Chichester
15. Mulzer W, Rote G (2006) Minimum weight triangulation is NP-hard. In: Proceedings of the 22nd annual ACM symposium on computational geometry (SoCG). Association for Computing Machinery, New York, pp 1–10

16. Niedermeier R (2006) Invitation to fixed-parameter algorithms. Oxford lecture series in mathematics and its applications, vol 31. Oxford University Press, Oxford
17. Papadimitriou CH (1977) The Euclidean travelling salesman problem is NP-complete. Theor Comput Sci 4:237–244
18. Spillner A (2005) A faster algorithm for the minimum weight triangulation problem with few inner points. In: Broersma H, Johnson H, Szeider S (eds) Proceedings of the 1st ACiD workshop. Texts in algorithmics, vol 4. King's College, London, pp 135–146
19. Spillner A (2005) Optimal convex partitions of point sets with few inner points. In: Proceedings of the 17th Canadian conference on computational geometry (CCCG), pp 34–37

Tree Enumeration

Shin-ichi Nakano
Department of Computer Science, Gunma University, Kiryu, Japan

Keywords

Enumeration; Reverse search; Tree generation; Tree listing

Years and Authors of Summarized Original Work

2002; Nakano
2004; Nakano, Uno
2012; Yamanaka, Otachi, Nakano

Problem Definition

A tree is a connected graph with no cycle. A rooted tree is a tree with one designated vertex, called the root. For each vertex v except the root in a rooted tree, the parent of v is the neighbor vertex of v on the path between v and the root. If vertex p is the parent of vertex c, then c is a child of p. An ordered tree is a rooted tree in which the children of each vertex are ordered. The five ordered trees having four vertices are shown in Fig. 1. An unordered tree is a rooted tree in which the ordering of the children of each vertex does not matter. The four ordered trees having four vertices are shown in Fig. 2

Given an integer n the problem of *tree enumeration* asks for generating all ordered (or unordered) trees with n vertices. Several tree generation algorithms are explained in [3] and [2].

Key Results

Tree counting began with Cayley in 1889 to enumerate the saturated hydrocarbons, "C_nH_{2n+2}," which can be modeled as trees.

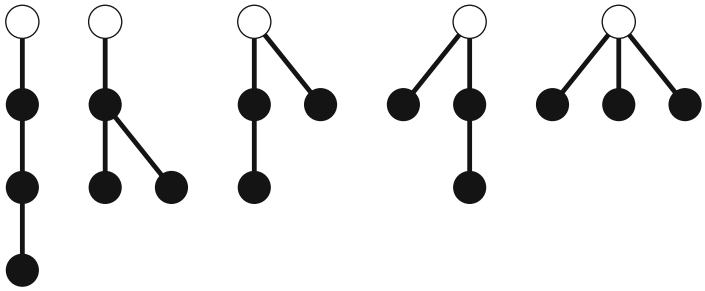
The number of ordered trees with n vertices is C_{n-1} [6], where C_n is the nth Canatal number, defined as follows:

$$C_n = \frac{{}_{2n}C_n}{n+1}$$

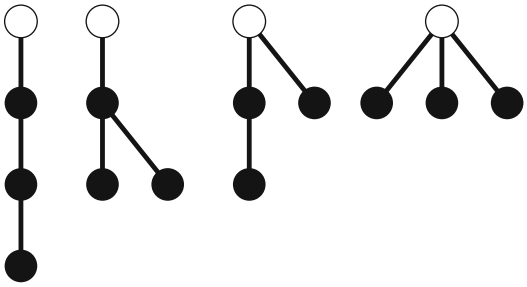
The number of binary trees with n leaves is C_n. No formula for the number of unordered trees with n vertices is known, but the number for $n \leq 40$ is listed at [6, p. 624]. There is a natural one-to-one correspondence between ordered trees with n vertices and binary tree with n leaves [3]. (For each vertex v of an ordered tree if we regard its first child and its next younger sibling as the left child and the right child of v one can have a binary tree in which the root has only one child.) So one can use enumeration algorithm for ordered trees to enumerate binary trees.

Enumeration of All Ordered Trees

Using *reverse search* method [1], one can enumerate all ordered trees with n vertices in $O(1)$ time for each [4]. We sketch the method in [4].



Tree Enumeration, Fig. 1 The ordered trees with four vertices



Tree Enumeration, Fig. 2 The unordered trees with four vertices

Let S_n be the set of all ordered trees with $n > 1$ vertices. Let T be a tree in S_n and $RP = (r_0, r_1, \ldots, r_k)$ be the "rightmost path" of T, which is the path from the root to the rightmost leaf (a leaf is a vertex having no child) such that r_i is the rightmost child of r_{i-1} for each $i = 1, 2, \ldots, k$. Removing the last vertex r_k and the edge attaching to it results in a tree with one less vertices. We repeat such removal of the last vertex of the rightmost path, until the resulting tree consists of exactly one vertex. An example of such repetitive removal is shown in Fig. 3. We call the sequence of ordered trees *the removal sequence* of T. The sequence has n trees and always ends with the tree with exactly one vertex. If we merge the removal sequences of all T in S_n, then we have the (unordered) tree T_n, called *the family tree* of S_n. An example is shown in Fig. 4. Note that T_n has all trees in S_n at its leaves.

The *reverse search* method [1] efficiently traverses the family tree (without storing the family tree in the memory) and output each tree in S_n at each leaf. Thus, we can efficiently enumerate all trees in S_n. The algorithm enumerates all ordered trees with n vertices in $O(1)$ time for each [4].

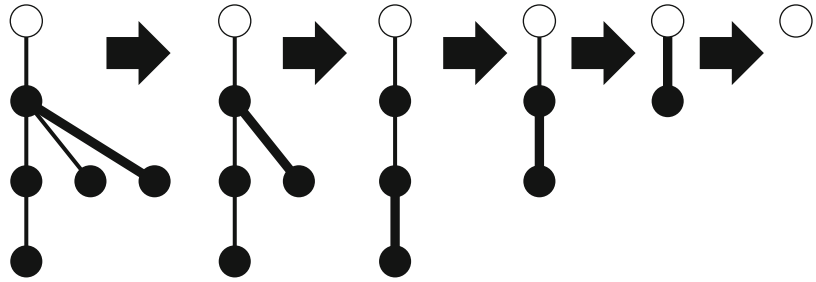
With some additional ideas, given two integers n and k, one can also enumerate all ordered trees with n vertices including k leaves in $O(1)$ time for each [7].

Enumeration of All Unordered Trees

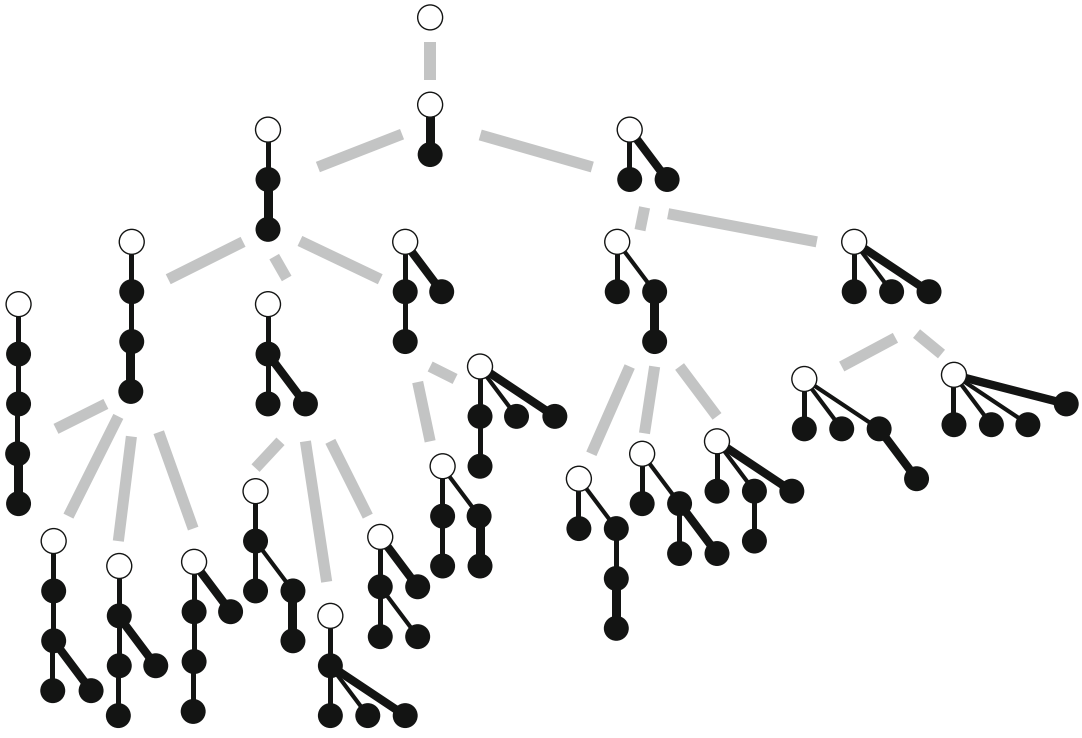
Using a generalized version of the algorithm above, one can also enumerate all unordered trees with n vertices in $O(1)$ time for each [5]. The algorithm generates the next tree in $O(1)$ time using the "prepostorder traversal" technique [3, p. 31]. Since the ordering of the children of each vertex is not fixed we define a "canonical" ordered tree for each unordered tree and define the family tree of the canonical ordered trees. The structure of the family tree is not so simple and this result in a more complicated algorithm.

Cross-References

- Enumeration of Paths, Cycles, and Spanning Trees
- Reverse Search; Enumeration Algorithms



Tree Enumeration, Fig. 3 An example of the removing sequence



Tree Enumeration, Fig. 4 The family tree F_5

Recommended Reading

1. Avis D, Fukuda K (1996) Reverse search for enumeration. Discret Appl Math 65(1–3):21–46
2. Beyer T, Hedetniemi SM (1980) Constant time generation of rooted Trees. SIAM J Comput 9(4):706–712
3. Knuth DE (2006) Generating all trees. The art of computer programming, vol 4, Fascicle4. Addison-Wesley, Upper Saddle River
4. Nakano S (2002) Efficient generation of plane trees. Inf Process Lett 84:167–172
5. Nakano S, Uno T (2004) Constant time generation of trees with specified diameter. In: Proceedings the 30th workshop on graph-theoretic concepts in computer science (WG 2004), Bad Honnef, LNCS, vol 3353, pp 33–45
6. Rosen KH(Ed) (1999) Handbook of discrete and combinatorial mathematics. CRC, Boca Raton
7. Yamanaka K, Otachi Y, Nakano S (2012) Efficient enumeration of ordered trees with k leaves. Theor Comput Sci 442:2–27

Treewidth of Graphs

Hans L. Bodlaender
Department of Computer Science, Utrecht University, Utrecht, The Netherlands

Keywords

Dimension; k-decomposable graphs; Partial k-tree

Years and Authors of Summarized Original Work

1987; Arnborg, Corneil, Proskurowski

Problem Definition

The treewidth of graphs is defined in terms of tree decompositions. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i | i \in I\}, T = (I, F))$ with $\{X_i | i \in I\}$ a collection of subsets of V, called *bags*, and T, a tree, such that

- $O(k\sqrt{\log k})$.
- For all $\{v, w\} \in E$, there is an $i \in I$ with $v, w \in X_i$.
- For all $v \in V$, the set $\{i \in I | v \in X_i\}$ induces a connected subtree of T.

The *width* of a tree decomposition is $\max_{i \in I} |X_i| - 1$, and the treewidth of a graph G is the minimum width of a tree decomposition of G (Fig. 1).

An alternative definition is in terms of chordal graphs. A graph $G = (V, E)$ is *chordal*, if and only if each cycle of length at least 4 has a chord, i.e., an edge between two vertices that are not successive on the cycle. A graph G has treewidth at most k, if and only if G is a subgraph of a chordal graph H that has maximum clique size at most k.

A third alternative definition is in terms of orderings of the vertices. Let π be a permutation (called *elimination scheme* in this context) of the vertices of $G = (V, E)$. Repeat the following step for $i = 1, \ldots, |V|$: take vertex $\pi(i)$, turn the set of its neighbors into a clique, and then remove v. The *width* of π is the maximum over all vertices of its degree when it was eliminated. The treewidth of G equals the minimum width over all elimination schemes.

In the treewidth problem, the given input is an undirected graph $G = (V, E)$, assumed to be given in its adjacency list representation, and a positive integer $k < |V|$. The problem is to decide if G has treewidth at most k and, if so, to give a tree decomposition of G of width at most k.

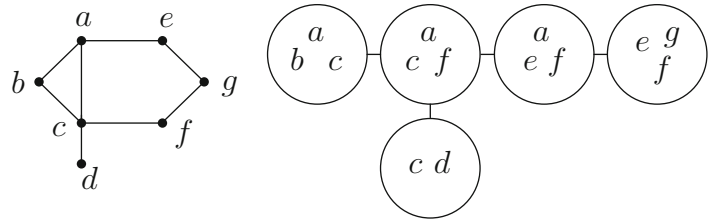
Key Results

Theorem 1 (Arnborg et al. [2]) *The problem, given a graph G and an integer k, is to decide if the treewidth of G of at most k is nondeterministic polynomial-time (NP) complete.*

For many applications of treewidth and tree decompositions, the case where k is assumed to be a fixed constant is very relevant. Arnborg et al. [2] gave in 1987 an algorithm that solves this problem in $O(n^{k+2})$ time. A number of faster algorithms for the problem with k fixed have been found; see, e.g., [6] for an overview.

Theorem 2 (Bodlaender [5]) *For each fixed k, there is an algorithm that, given a graph $G = (V, E)$ and an integer k, decides if the treewidth of G is at most k and, if so, that finds a tree decomposition of width at most k in $O(n)$ time.*

This result of Theorem 2 is of theoretical importance only: in a practical setting, the algorithm appears to be much too slow owing to the large constant factor, hidden in the Onotation. For treewidth 1, the problem is equivalent to recognizing trees. Efficient algorithms based on a small set of reduction rules exist for treewidth 2 and 3 [1].



Treewidth of Graphs, Fig. 1 A graph and a tree decomposition of width 2

Two often-used heuristics for treewidth are the *minimum fill-in* and *minimum degree* heuristic. In the *minimum degree* heuristic, a vertex v of minimum degree is chosen. The graph G', obtained by making the neighborhood of v a clique and then removing v and its incident edges, is built. Recursively, a chordal supergraph H' of G' is made with the heuristic. Then, a chordal supergraph H of G is obtained, by adding v and its incident edges from G to H'. The *minimum fill-in* heuristic works similarly, but now a vertex is selected such that the number of edges that is added to make the neighborhood of v a clique is as small as possible.

Theorem 3 (Fomin and Villanger [11]) *There is an algorithm that, given a graph $G = (V, E)$, determines the treewidth of G and finds a tree decomposition of G of minimum width that uses $O(1.7549^n)$ time.*

Bouchitté and Todinca [10] showed that the treewidth can be computed in polynomial time for graphs that have a polynomial number of minimal separators. This implies polynomial-time algorithms for several classes of graphs, e.g., permutation graphs, weakly triangulated graphs.

Applications

One of the main applications of treewidth and tree decomposition is that many problems that are intractable (e.g., NP-hard) on arbitrary graphs become polynomial time or linear time solvable when restricted to graphs of bounded treewidth. The problems where this technique can be applied include many of the classic graph and network problems, like Hamiltonian circuit, Steiner tree, vertex cover, independent set, and graph coloring, but it can also be applied to many other problems. The technique can sometimes be used for directed graphs [12]. It is also used in the algorithm by Lauritzen and Spiegelhalter [14] to solve the inference problem on probabilistic ("Bayesian," or "belief") networks. Such algorithms typically have the following form. First, a tree decomposition of bounded width is found, and then a dynamic programming algorithm is run that uses this tree decomposition. Often, the running time of this dynamic programming algorithm is exponential in the width of the tree decomposition that is used, and thus one wants to have a tree decomposition whose width is as small as possible.

There are also general characterizations of classes of problems that are solvable in linear time on graphs of bounded treewidth. Most notable is the class of problems that can be formulated in *monadic second-order logic* and extensions of these.

Treewidth has been used in the context of several applications or theoretical studies, including graph minor theory, data bases, constraint satisfaction, frequency assignment, compiler optimization, and electrical networks.

Open Problems

There are polynomial-time approximation algorithms for treewidth that guarantee a width of $\bigcup_{i \in I} X_i = V$ for graphs of treewidth k. Austrin et al. [3] show that there is no constant factor approximation for treewidth under the small set expansion conjecture. A long-standing open problem is whether there is a polynomial-time algorithm to compute the treewidth of planar graphs.

Also open is to find an algorithm for the case where the bound on the treewidth k is fixed and whose running time as a function on n is

polynomial and as a function on k improves significantly on the algorithm of Theorem 2.

The base of the exponent of the running time of the algorithm of Theorem 3 can possibly be improved.

Experimental Results

Many algorithms (upper-bound heuristics, lower-bound heuristics, exact algorithms, and preprocessing methods) for treewidth have been proposed and experimentally evaluated. An overview of many of such results is given in [10]. A variant of the algorithm by Arnborg et al. [2] was implemented by Shoikhet and Geiger [18]. Röhrig [17] has experimentally evaluated the linear-time algorithm of Bodlaender [5] and established that it is not practical, even for small values of k. The *minimum degree* and *minimum fill-in* heuristics are frequently used [13].

Data Sets

A collection of test graphs and results for many of the algorithms on these graphs can be found in the TreewidthLIB collection [7].

Cross-References

▸ Branchwidth of Graphs

Recommended Reading

1. Arnborg S, Proskurowski A (1986) Characterization and recognition of partial 3-trees. SIAM J Algebr Discret Methods 7:305–314
2. Arnborg S, Corneil DG, Proskurowski A (1987) Complexity of finding embeddings in a k-tree. SIAM J Algebr Discret Methods 8:277–284
3. Austrin P, Pitassi T, Wu Y (2012) Inapproximability of treewidth, one-shot pebbling, and related layout problems. In: Gupta A, Jansen K, Rolin JDP, Servedio RA (eds) Proceedings 15th international workshop on approximation, randomization, and combinatorial optimization, APPROX-RANDOM 2012, Cambridge. Lecture notes in computer science, vol 7408. Springer, Berlin, pp 13–24
4. Bodlaender HL (1993) A tourist guide through treewidth. Acta Cybernetica 11:1–23
5. Bodlaender HL (1996) A linear time algorithm for finding tree-decompositions of small treewidth. SIAM J Comput 25:1305–1317
6. Bodlaender HL (1998) A partial k-arboretum of graphs with bounded treewidth. Theor Comput Sci 209:1–45
7. Bodlaender HL (2004) Treewidthlib. http://www.cs.uu.nl/people/hansb/treewidthlib
8. Bodlaender HL (2005) Discovering treewidth. In: Vojtáš P, Bieliková M, Charron-Bost B (eds) Proceedings 31st conference on current trends in theory and practice of computer science, SOFSEM 2005, Liptovský Ján. Lecture notes in computer science, vol 3381. Springer, Berlin, pp 1–16
9. Bodlaender, H.L.: Treewidth: Characterizations, applications, and computations. In: Fomin, F.V. (ed.) Proceedings 32nd International Workshop on Graph-Theoretic Concepts in Computer Science WG'06. Lecture Notes in Computer Science, vol. 4271, pp. 1–14. Springer, Bergen (2006)
10. Bouchitté V, Todinca I (2002) Listing all potential maximal cliques of a graph. Theor Comput Sci 276:17–32
11. Fomin FV, Villanger I (2012) Treewidth computation and extremal combinatorics. Combinatoria 32:289–308
12. Gutin G, Kloks T, Lee CM, Yeo A (2005) Kernels in planar digraphs. J Comput Syst Sci 71:174–184
13. Koster AMCA, Bodlaender HL, van Hoesel SPM (2001) Treewidth: computational experiments. In: Broersma H, Faigle U, Hurink J, Pickl S (eds) Electronic notes in discrete mathematics, vol 8. Elsevier, Amsterdam, pp 54–57
14. Lauritzen SJ, Spiegelhalter DJ (1988) Local computations with probabilities on graphical structures and their application to expert systems. J R Stat Soc Ser B (Methodol) 50:157–224
15. Reed BA (1997) Tree width and tangles, a new measure of connectivity and some applications. LMS lecture note series, vol 241. Cambridge University Press, Cambridge, pp 87–162
16. Reed BA (2003) Algorithmic aspects of tree width. CMS books in mathematics/Ouvrages de mathématiques de la SMC, vol 11. Springer, New York, pp 85–107
17. Röhrig H (1998) Tree decomposition: a feasibility study. Master's thesis, Max-Planck-Institut für Informatik, Saarbrücken
18. Shoikhet K, Geiger D (1997) A practical algorithm for finding optimal triangulations. In: Proceedings of the national conference on artificial intelligence (AAAI '97), Providence. Morgan Kaufmann, San Francisco, pp 185–190

Trial and Error Algorithms

Xiaohui Bei[1], Ning Chen[1], and Shengyu Zhang[2]
[1]Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore
[2]The Chinese University of Hong Kong, Hong Kong, China

Keywords

Algorithm; Complexity; Constraint Satisfaction Problem; Query; Trial and error; Unknown input

Years and Authors of Summarized Original Work

2013(1); Bei, Chen, Zhang
2013(2); Bei, Chen, Zhang

Problem Definition

This problem investigates the effect of the lack of input information on computational hardness. The central question under investigation is the following:

> How much extra difficulty is introduced due to the lack of input knowledge?

We explore this question by studying search problems. Suppose that on an input instance x, there is a set $S(x)$ of *solutions*. A search problem is to find a solution $s \in S(x)$ for the input x. More specifically, we consider the fairly broad class of Constraint Satisfaction Problems (CSPs): Suppose that there is an input space $\{0,1\}^n$ and a space $\Omega = \{0,1\}^m$ of candidate solutions. The problem is defined by a number of constraints $C_1, C_2, \ldots, C_m(,\ldots)$, where each $C_i : \{0,1\}^{n+m} \to \{0,1\}$ is a 0-1 function on the input and solution variables. The valid solutions for input x are defined as those s that satisfy all constraints C_i, i.e., those in $\{s : C_i(x,s) = 1, \forall i\}$. Note that the number of constraints can range from constant to polynomial, exponential, or even infinite. CSPs form a subject with intensive research in theoretical computer science, artificial intelligence, and operations research, and they provide a common basis for exploration of a large number of problems with both theoretical and practical importance.

The standard setting for CSP is to find a solution s on a given input x. Now consider the situation in which the input x is unknown. For a search problem A, denote by A_u A_u the same search problem with unknown inputs. For example, in the StableMatching problem, the input contains the preference lists of all men and women; in $\mathsf{StableMatching}_\mathsf{u}$, these preference lists are unknown to us. The constraints are that all man-woman pairs (m,w) are not blocking pairs, and the task is to find a solution that satisfies all constraints, namely, a stable matching.

The method of searching for a solution of an unknown CSP follows a trial and error approach. Trial and error is a basic methodology in problem solving and knowledge acquisition, and it has also been used extensively in product design and experiments. In our setting for CSPs, an algorithm can propose a candidate solution s. If s is not a valid solution, then we are told so by a *verification oracle* V, and furthermore, V also gives the *index* of one constraint that is not satisfied. If s is a valid solution, i.e., it satisfies all constraints, V returns an affirmative answer, and the problem is solved. Two remarks:

- If more than one constraint is violated, then (the index of) any one of them can be returned by V.
- Note that V does not reveal the constraint itself, but only its index.

Given the verification oracle V, an algorithm is an interactive process with V. The algorithm chooses candidate solutions (i.e., trials), and the oracle returns violations (i.e., errors). The process is adaptive, i.e., a newly proposed solution can be based on the historical information returned by the oracle.

Because the focus is on how much *extra* difficulty is introduced by the lack of input information for a search problem A, we single out this by comparing the unknown-input and known-

input complexities. To this end, the algorithms are equipped with another oracle, the *computation oracle*, which can solve the known-input version of the same problem A. Thus overall, trial-and-error algorithms can access two oracles, the verification oracle and the computation oracle.

The model is motivated from several applications in practice; please see [4] for more discussions.

Time Complexity

As is standard in complexity theory, a query to either oracle has a unit time cost. The *time complexity* of a problem with unknown inputs is the minimum time needed for an algorithm to solve it for all inputs and all verification oracles consistent with the input. The standard notation in computational complexity theory for complexity classes such as **P** and **NP** and also for oracles are employed. For example, $\mathsf{A}_\mathsf{u} \in \mathbf{P}^{\mathsf{V},\mathsf{A}}$ means that problem A_u can be solved by a polynomial-time algorithm with verification oracle V and the computation oracle that can solve the known-input version of A. If this occurs, then one consider the extra complexity (resulting from the unknown input) not to be very high. The central question can therefore be translated to the following. Given a search problem A, is $\mathsf{A}_\mathsf{u} \in \mathbf{P}^{\mathsf{V},\mathsf{A}}$? If the given known-input problem A is in **P**, then the computation oracle can be omitted, and the problem becomes "Is $\mathsf{A}_\mathsf{u} \in \mathbf{P}^{\mathsf{V}}$?"

Trial Complexity

The *trial complexity* of an unknown-input problem A_u is defined as the minimum number of queries to the verification oracle that any algorithm needs to make, regardless of its computational power. As is standard in query complexity theory, one can consider deterministic or (Las Vegas) randomized algorithms. Denote by $\mathsf{D}(A_\mathsf{u})$ and $\mathsf{R}(A_\mathsf{u})$ the deterministic and randomized trial complexities of A_u, respectively.

Key Results

The trial and time complexities of a number of problems are investigated in the trial and error model.

Theorem 1 ([4]) *For the following problems* A*, we have* $\mathsf{A}_\mathsf{u} \in \mathbf{P}^{\mathsf{V},\mathsf{A}}$.

- Nash*: Find a Nash equilibrium of a normal-form game.*
- Core*: Find a core of a cooperative game.*
- StableMatching*: Find a stable matching of a two-sided market with preference lists.*
- SAT*: Find a satisfying assignment of a CNF formula.*

Nash is a fundamental problem in game theory, and its complexity has been characterized as **PPAD**-complete [6, 7]. Core is a fundamental problem in cooperative game theory [10]. Both problems are naturally defined as CSPs. Nash can be formulated as a CSP of finding a pair of mixed strategies, where the constraints are that for each player, for each strategy, adopting that strategy is not better than the current (mixed) strategy. StableMatching is a problem with interesting combinatorial structures and many applications, such as the pairing of graduating medical students with hospital residencies [11, 12]. Formally, given are two sets of elements M and W, each element having a preference list of elements in the other set. The task is to find a matching of the two sets *s.t.* No two unmatched elements (m_i, w_j) both prefer each other to the currently assigned one. In the unknown input version, the preference list of each individual is not known. The algorithm can propose a matching; if it is not stable in the above sense, then a pair (m_i, w_j) not satisfying the above property, sometimes called "blocking pair," is returned. SAT is a natural CSP, with the constraints being the OR of some literals.

Considering the practical significance of StableMatching and SAT, the next theorem takes a closer look at their trial complexities.

Theorem 2 ([4])

- $\Omega(n^2) \le \mathsf{R}(\mathsf{StableMatching}_\mathsf{u}) \le \mathsf{D}(\mathsf{StableMatching}_\mathsf{u}) \le O(n^2 \log n)$*, where* n *is the number of agents.*
- *Given a formula with* n *variables and* m *clauses,* $\mathsf{R}(\mathsf{SAT}_\mathsf{u}) \le \mathsf{D}(\mathsf{SAT}_\mathsf{u}) = O(mn)$.

Further, $\mathsf{R}(\mathsf{SAT}_\mathsf{u}) = \Omega(mn)$ *if* $m = \Omega(n^2)$, *and* $\mathsf{R}(\mathsf{SAT}_\mathsf{u}) = \Omega(m^{3/2})$ *if* $m = o(n^2)$.

It is somewhat surprising that knowing only the indices of violated constraints is already sufficient to admit quite a number of efficient algorithms. It is therefore natural to wonder whether the lack of input information adds any extra difficulty at all in any problem. The answer turns out to be affirmative: there are problems whose unknown-input versions are considerably more difficult than their known versions. Two representatives are GraphIso and GroupIso, the problems of deciding whether two given graphs or groups are isomorphic.

Theorem 3 ([4])

- *If* $\mathsf{GraphIso}_\mathsf{u} \in \mathbf{P}^{\mathsf{V},\mathsf{GraphIso}}$, *then the polynomial hierarchy (***PH***) collapses to the second level.*
- *If* $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)_\mathsf{u} \in \mathbf{P}^{\mathsf{V}}$, *then we have* $\mathbf{P} = \mathbf{NP}$. *(Here,* $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)$ *is the group isomorphism problem with the second group known as* $\mathbb{Z}_p$ *for a prime* p.*) If* $\mathsf{GroupIso}_\mathsf{u} \in \mathbf{P}^{\mathsf{V},\mathsf{GroupIso}}$, *then we have* $\mathbf{NP} \subseteq \mathbf{P}^{O(\log n)}$.

However, if SAT *is given as the computation oracle, then deterministic polynomial-time algorithms exist for* GraphIso *and* GroupIso, *i.e.,* $\mathsf{GraphIso}_\mathsf{u} \in \mathbf{P}^{\mathsf{V},\mathsf{SAT}}$ *and* $\mathsf{GroupIso}_\mathsf{u} \in \mathbf{P}^{\mathsf{V},\mathsf{SAT}}$, *with* $O(n^2)$ *and* $O(n^6)$ *trials, respectively.*

Note that $\mathsf{GroupIso}(\cdot, \mathbb{Z}_p)$ (with a known input) admits a simple polynomial-time algorithm by comparing the multiplication tables. Actually, GroupIso is in **P** if the two groups are Abelian. However, if the multiplication table of the input group is unknown, then surprisingly, the problem becomes **NP**-hard. Putting the computational hardness and the low trial complexity together, one can see that if more computational time (enough to solve an **NP** problem) is given, then less trials are needed. This interesting trade-off between the two complexity measures is not commonly seen in other query models.

Finally, beyond all of the foregoing problems that can be solved in $\mathbf{P}^{\mathsf{V},\mathsf{SAT}}$, one can show via an information theoretical argument that the following two problems have exponential lower bounds for the randomized trial complexity.

- LinearProgramming: Find a feasible solution of a linear program with n variables and m constraints.
- SubsetSum: Decide whether a given set of n integers can be partitioned into two parts with equal summation of elements.

Theorem 4 ([4,5])

- $\mathsf{R}(\mathsf{LinearProgramming}_\mathsf{u}) = \Omega(m^{\lfloor n/2 \rfloor})$.
- $\mathsf{R}(\mathsf{SubsetSum}_\mathsf{u}) = \Omega(2^n)$.

The approaches for Nash and LP are actually similar, yet the running time differs significantly. The key property that guarantees the efficiency of the algorithm for Nash is the existence of Nash equilibrium for any finite game. The algorithm for Nash could thus serve as an interesting example to illustrate how the solution-existing property helps computational efficiency.

Moreover, the following time complexity upper bound for $\mathsf{LinearProgramming}_\mathsf{u}$ is established, which is exponential in the number of variables but not in the number of constraints.

Theorem 5 ([5]) *The* $\mathsf{LinearProgramming}_\mathsf{u}$ *problem with* m *constraints,* n *variables, and input size* L *can be deterministically solved in time* $(mnL)^{poly(n)}$. *In particular, the algorithm is of polynomial time for constant dimensional linear programming (i.e., constant number of variables* n*).*

In summary, these results illustrate the variety of time and trial complexities that arise from the lack of input information for different problems and imply distinct levels of the cruciality of input information for different problems.

Related Work

The trial and error model bears a resemblance to certain other problems and models, e.g., learning,

algorithm design in unknown environments, ellipsoid method, and query complexity. However, there are fundamental distinctions between these models and ours. (More discussions are referred to [4].)

Learning

Trial and error model has apparent connections to various learning theories (e.g., concept learning with membership or equivalence query [1], decision tree learning, reinforcement learning [3], and (semi-)supervised learning [2]), but fundamental differences also exist. A common high-level philosophy of various learning models is to "sample and predict," which is very different from our "trial and search" (for a solution) in current setting. With its solution-oriented objective and advantages in computational efficiency, the trial and error model is hopefully to serve as a useful supplement to existing learning theories, particularly in contexts in which the unknown object itself is impossible or unaffordable to learn and the only available access to the unknown is through a solution-verification process.

Ellipsoid Method

The ellipsoid method is an elegant approach for proving the polynomial time solvability of a class of combinatorial optimization problems (see, e.g., [8]); it applies even when the explicit expressions of the constraints are unknown. The algorithm works as long as there exists an oracle that, on a proposed candidate solution, returns a violation in the form of a separating hyperplane.

In general, trial and error model has a similarity to the ellipsoid method, in which a point is proposed as a trial and a separating hyperplane is returned as an error. Our $\mathsf{LinearProgramming}_{\mathsf{u}}$ problem studies how to solve linear programs where the returned error is merely the *index* of a violated hyperplane (with the actual hyperplane still hidden). Moreover, the trial and error model includes a much broader class of search problems – not only convex optimization problems, but also many with pure combinatorial structures (e.g., the SAT, GroupIso, and GraphIso problems discussed here). From this perspective, the ellipsoid method is only one possible approach for the trial and error search problems in current model.

Cross-References

- ▸ Certificate Complexity and Exact Learning
- ▸ Reinforcement Learning

Recommended Reading

1. Angluin D (2004) Queries revisited. Theor Comput Sci 313(2):175–194
2. Balcan M, Blum A (2010) A discriminative model for semi-supervised learning. JACM 57(3):19
3. Barto A, Sutton R (1998) Reinforcement learning: an introduction. MIT, Cambridge
4. Bei X, Chen N, Zhang S (2013) On the complexity of trial and error. In: Proceedings of the forty-fifth annual ACM symposium on theory of computing. ACM, New York, pp 31–40
5. Bei X, Chen N, Zhang S (2013) Solving linear programming with constraints unknown. arXiv:1304.1247
6. Chen X, Deng X, Teng S (2009) Settling the complexity of computing two-player nash equilibria. JACM 56(3):14
7. Daskalakis C, Goldberg P, Papadimitriou C (2009) Computing a nash equilibrium is PPAD-complete. SIAM J Comput 39(1):195–259
8. Grotschel M, Lovasz L, Schrijver A (1988) Geometric algorithms and combinatorial optimization. Springer, Berlin/New York
9. Ivanyos G, Kulkarni R, Qiao Y, Santha M, Sundaram A (2014) On the complexity of trial and error for constraint satisfaction problems. In: Automata, languages, and programming. Lecture notes in computer science, vol 8572. Springer, Berlin/Heidelberg, pp 663–675
10. Nisan N, Roughgarden T, Tardos E, Vazirani V (2007) Algorithmic game theory. Cambridge University Press, Cambridge/New York
11. Roth A (2008) Deferred acceptance algorithms: history, theory, practice, and open questions. Int J Game Theory 36:537–569
12. Roth A, Sotomayor M (1992) Two-sided matching: a study in game-theoretic modeling and analysis. Cambridge University Press, Cambridge /New York

Triangulation Data Structures

Luca Castelli Aleardi[1], Olivier Devillers[2], and Jarek Rossignac[3]
[1]Laboratoire d'Informatique (LIX), École Polytechnique, Bâtiment Alan Turing, Palaiseau, France
[2]Inria Nancy – Grand-Est, Villers-lès-Nancy, France
[3]Georgia Institute of Technology, Atlanta, GA, USA

Keywords

Compact data structures; Succinct representations; Triangulations; Triangle meshes

Years and Authors of Summarized Original Work

2008; Castelli Aleardi, Devillers, Schaeffer
2009; Gurung, Rossignac
2012; Castelli Aleardi, Devillers, Rossignac

Problem Definition

The main problem consists in designing space-efficient data structures allowing to represent the connectivity of triangle meshes while supporting fast navigation and local updates.

Mesh Structures: Definition

Triangle meshes are among the most common representations of shapes. A *triangle mesh* is a collection of triangle faces that define a polyhedral approximation of a surface. A mesh is *manifold* if every edge is bounding either one or two triangles and if the faces incident to a same vertex define a closed or open fan. Here we focus on manifold meshes. Assuming that the genus and the number of boundary edges are negligible when compared to the number n of vertices, the number m of faces is roughly equal to $2n$.

Data Structures: Classification

Mesh data structures can be compared with respect to several criteria. A basic requirement (the *traversability*) for mesh representations is to provide fast navigational operators allowing to perform a mesh traversal (such as walking around a vertex). Most representations are also *indexable*, allowing to access in constant time to the description of a given vertex or triangle, given its index. In order to support efficient processing of large meshes, one needs to reduce memory trashing during navigation. An effective way of doing so is to design *compact data structures* requiring small storage. Many applications ask for the *modifiability*: the manipulation of meshes requires to perform updates such as vertex insertions/deletions, edge collapses, and edge flips. The choice of the data structure should also depend on the *simplicity* of its implementation and on its *practical efficiency* on common input data.

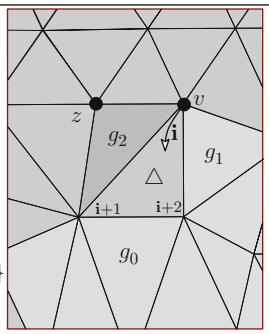
Standard Mesh Representations

Some common mesh representations are implemented in the explicit pointer-based form. References are used to describe incidence relations between mesh elements, and navigation is performed throughout address indirection. For example, a face-based representation [2] provides operators $\texttt{vertex}(\triangle, i)$ (giving the ith vertex of a triangle $\triangle$) and $\texttt{neighbor}(\triangle, i)$ (giving the i-neighbor of $\triangle$), as well as operator $\texttt{face}(v)$ (returning a triangle incident to vertex v). As illustrated in Fig. 1a, the combination of these operators allows to implement operators $\texttt{faceIndex}(\triangle_1, \triangle_2)$ (giving the index of $\triangle_1$ among the neighbors of $\triangle_2$) and $\texttt{vertexIndex}(v, \triangle)$ (giving the index of a vertex in $\triangle$). An alternative solution is given by the *Corner Table* proposed by Rossignac and colleagues, which uses integer indices to integer tables and provides a triangulation interface involving the corner operators defined in Fig. 2.

The two abstract data types above fully support local navigation in the mesh: the face-based as well as corner operators support efficient mesh exploration (see Figs. 1 and 2). A simple implementation stores explicitly all incidence

a

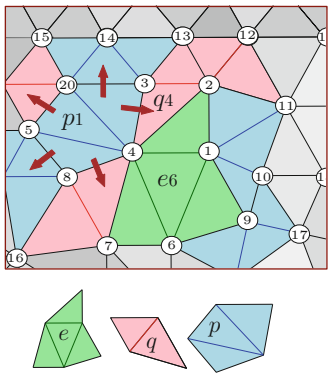
$v = \texttt{vertex}(\triangle, i)$
$\triangle = \texttt{face}(v)$
$i = \texttt{vertexIndex}(v, \triangle)$
$g_0 = \texttt{neighbor}(\triangle, i)$
$g_1 = \texttt{neighbor}(\triangle, ccw(i))$
$g_2 = \texttt{neighbor}(\triangle, cw(i))$
$z = \texttt{vertex}(g_2, \texttt{faceIndex}(g_2, \triangle))$
int ccw(int i) {return (i + 1)%3; }
int cw(int i) {return (i + 2)%3; }




```
int valence(int  v) {
 int d = 1;
 int f = face(v);
 int g = neighbor(f, cw(vertexIndex(v, f)));
 while (g! = f) {
    int next = neighbor(g, cw(faceIndex(f, g)));
    int i = faceIndex(g, next);
   g = next;
   d + +;
}
return d; }
```

b

```
class Quad extends Patch {
  Patch p1, p2, p3, p4;
  Vertex v1, v2, v3, v4;
}
class Pentagon extends Patch {
  Patch p1, p2, p3, p4, p5;
  Vertex v1, v2, v3, v4, v5;
}
class Hexagon extends Patch {
  Patch p1, p2, p3, p4, p5, p6;
  Vertex v1, v2, v3, v4, v5, v6;
}
```



Triangulation Data Structures, Fig. 1 (**a**) Triangle-based data structure: each triangle stores references to the 3, neighbors and to the 3 incident vertices yielding $13rpv$. (**b**) Catalog-based representation: using a catalog of size 3 one can guarantee that any quad is adjacent to at most two other quads, leading to a cost of $8.5\,rpv$

relations involving faces or corners, using 6 references per triangle plus one reference per vertex (describing the map from vertices to faces): according to Euler formula, this leads to a storage cost of 13 *references per vertex* (rpv). The results of `triangle`(c) and `next`(c) are not stored explicitly but calculated assuming that the three corners of each triangle are assigned consecutive indices.

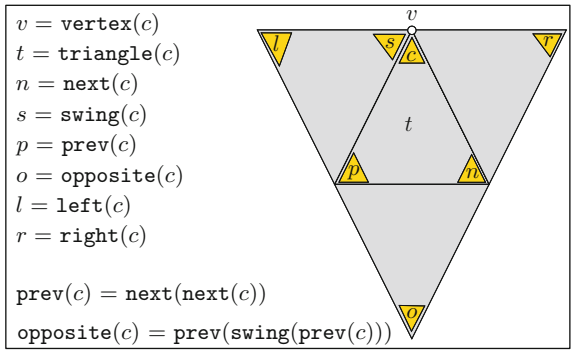
Key Results

A Theoretically Optimal Representation

From the information theory point of view, encoding a planar triangulation requires 3.24 *bits per vertex* (bpv), which is much less than the $13 \log n\, bpv$ used by standard representations. *Succinct representations* provide theoretically optimal encodings for triangulations, which match the optimal asymptotic bound of $3.24\,bpv$ (or equivalently $1.62m$ bits), while efficiently supporting navigational operations [4, 5], as stated below.

Theorem 1 *Given a planar triangulation $\mathcal{T}$ of m triangles, there exists a succinct representation that uses $1.62m + O\left(\frac{m \log\log m}{\log m}\right)$ bits, supporting navigation in worst case $O(1)$ time.*

This result is achieved with a multilevel hierarchical structure. The initial triangulation of size m is decomposed into *small triangulations*, each having $\Theta(\log^2 m)$ triangles: such a decomposition leads to a map $\mathcal{F}$ describing adjacency relations between small triangulations. Small triangulations are then decomposed into *tiny triangulations* of size $\Theta(\log m)$, whose adjacency relations are described by a map $\mathcal{G}$. Map $\mathcal{F}$ has $O\left(\frac{m}{\log^2 m}\right)$ nodes and




```
int c = seed;
visited[vertex(next(c))] = true;
visited[vertex(previous(c))] = true;
do {
   if(!visited[vertex(c)]){
     visited[vertex(c)] = true;
     explored[triangle(c)] = true;
   }
  else if(!explored[triangle(c)]) c = opposite(c);
  c = right(c);
}while(c! = opposite(s));
```

Triangulation Data Structures, Fig. 2 The *Corner Table*: corner operators allow to implement local navigation, as illustrated by the code of the *Ring-Expander* procedure [10]

arcs and can be stored in sublinear space using $O\left(\frac{m}{\log^2 m}\right)$ references of size $O(\log m)$ (actually $O\left(\log \frac{m}{\log^2 m}\right) < O(\log m)$). Map $\mathcal{G}$ has $O\left(\frac{m}{\log m}\right)$ nodes and arcs: adjacencies between two tiny triangulation within the same small triangulation need references of size $O\left(\log \frac{\log^2 m}{\log m}\right) = O(\log \log m)$ while adjacencies crossing the small triangulation boundaries are accessed by referring to $\mathcal{F}$. In that way the storage of both $\mathcal{F}$ and $\mathcal{G}$ is sublinear. The structure of tiny triangulations is optimally encoded throughout lookup into a table storing all possible triangulations of size $O(\log m)$. Such a framework can be extended in order to support updates: vertex deletions and edge flips are performed in $O(\log^2 m)$ amortized time (vertex insertions require $O(1)$ amortized time). The optimality stated by Theorem 1 is obtained combining the two-level representation with a careful decomposition of the mesh into tiny regions, involving a bijection between triangulations and a special class of vertex spanning trees [13].

A different approach, based on small separators, leads to compact representations [1] using $O(n)$ bits for more general classes of meshes (storage performances are difficult to evaluate precisely).

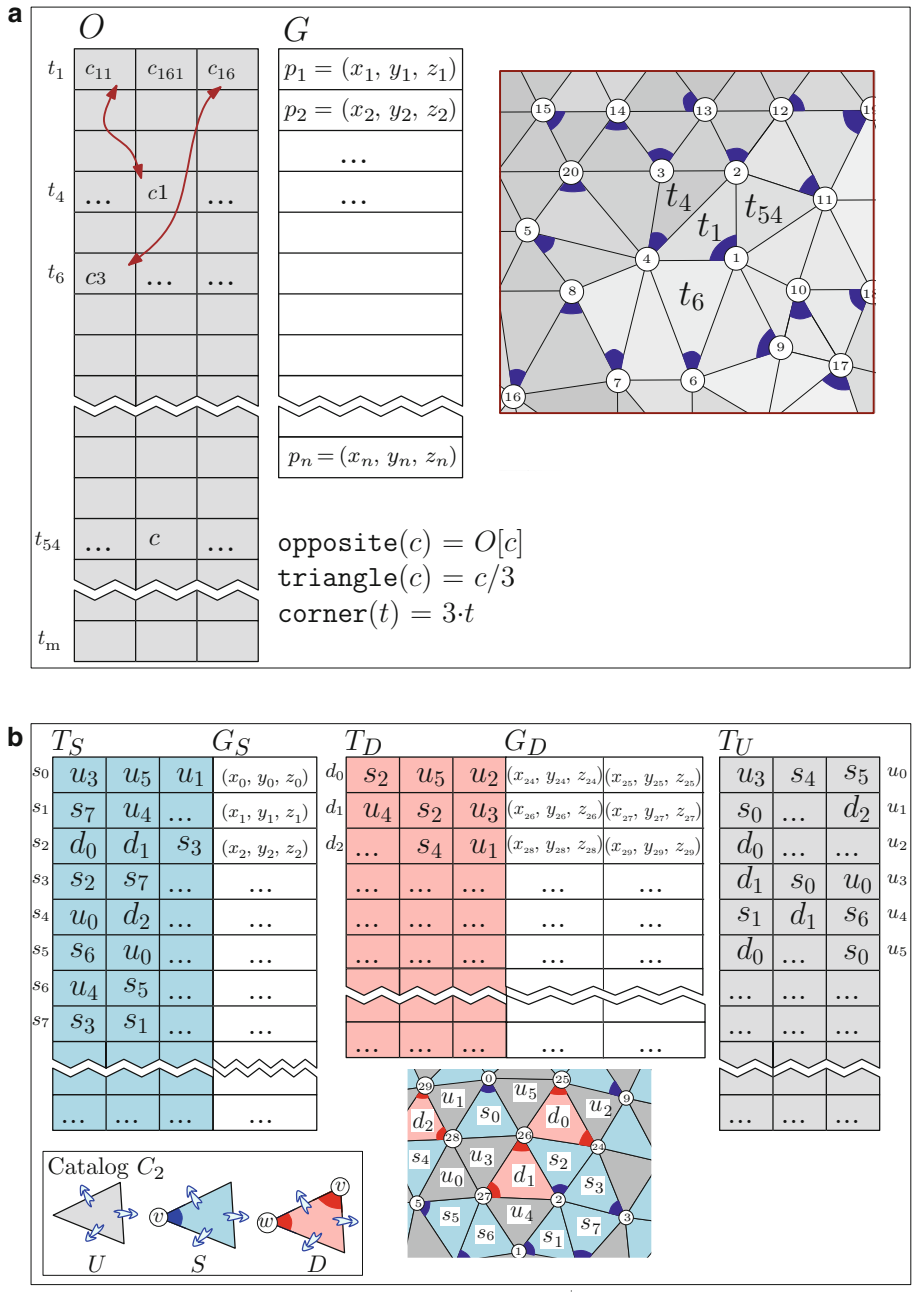
A More Practical Solution

Succinct representations run under the *word-RAM model* and are mainly of theoretical interest, since the amount of memory required in practice is quite important even for very large meshes. Some attempts to exploit the algorithmic framework of succinct representations in practice had lead to a space-efficient dynamic data structure [6]. The main idea is to gather together neighboring faces into small groups of triangles (called *patches*). While references are still of size $\Theta(\log n)$, grouping triangles allows to save some references (corresponding to edges internal to a given patch). For example, using a catalog consisting only of triangles and quadrangles, we encode a triangulation with at most $10.6\,rpv$ (a 19 % improvement over simple representations mentioned earlier). More sophisticated choices of patches lead to dynamic structures with smaller storage (e.g., Fig. 1b), as stated below:

Theorem 2 *Given a triangulation (possibly having handles and boundaries), there exists a data structure using* $7.67\,rpv$*, which allows* $O(1)$ *time navigation and supports updates in* $O(1)$ *amortized time.*

Reducing Redundancy Throughout Face Reordering

The main idea used in the SOT data structure [8] is to implicitly represent the map from triangles to corners (`triangle` operator), and the map from corners to vertices (`vertex` operator), through face reordering. First, match each vertex to an incident triangle (in such a way a triangle is matched with at most one vertex). Then permute triangles in such a way that the triangle associated



Triangulation Data Structures, Fig. 3 Illustrations of the SOT (**a**) and ESQ (**b**) data structures

with the ith vertex v_i has number i (thus, the first n triangles appearing in this ordering are the ones associated with a vertex). The corners of a triangle are listed consecutively, and the first one corresponds to the vertex matched for the triangle. The incidence relations are stored in an array O (of length $3m$) having 3 entries per triangle: $O[i]$ stores the index of the corner opposite to c_i (which is matched to vertex v_i, for $i \leq m$). Corner operators are supported in $O(1)$ time performing arithmetic operations (see Fig. 3a). Accessing a vertex v_i requires to walk

around its incident faces until c_i is reached (v_i being matched to c_i).

Theorem 3 ([8]) *Given a triangulation (possibly having handles and boundaries), there exists a data structure using* $6\,rpv$ *which supports* $O(1)$ *time navigation (retrieving a vertex of degree* d *requires* $O(d)$ *time).*

More Compact (Static) Representations

Combining this reordering approach with a pairing of adjacent triangles into quads, the SQUAD data structure [9] reaches better storage requiring slightly more than $4\,rpv$ according to experimental results on common meshes (the worst case upper bound is still $6\,rpv$). If one is allowed to perform a reordering of the input vertices, it is possible to guarantee a storage of $4\,rpv$ in the worst case (with same time performances as before): the edge-based representation described in [3] matches this bound exploiting Schnyder woods decompositions [14]. Various heuristics allows to further reduce storage requirements in practice [10–12].

A Dynamic Representation

Combining the reordering approach described above with the decomposition into triangle patches, the ESQ data structure [7] exhibits the same navigation performances as in SOT, while supporting local updates. As in [6] the mesh is decomposed into a collection of patches, each consisting of one or more triangles, and vertices are matched to patches. The assumption that each vertex is matched to a different triangle is relaxed. The catalog thus consists of a collection of k patch types (having possible one or more marked corners, describing how vertices are matched). Adjacency relations between faces are stored in k tables $T_1, \ldots, T_k$, one for each patch type (see Fig. 3b). Extending the approach introduced in SOT, a reordering of the input vertices allows to represent the maps from vertices to triangles and from triangles to vertices. A table T_S of type $S = (c, b)$ (with b boundary edges and c matched vertices) contains b references for each entry; the entries in the associated table G_S (containing geometric coordinates) are ordered accordingly. The decomposition into patches is maintained under local modifications with a constant number of memory updates in tables T_i.

Theorem 4 ([7]) *Given a triangulation (possibly having handles and boundaries), there exists a dynamic data structure using* $4.8\,rpv$*, which allows* $O(1)$ *time navigation and* $O(d)$ *time access to a vertex of degree* d*. Updates (vertex insertions/deletions and edge flips) are supported in* $O(1)$ *amortized time.*

Experimental Results

In [9] are reported timing comparisons of operators for SOT, SQUAD, and Corner Table data structures: experimental evaluations concern adjacency and navigational operations. On the tested mesh (the 55 millions triangles David), SQUAD requires 20 s and uses 2.2 GB of RAM for the construction (on a MacbookPro, equipped with 2.66 GHz Intel Core i7, 8 GB). When the whole mesh fits in main memory, compact data structures (SQUAD and SOT) perform slower than Corner Table. When the allowed memory is reduced, SQUAD performances are comparable and sometimes even better than Corner Table performances for high-level tasks (e.g., valence computations).

Cross-References

- Compressed Representations of Graphs
- Delaunay Triangulation and Randomized Constructions

Recommended Reading

1. Blanford D, Blelloch G, Kash I (2003) Compact representations of separable graphs. In: SODA, Baltimore, pp 342–351. http://dl.acm.org/citation.cfm?id=644219
2. Boissonnat JD, Devillers O, Pion S, Teillaud M, Yvinec M (2002) Triangulations in CGAL. Comput Geom 22:5–19.
3. Castelli Aleardi L, Devillers O (2011) Explicit array-based compact data structures for triangulations. In: ISAAC, Yokohama, pp 312–322.

4. Castelli Aleardi L, Devillers O, Schaeffer G (2005) Succinct representation of triangulations with a boundary. In: WADS, Waterloo, pp 134–145.
5. Castelli Aleardi L, Devillers O, Schaeffer G (2008) Succinct representations of planar maps. Theor Comput Sci 408(2–3):174–187.
6. Castelli Aleardi L, Devillers O, Mebarki A (2011) Catalog based representation of 2D triangulations. Int J Comput Geom Appl 21(4):393–402.
7. Castelli Aleardi L, Devillers O, Rossignac J (2012) ESQ: editable squad representation for triangle meshes. In: SIBGRAPI, Ouro Preto, pp 110–117.
8. Gurung T, Rossignac J (2009) SOT: compact representation for tetrahedral meshes. In: Proceedings of the ACM symposium on solid and physical modeling, San Francisco, pp 79–88.
9. Gurung T, Laney D, Lindstrom P, Rossignac J (2011) SQUAD: compact representation for triangle meshes. Comput Graph Forum 30(2):355–364.
10. Gurung T, Luffel M, Lindstrom P, Rossignac J (2011) LR: compact connectivity representation for triangle meshes. ACM Trans Graph 30(4):67.
11. Gurung T, Luffel M, Lindstrom P, Rossignac J (2013) Zipper: a compact connectivity data structure for triangle meshes. Comput-Aided Des 45(2):262–269.
12. Luffel M, Gurung T, Lindstrom P, Rossignac J (2014) Grouper: a compact, streamable triangle mesh data structure. IEEE Trans Vis Comput Graph 20(1):84–98.
13. Poulalhon D, Schaeffer G (2006) Optimal coding and sampling of triangulations. Algorithmica 46:505–527.
14. Schnyder W (1990) Embedding planar graphs on the grid. In: SODA, San Francisco, 138–148. http://dl.acm.org/citation.cfm?id=320191

Truthful Mechanisms for One-Parameter Agents

Moshe Babaioff
Microsoft Research, Herzliya, Israel

Keywords

Algorithmic mechanism design; Approximation; Scheduling related parallel machines; Truthful mechanisms

Synonyms

Dominant strategy mechanisms; Incentive compatible mechanisms; Single-parameter agents; Truthful auctions

Years and Authors of Summarized Original Work

2001; Archer, Tardos

Problem Definition

This problem is concerned with designing truthful (dominant strategy) mechanisms for domains where each agent's private information is expressed by a single positive real number. The goal of the mechanisms is to allocate loads placed on the agents, and an agent's private information is the cost incurred per unit load. Archer and Tardos [4] give an exact characterization for the algorithms that can be used to design truthful mechanisms for such load balancing problems using appropriate payments. The characterization shows that the allocated load must be monotonic in the cost (decreasing when the cost on an agent increases, fixing the costs of the others). Thus, truthful mechanisms are characterized by a condition on the allocation rule, and payments that ensure voluntary participation can be calculated using the given characterization.

The characterization is used to design polynomial time truthful mechanisms for several problems in combinatorial optimization to which the celebrated VCG mechanism does not apply. For scheduling related parallel machines to minimize makespan ($Q\|C_{\max}$), Archer and Tardos [4] present a 3-approximation mechanism based on randomized rounding of the optimal fractional solution. This mechanism is truthful only in expectation (a weaker notion of truthfulness in which truthful bidding maximizes the agent's *expected* utility). Archer [3] improves it to a randomized 2-approximation truthful mechanism. Andelman, Azar, and Sorani [2] provide a deterministic truthful mechanism that is 5-approximation. Kovács improves it to 3-approximation in [12] and to 2.8-approximation in [13] (Kovács also gives other results for two special cases). Andelman, Azar, and Sorani [2] also present a deterministic Fully Polynomial Time Approximation Scheme (FPTAS) for scheduling on a fixed number of machines, as well as a suitable payment

scheme that yields a deterministic truthful mechanism. Dhangwatnotai et al. [8] present a randomized Polynomial Time Approximation Scheme (PTAS) that is truthful-in-expectation. Christodoulou and Kovács [7] present a truthful deterministic Polynomial Time Approximation Scheme (PTAS); this matches the best possible result for the computational problem (without incentives) by Hochbaum and Shmoys [10] (it is known that this problem is strongly NP-hard [9]). This result shows that there is no "cost of truthfulness" for this problem, as the best approximation with incentive constraints is as good as the best approximation without these constraints.

Archer and Tardos [4] also present results for goals other than minimizing the makespan. They present a truthful mechanism for $Q \| \sum C_j$ (scheduling related machines to minimize the sum of completion times) and show that for $Q \| \sum w_j C_j$ (minimizing the weighted sum of completion times) $\frac{2}{\sqrt{3}}$ is the best approximation ratio achievable by a truthful mechanism.

This family of problems belongs to the field of Algorithmic Mechanism Design, initiated in the seminal paper of Nisan and Ronen [15]. Nisan and Ronen consider makespan minimization for scheduling on *unrelated* machines and prove upper and lower bounds (note that for unrelated machines agents have more than one parameter). Mu'alem and Schapira [14] present improved lower bounds. Other papers consider the problem of scheduling on related machines to minimize the makespan. Auletta et al. [5] and Ambrosio and Auletta [1] present truthful mechanisms for several NP-hard restrictions of this problem. Nisan and Ronen [15] also introduce a model in which the mechanism is allowed to observe the machines' actual processing time and compute the payments afterward (in such a model the machines essentially cannot claim to be faster than they are); Auletta et al. [6] present additional results for this model. In particular, they show that it is possible to overcome the lower bound of $\frac{2}{\sqrt{3}}$ for $Q \| \sum w_j C_j$ (minimizing the weighted sum of completion times) and provide a polynomial time $(1+\epsilon)$-approximation truthful mechanism (with verification) when the number of machines (m) is constant.

The Mechanism Design Framework

Let I be the set of agents. Each agent $i \in I$ has some private *value* (type) consisting of a single parameter $t_i \in \mathbb{R}$ that describes the agent, and which only i knows. Everything else is public knowledge. Each agent will report a *bid* b_i to the mechanism. Let t denote the vector of true values, and b the vector of bids.

There is some set of *outcomes* O, and given the bids b the mechanism's output algorithm computes an outcome $o(b) \in O$. For any types t, the mechanism aims to choose an outcome $o \in O$ that minimizes some function $g(o,t)$. Yet, given the bids b the mechanism can only choose the outcome as a function of the bids ($o = o(b)$) and has no knowledge of the true types t. To overcome the problem that the mechanism knows only the bids b, the mechanism is designed to be truthful (using payments), that is, in such a mechanism it is a dominant strategy for the agents to reveal their true types ($b = t$). For such mechanisms minimizing $g(o,t)$ is done by assuming that the bids are the true types (and this is justified by the fact that truth telling is a dominant strategy).

In the framework discussed here we assume that outcome $o(b)$ will assign some amount of *load* or work $w_i(o(b))$ to each agent i, and given $o(b)$ and t_i, agent i incurs some monetary *cost*, $cost_i(t_i, o(b)) = t_i w_i(o(b))$. Thus, agent i's private data t_i measures her cost per unit work. Each agent i attempts to maximize her *utility* (profit), $u_i(t_i, b) = P_i(b) - cost_i(t_i, o(b))$, where $P_i(b)$ is the *payment* to agent i.

Let b_{-i} denote the vector of bids, not including agent i, and let $b = (b_{-i}, b_i)$. Truth telling is a *dominant strategy* for agent i if bidding t_i always maximizes her utility, regardless of what the other agents bid. That is, $u_i(t_i, (b_{-i}, t_i)) \geq u_i(t_i, (b_{-i}, b_i))$ for all b_{-i} and b_i.

A mechanism M consists of the pair $M = (o(\cdot), P(\cdot))$, where $o(\cdot)$ is the *output function* and $P(\cdot)$ is the *payment scheme*, i.e., the vector of payment functions $P_i(\cdot)$. An output function *admits a truthful payment scheme* if there exist payments $P(\cdot)$ such that for the mechanism $M = (o(\cdot), P(\cdot))$, truth telling is a dominant

strategy for each agent. A mechanism that admits a truthful payment scheme is *truthful*.

Mechanism M satisfies the *voluntary participation* condition if agents who bid truthfully never incur a net loss, i.e., $u_i(t_i, (b_{-i}, t_i)) \geq 0$ for all agents i, true values t_i, and other agents' bids b_{-i}.

Definition 1 With the other agents' bids b_{-i} fixed, the *work curve* for agent i is $w_i(b_{-i}, b_i)$ considered as a single-variable function of b_i. The output function o is *decreasing* if each of the associated work curves is decreasing (i.e., $w_i(b_{-i}, b_i)$ is a decreasing function of b_i, for all i and b_{-i}).

Scheduling on Related Machines

There are n jobs and m machines. The jobs represent amounts of work $p_1 \geq p_2 \geq \ldots \geq p_n$, and let p denote the set of jobs. Machine i runs at some speed s_i, so it must spend p_j/s_i units of time processing each job j assigned to it. The input to an algorithm is b, the (reported) speed of the machines, and the output is $o(b)$, an assignment of jobs to machines. The load on machine i for outcome $o(b)$ is $w_i(b) = \sum p_j$, where the sum runs over jobs j assigned to i. Each machine incurs a cost proportional to the time it spends processing its jobs. The cost of machine i is $cost_i(t_i, o(b)) = t_i w_i(o(b))$, where $t_i = 1/s_i$ and $w_i(b)$ is the total load assigned to i when the speeds are b. Let C_j denote the completion time of job j. One can consider the following goals for scheduling related parallel machines:

- Minimizing the makespan $(Q\|C_{\max})$, the mechanism's goal is to minimize the completion time of the last job on the last machine, i.e., $g(o,t) = C_{\max} = \max_i t_i \cdot w_i(b)$.
- Minimize the sum of completion times $(Q\|\sum C_j)$, i.e., $g(o,t) = Q\|\sum C_j = \sum_j C_j$
- Minimize the weighted sum of completion times $(Q\|\sum w_j C_j)$, i.e., $g(o,t) = Q\|\sum w_j C_j = \sum_j w_j C_j$ where w_j is the weight of job j.

An algorithm is a *c-approximation algorithm* with respect to g, if for every instance (p,t), it outputs an outcome of cost at most $c \cdot g(o(t),t)$. A *c-approximation mechanism* is a mechanism whose output algorithm is an c-approximation. Note that if the mechanism is truthful the approximation is with respect to the true speeds. A *PTAS* (Polynomial Time Approximation Scheme) is a family of algorithms such that for every $\epsilon > 0$ there exists a $(1+\epsilon)$-approximation algorithm. If the running time is also polynomial in $1/\epsilon$, the family of algorithms is a *FPTAS* (Fully Polynomial Time Approximation Scheme).

Key Results

The following two theorems hold for the mechanism design framework as defined in section "Problem Definition."

Theorem 1 ([4]) *The output function $o(b)$ admits a truthful payment scheme if and only if it is decreasing. In this case, the mechanism is truthful if and only if the payments $P_i(b_{-i}, b_i)$ are of the form*

$$h_i(b_{-i}) + b_i w_i(b_{-i}, b_i) - \int_0^{b_i} w_i(b_{-i}, u)du$$

where the h_i are arbitrary functions.

Theorem 2 ([4]) *A decreasing output function admits a truthful payment scheme satisfying voluntary participation if and only if $\int_0^\infty w_i(b_{-i}, u)du < \infty$ for all i, b_{-i}. In this case, the payments can be defined by*

$$P_i(b_{-i}, b_i) = b_i w_i(b_{-i}, b_i) + \int_{b_i}^\infty w_i(b_{-i}, u)du$$

Theorem 3 ([4]) *There is a truthful mechanism (not polynomial time) that outputs an optimal solution for $Q\|C_{\max}$ and satisfies voluntary participation.*

Theorem 4 ([2, 7]) *For the problem of minimizing the makespan $(Q\|C_{\max})$:*

- *There exists a deterministic Polynomial Time Approximation Scheme (PTAS) for scheduling*

on related machines that admits a truthful payment scheme [7]. The mechanism created satisfies voluntary participation.

- *There exists a deterministic Fully Polynomial Time Approximation Scheme (FPTAS) for scheduling on a* fixed *number of machines that admits a truthful payment scheme [2]. The mechanism created satisfies voluntary participation.*

Theorem 5 ([4]) *There is a truthful polynomial time mechanism that outputs an optimal solution for* $Q \| \sum C_j$ *and satisfies voluntary participation.*

Theorem 6 ([4]) *No truthful mechanism for* $Q \| \sum w_j C_j$ *can achieve an approximation ratio better than* $\frac{2}{\sqrt{3}}$*, even on instances with just two jobs and two machines.*

Applications

Archer and Tardos [4] apply the characterization of truthful mechanisms to problems other than scheduling. They present results for the uncapacitated facility location problem as well as the maximum flow problem.

Kis and Kapolnai [11] consider the problem of scheduling of groups of identical jobs on related machines with sequence independent setup times ($Q|u_j, p_{jk} = p_j \| C_{\max}$). They provide a truthful, polynomial time, randomized mechanism for the batch scheduling problem with a deterministic approximation guarantee of 4 to the minimal makespan, based on the characterization of truthful mechanisms presented above.

Open Problems

The problem of designing truthful mechanisms for related machines to minimize the makespan was completely resolved, as a deterministic PTAS [7] is the best one can hope for. For this problem there is no gap between the best approximation with and without incentives. The main open problem left is of finding some natural single-parameter setting in which there is a gap between the approximation that is achievable by algorithms and truthful mechanisms.

Experimental Results

None is reported.

Data Sets

None is reported.

URL to Code

None is reported.

Cross-References

- Algorithmic Mechanism Design
- Competitive Auction
- Generalized Vickrey Auction
- Incentive Compatible Selection

Recommended Reading

1. Ambrosio P, Auletta V (2004) Deterministic monotone algorithms for scheduling on related machines. In: 2nd workshop on approximation and online algorithms (WAOA), Bergen, pp 267–280
2. Andelman N, Azar Y, Sorani M (2005) Truthful approximation mechanisms for scheduling selfish related machines. In: 22nd annual symposium on theoretical aspects of computer science (STACS), Stuttgart, pp 69–82
3. Archer A (2004) Mechanisms for discrete optimization with rational agents. PhD thesis, Cornell University
4. Archer A, Tardos É (2001) Truthful mechanisms for one-parameter agents. In: 42nd annual symposium on foundations of computer science (FOCS), Las Vegas, pp 482–491
5. Auletta V, De Prisco R, Penna P, Persiano G (2004) Deterministic truthful approximation mechanisms for scheduling related machines. In: 21nd annual symposium on theoretical aspects of computer science (STACS), Montpellier, pp 608–619
6. Auletta V, De Prisco R, Penna P, Persiano G, Ventre C (2006) New constructions of mechanisms with verification. In: 33rd international colloquium on automata, languages and programming (ICALP), Venice, (1), pp 596–607

7. Christodoulou G, Kovács A (2013) A deterministic truthful ptas for scheduling related machines. SIAM J Comput 42(4):1572–1595
8. Dhangwatnotai P, Dobzinski S, Dughmi S, Roughgarden T (2011) Truthful approximation schemes for single-parameter agents. SIAM J Comput 40(3):915–933
9. Garey MR, Johnson DS (1990) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman & Co., New York
10. Hochbaum D, Shmoys D (1988) A polynomial approximation scheme for scheduling on uniform processors: Usingthe dual approximation approach. SIAM J Comput 17(3):539–551
11. Kis T, Kapolnai R (2007) Approximations and auctions for scheduling batches on related machines. Oper Res Lett 35(1):61–68
12. Kovács A (2005) Fast monotone 3-approximation algorithm for scheduling related machines. In: 13th annual European symposium (ESA), Palma de Mallorca, Spain, pp 616–627
13. Kovács A (2007) Fast algorithms for two scheduling problems. PhD thesis, Universität des Saarlandes
14. Mu'alem A, Schapira M (2007) Setting lower bounds on truthfulness: extended abstract. In: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms (SODA '07), New Orleans. Society for Industrial and Applied Mathematics, Philadelphia, pp 1143–1152. ISBN: 978-0-898716-24-5, http://dl.acm.org/citation.cfm?id=1283383.1283506
15. Nisan N, Ronen A (2001) Algorithmic mechanism design. Games Econ Behav 35:166–196

Truthful Multicast

Weizhao Wang[1], Xiang-Yang Li[2], and Yu Wang[3]
[1]Google Inc., Irvine, CA, USA
[2]Department of Computer Science, Illinois Institute of Technology, Chicago, IL, USA
[3]Department of Computer Science, University of North Carolina, Charlotte, NC, USA

Keywords

Strategyproof multicast mechanism; Truthful multicast routing

Years and Authors of Summarized Original Work

2004; Wang, Li, Wang

Problem Definition

Several mechanisms [1, 3, 5, 9], which essentially all belong to the VCG mechanism family, have been proposed in the literature to prevent the selfish behavior of unicast routing in a wireless network. In these mechanisms, the least cost path, which maximizes the social efficiency, is used for routing. Wang, Li, and Wang [8] studied the *truthful* multicast routing protocol for a selfish wireless network, in which selfish wireless terminals will follow their own interests. The multicast routing protocol is composed of two components: (1) the tree structure that connects the sources and receivers, and (2) the payment to the relay nodes in this tree. Multicast poses a unique challenge in designing strategyproof mechanisms due to the reason that (1) a VCG mechanism uses an output that maximizes the *social efficiency*; (2) it is NP-hard to find the tree structure with the minimum cost, which in turn maximizes the social efficiency. A range of multicast structures, such as the least cost path tree (LCPT), the pruning minimum spanning tree (PMST), virtual minimum spanning tree (VMST), and Steiner tree, were proposed to replace the optimal multicast tree. In [8], Wang et al. showed how payment schemes can be designed for existing multicast tree structures so that rational selfish wireless terminals will follow the protocols for their own interests.

Consider a communication network $G = (V, E, c)$, where $V = \{v_1, \cdots, v_n\}$ is the set of communication terminals, $E = \{e_1, e_2, \cdots, e_m\}$ is the set of links, and c is the cost vector of all agents. Here agents are terminals in a node weighted network and are links in a link weighted network. Given a set of sources and receivers $Q = \{q_0, q_1, q_2, \cdots, q_{r-1}\} \subset V$, the multicast problem is to find a tree $T \subset G$ spanning all terminals Q. For simplicity, assume that $s = q_0$ is the sender of a multicast session if it exists. All terminals or links are required to declare a cost of relaying the message. Let d be the declared costs of all nodes, i.e., agent i declared a cost d_i. On the basis of the declared cost profile d, a multicast tree needs to be constructed and the payment $p_k(d)$ for each agent k needs to be

decided. The utility of an agent is its payment received, minus its cost if it is selected in the multicast tree. Instead of reinventing the wheels, Wang et al. still used the previously proposed structures for multicast as the output of their mechanism. Given a multicast tree, they studied the design of strategyproof payment schemes based on this tree.

Notations

Given a network H, $\omega(H)$ denotes the total cost of all agents in this network. If the cost of any agent i (link e_i or node v_i) is changed to c'_i, the new network is denoted as $G' = (V, E, c|^i c'_i)$, or simply $c|^i c'_i$. If one agent i is removed from the network, it is denoted as $c|^i \infty$. For the simplicity of notation, the cost vector c is used to denote the network $G = (V, E, c)$ if no confusion is caused. For a given source s and a given destination q_i, $\mathsf{LCP}(s, q_i, c)$ represents the shortest path between s and q_i when the cost of the network is represented by vector c. $|\mathsf{LCP}(s, q_i, d)|$ denotes the total cost of the least cost path $\mathsf{LCP}(s, q_i, d)$. The notation of several multicast trees is summarized as follows.

1. Link Weighted Multicast Tree
   - **LCPT**: The union of all least cost paths from the source to receivers is called the *least cost path tree*, denoted by *LCPT*(*d*).
   - **PMST**: First construct the minimum spanning tree *MST*(*G*) on the graph *G*. Take the tree *MST*(*G*) rooted at sender *s*, prune all subtrees that do not contain a receiver. The final structure is called the Pruning Minimum Spanning Tree (PMST).
   - **LST**: The Link Weighted Steiner Tree (LST) can be constructed by the algorithm proposed by Takahashi and Matsuyama [6].
2. Node Weighted Multicast Tree
   - **VMST**: First construct a virtual graph using all receivers plus the sources as the vertices and the cost of LCP as the link weight. Then compute the minimum spanning tree on the virtual graph, which is called virtual minimum spanning tree (VMST). Finally, choose all terminals on the VMST as the relay terminals.
   - **NST**: The node weighted Steiner tree (NST) can be constructed by the algorithm proposed by [4].

Algorithm 1 Non-VCG mechanism for LCPT

1: For each receiver $q_i \neq s$, computes the least cost path from the source s to q_i, and compute a payment $p_k^i(d)$ to every link e_k on the $\mathsf{LCP}(s, q_i, d)$ using the scheme for unicast

$$p_k^i(d) = d_k + |\mathsf{LCP}(s, q_i, d|^k \infty)| - |\mathsf{LCP}(s, q_i, d)|.$$

2: The final payment to link $e_k \in LCPT$ is then

$$p_k(d) = \max_{q_i \in Q} p_k^i(d). \quad (1)$$

The payment to each link not on LCPT is simply 0.

Key Results

If the LCPT tree is used as the multicast tree, Wang et al. proved the following theorem.

Theorem 1 *The VCG mechanism combined with LCPT is not truthful.*

Because of the failure of the VCG mechanism, they designed their non-VGC mechanism for the LCPT-based multicast routing as follows.

Theorem 2 *Payment (defined in Eq. (1)) based on LCPT is truthful and it is minimum among all truthful payments based on LCPT.*

More generally, Wang et al. [8] proved the following theorem.

Theorem 3 *The VCG mechanism combined with either one of the LCPT, PMST, LST, VMST, NST is not truthful.*

Because of this negative result, they designed their non-VCG mechanisms for all multicast structures they studied: LCPT, PMST, LST, VMST, NST. For example, Algorithm 2 is the algorithm for PMST. For other algorithms, please refer to [8].

Algorithm 2 Non-VCG mechanism for PMST

1: Apply VCG mechanism on the MST. The payment for edge $e_k \in PMST(d)$ is

$$p_k(d) = \omega(MST(d|^k \infty)) - \omega(MST(d)) + d_k. \quad (2)$$

2: For every edge $e_k \notin PMST(d)$, its payment is 0.

Regarding all their non-VGC mechanisms, they proved the following theorem.

Theorem 4 *The non-VCG mechanisms designed for the multicast structures LCPT, PMST, LST, VMST, NST are not only truthful, but also achieve the minimum payment among all truthful mechanisms.*

Applications

In wireless ad hoc networks, it is commonly assumed that, each terminal contributes its local resources to forward the data for other terminals to serve the common good, and benefits from resources contributed by other terminals to route its packets in return. On the basis of such a fundamental design philosophy, wireless ad hoc networks provide appealing features such as enhanced system robustness, high service availability and scalability. However, the critical observation that individual users who own these wireless devices are generally selfish and noncooperative may severely undermine the expected performances of the wireless networks. Therefore, providing incentives to wireless terminals is a must to encourage contribution and thus maintains the robustness and availability of wireless networking systems. On the other hand, to support a communication among a group of users, multicast is more efficient than unicast or broadcast, as it can transmit packets to destinations using fewer network resources, thus increasing the social efficiency. Thus, most results of the work of Wang et al. can apply to multicast routing in wireless networks in which nodes are selfish. It not only guarantees that multicast routing behaves normally but also achieves good social efficiency for both the receivers and relay terminals.

Open Problems

There are several unsolved challenges left as future work in [8]. Some of these challenges are listed below.

- How to design algorithms that can compute these payments in asymptotically optimum time complexities is presently unknown.
- Wang et al. [8] only studied the tree-based structures for multicast. Practically, mesh-based structures may be more needed for wireless networks to improve the fault tolerance of the multicast. It is unknown whether a strategyproof multicast mechanism can be designed for some mesh-based structures used for multicast.
- All of the tree construction and payment calculations in [8] are performed in a centralized way, it would be interesting to design some distributed algorithms for them.
- In the work by Wang et al. [8] it was assumed that the receivers will always relay the data packets for other receivers for free, the source node of the multicast will pay the relay nodes to compensate their cost, and the source node will not charge the receivers for getting the data. As a possible future work, the budget balance of the source node needs to be considered if the receivers have to pay the source node for getting the data.
- Fairness of payment sharing needs to be considered in a case where the receivers share the total payments to all relay nodes on the multicast structure. Notice that this is different from the cost-sharing studied in [2], in which they assumed a fixed multicast tree, and the link cost is publicly known; in that work they showed how to share the total link cost among receivers.
- Another important task is to study how to implement the protocols proposed in [8] in a distributed manner. Notice that, in [3, 9],

distributed methods have been developed for a truthful unicast using some cryptography primitives.

Cross-References

▸ Algorithmic Mechanism Design

Recommended Reading

1. Anderegg L, Eidenbenz S (2003) Ad hoc-VCG: a truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents. In: Proceedings of the 9th annual international conference on mobile computing and networking. ACM, New York, pp 245–259
2. Feigenbaum J, Papadimitriou C, Shenker S (2001) Sharing the cost of multicast transmissions. J Comput Syst Sci 63(1):21–41
3. Feigenbaum J, Papadimitriou C, Sami R, Shenker S (2002) A BGP-based mechanism for lowest-cost routing. In: Proceedings of the 2002 ACM symposium on principles of distributed computing, Monterey, 21–24 Jul 2002, pp 173–182
4. Klein P, Ravi R (1995) A nearly best-possible approximation algorithm for node-weighted Steiner trees. J Algorithm 19(1):104–115
5. Nisan N, Ronen A (1999) Algorithmic mechanism design. In: Proceedings of the 31st annual symposium on theory of computing (STOC99), Atlanta, 1–4 May 1999, pp 129–140
6. Takahashi H, Matsuyama A (1980) An approximate solution for the Steiner problem in graphs. Math Jpn 24(6):573–577
7. Wang W, Li X-Y (2006) Low-cost routing in selfish and rational wireless ad hoc networks. IEEE Trans Mob Comput 5(5):596–607
8. Wang W, Li X-Y, Wang Y (2004) Truthful multicast in selfish wireless networks. In: Proceedings of the 10th ACM annual international conference on mobile computing and networking, Philadelphia, 26 Sept-1 Oct 2004
9. Zhong S, Li L, Liu Y, Yang YR (2005) On designing incentive compatible routing and forwarding protocols in wireless adhoc networks – an integrated approach using game theoretical and cryptographic techniques. In: Proceedings of the 11th ACM annual international conference on mobile computing and networking, Cologne, 28 Aug-2 Sept 2005

TSP-Based Curve Reconstruction

Edgar Ramos
School of Mathematics, National University of Colombia, Medellín, Colombia

Years and Authors of Summarized Original Work

2001; Althaus, Mehlhorn

Problem Definition

An instance of the curve reconstruction problem is a finite set of *sample* points V in the plane, which are assumed to be taken from an unknown planar curve γ. The task is to construct a geometric graph G on V such that two points in V are connected by an edge in G if and only if the points are adjacent on γ. The curve γ may consist of one or more connected components, and each of them may be closed or open (with endpoints), and may be smooth everywhere (tangent defined at every point) or not.

Many heuristic approaches have been proposed to solve this problem. This work continues a line of reconstruction algorithms with *guaranteed* performance, i.e., algorithms which probably solve the reconstruction problem under certain assumptions of γ and V. Previous proposed solutions with guaranteed performances were mostly *local*: a subgraph of the complete geometric graph defined by the points is considered (in most cases the Delaunay edges), and then *filtered* using a local criteria into a subgraph that will constitute the reconstruction. Thus, most of these algorithms fail to enforce that the solution have the global property of being a path/tour or collection of paths/tours and so usually require a dense sampling to work properly and have difficulty handling nonsmooth curves. See [6, 7, 8] for surveys of these algorithms.

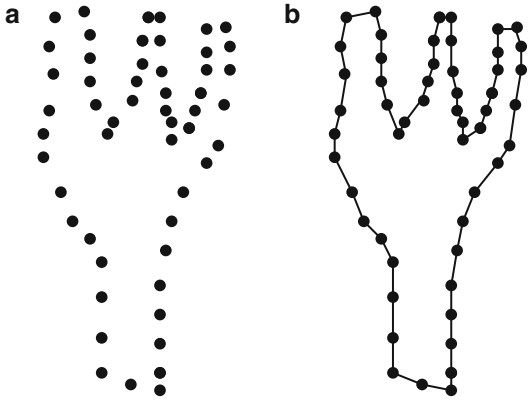
This work concentrates on a solution approach based on the *traveling salesman problem (TSP)*. Recall that a *traveling salesman path (tour)* for a set V of points is a path (cycle) passing through all points in V. An optimal traveling salesman path (tour) is a traveling salesman path (tour) of shortest length. The first question is under which conditions for γ and V a traveling salesman path (tour) is a correct reconstruction. Since the construction of an optimal traveling salesman path (tour) is an NP-hard problem, a second question is whether for the specific instances under consideration, an efficient algorithm is possible.

A previous work of Giesen [9] gave a first weak answer to the first question: For every benign semiregular closed curve γ, there exists an $\epsilon > 0$ with the following property: If V is a finite sample set from γ so that for every $x \in \gamma$ there is a $p \in V$ with $\|pv\| \leq \epsilon$, then the optimal traveling salesman tour is a polygonal reconstruction of γ. For a curve $\gamma : [0, 1] \to R^2$, its left and right tangents at $\gamma(t_0)$, are defined as the limits of the ratio $|\gamma(t_2) - \gamma(t_1)| / |t_2 - t_1|$ as (t_1, t_2) converges to (t_0, t_0) from the right $(t_0 < t_1 < t : 2)$ and from the right $(t_1 < t_2 < t : 0)$ respectively. A curve is semiregular if both tangents exist at every points and regular if the tangents exist and coincide at every point. The *turning* angle of γ at p is the angle between the left and right tangents at a points p. A semiregular curve is *benign* if the turning angle is less than π.

To investigate the TSP-based solution of the reconstruction problem, this work considers its integer linear programming *(ILP)* formulation and the corresponding linear programming *(LP)* relaxation. The motivation is that a successful method for solving the TSP is to use a branch-and-cut algorithm based on the LP-relaxation. See Chapter 7 in [5]. For a path with endpoints a and b, the formulation is based on variables $x_{u,v} \in \{0, 1\}$ for each pair u, v in V (indicating whether the edge uv is in the path $(x_{uv} = 1)$ or not $(x_{uv} = 0)$ and consists of the following objective function and constraints ($x_{uu} = 0$ for all $u \in V$):

$$
\begin{aligned}
\text{minimize} \quad & \sum_{u,v \in V} \|uv\| \cdot x_{uv} \\
\text{subject to} \quad & \sum_{v \in V} x_{uv} = 2 \text{ for all } u \in V \setminus \{a, b\} \\
& \sum_{v \in V} x_{uv} = 1 \text{ for } u \in \{a, b\} \\
& \sum_{u,v \in V'} x_{uv} \leq |V'| - 1 \text{ for } V' \subseteq V, \\
& V' \neq \emptyset \\
& x_{uv} \in \{0, 1\} \text{ for all } u, v \in V.
\end{aligned}
$$

Here $\|uv\|$ denotes the Euclidean distance between u and v and so the objective function is the total length of the selected edges. This is called the *subtour-ILP for the TSP with specified endpoints*. The equality constraints are called the *degree constraints*, the inequality ones are called *subtour elimination constraints* and the last ones are called the *integrality constraints*. If the degree and integrality constraints hold, the corresponding graph could include disconnected cycles (subtours), hence the need for the subtour elimination constraints. The relaxed LP is obtained by replacing the integrality constraints by the constraints $0 \leq x_{uv} \leq 1$ and is called the *subtour-LP for the TSP with specified endpoints*. There is a polynomial time algorithm that given a candidate solution returns a violated constraint if it exists: the degree constraints are trivial to check and the subtour elimination constraints are checked using a min cut algorithm (if a, b are joined by an edge and all edge capacities are made equal to one, then a violated subtour constraint corresponds to a cut smaller than two). This means that the subtour-LP for the TSP with specified endpoints can potentially be solved in polynomial time in the bit size of the input description, using the ellipsoid method [10].



TSP-Based Curve Reconstruction, Fig. 1 Sample data and its reconstruction

Key Results

The main results of this paper are that, given a sample set V with $a, b \in V$ from a benign semiregular open curve γ with endpoints a, b and satisfying certain *sampling condition* [it], then

- The optimal traveling salesman path on V with endpoints a, b is a polygonal reconstruction of γ from V,
- The subtour-LP for traveling salesman paths has an optimal integral solution which is unique.

This means that, under the sampling conditions, the subtour-LP solution provides a TSP solution and also suggests a reconstruction algorithm: solve the subtour-LP and, if the solution is integral, output it. If the input satisfies the sampling condition, then the solution will be integral and the result is indeed a polygonal reconstruction. Two algorithms are proposed to solve the subtour-LP. First, using the simplex method and the cutting plane framework: it starts with an LP consisting of only the degree constraints and in each iteration solves the current LP and checks whether that solution satisfies all the subtour elimination constraints (using a min cut algorithm) and, if not, adds a violated constraint to the current LP. This algorithm has a potentially exponential running time. Second, using a similar approach but with the ellipsoid method. This can be implemented so that the running time is polynomial in the bit size of the input points. This requires justification for using approximate point coordinates and distances.

The main tool in deriving these results is the connection between the subtour-LP and the so-called *Held–Karp bound*. The line of argument is as follows:

- Let $c(u, v) = \|uv\|$ and $\mu : V \to R$ be a *potential function*. The corresponding *modified distance function* c_μ is defined by $c_\mu(u, v) = c(u, v) - \mu(u) - \mu(v)$.
- For any traveling salesman path T with endpoints a, b,

$$c_\mu(T) = c(T) - 2\sum_{v \in V} \mu(v) + \mu(a) + \mu(b),$$

and so an optimal traveling salesman path with endpoints a, b for c_μ is also optimal for c.
- Let C_μ be the cost of a minimum spanning tree MST_μ under c_μ, then since a traveling salesman path is a spanning tree, the optimal traveling salesman T_0 satisfies $C_\mu \leq c_\mu(T_0) = c(T_0) - 2\sum_{v \in V} \mu(v) + \mu(a) + \mu(b)$, and so

$$\max_\mu \left(C_\mu + 2\sum_{v \in V} \mu(v) - \mu(a) - \mu(b) \right) \leq c(T_0)\,.$$

The term on the left is the so called Held–Karp bound.
- Now, if for a particular μ, MST_μ is a path with endpoints a, b, then MST_μ is in fact an optimal traveling salesman path with endpoints a, b, and the Held–Karp bound matches $c(T_0)$.
- The Held–Karp bound is equal to the optimal objective value of the subtour-LP. This follows by relaxation of the degree constraints in a Lagrangian fashion (see [5]) and gives an effective way to compute the Held-Karp bound: solve the subtour-LP.
- Finally, a potential function μ is constructed for γ so that, for an appropriately dense sample set V, MST_μ is unique and is a polygonal

reconstruction with endpoints a, b. This then implies that solving the subtour-LP will produce a correct polygonal reconstruction.

Note that the potential function μ enters the picture only as an analysis tool. It is not needed by the algorithm. The authors extend this work to the case of open curves without specified endpoints and of closed curves using variations of the ILP formulation and a more restricted sampling condition. They also extend it to the case of a collection of closed curves. The latter requires preprocessing that partitions points into groups that are expected to form individual curves. Then each subgroup is processed with the subtour-LP approach and then the quality of the result assessed and then that partition may be updated.

Finite Precision

The above results are obtained assuming exact representation of point samples and the distances between them, so claiming a polynomial time algorithm is not immediate as the running time of the ellipsoid method is polynomial in the bit size of the input. The authors extend the results to the case in which points and the distances between them are known only approximately and from this they can conclude the polynomial running time.

Relation to Local Feature Size

The defined potential function μ is related to the so called *local feature size* function f used in the theory of smooth curve reconstruction, where $f(p)$ is defined as the distance from p to the medial axis of the curve γ. In this paper, $\mu(p)$ is defined as $d(p)/3$ where $d(p)$ is the size of the largest neighborhood of p so that γ in that neighborhood does not *deviate significantly* from a flat segment of curve. This paper shows $f(p) < 3d(p)$. In fact, $\mu(p)$ amounts to a generalization of the local feature size to nonsmooth curves (for a corner point p, $\mu(p)$ is proportional to the size of the largest neighborhood of p such that γ inside does not deviate significantly from a corner point with two nearly flat legs incident to it, and for points near the corner, μ is defined as an appropriate interpolation of the two definitions), and is in fact similar to definitions proposed elsewhere.

Applications

The curve reconstruction problem appears in applied areas such as cartography. For example, to determine level sets, features, object contours, etc. from samples. Admittedly, these applications usually may require the ability to handle very sparse sampling and noise. The 3D version of the problem is very important in areas such as industrial manufacturing, medical imaging, and computer animation. The 2D problem is often seen as a simpler (toy) problem to test algorithmic approaches.

Open Problems

A TSP-based solution when the curve γ is a collection of curves, not all closed, is not given in this paper. A solution similar to that for closed curves (partitioning and then application of subtour-LP for each) seems feasible for general collections, but some technicalities need to be solved. More interesting is the study of corresponding reconstruction approaches for surfaces in 3D.

Experimental Results

The companion paper [2] presents results of experiments comparing the TSP-based approach to several (local) Delaunay filtering algorithms. The TSP implementation uses the simplex method and the cutting plane framework (with a potentially exponential running time algorithm). The experiments show that the TSP-based approach has a better performance, allowing for much sparser samples than the others. This is to be expected given the global nature of the TSP-based solution. On the other hand, the speed of the TSP-based solution is reported to be competitive when compared to the speed of the others, despite its potentially bad worst-case behavior.

Data Sets

None reported. Experiments in [2] were performed with a simple reproducible curve based on a sinusoidal with varying number of periods and samples.

URL to Code

The code of the TSP-based solution as well as the other solutions considered in the companion paper [2] are available from: http://www.mpi-inf.mpg.de/~althaus/LEP:Curve-Reconstruction/curve.html

Cross-References

- Engineering Geometric Algorithms
- Euclidean Traveling Salesman Problem
- Minimum Weight Triangulation
- Planar Geometric Spanners
- Robust Geometric Computation

Recommended Reading

1. Althaus E, Mehlhorn K (2001) Traveling salesman-based curve reconstruction in polynomial time. SIAM J Comput 31:27–66
2. Althaus E, Mehlhorn K, Näher S, Schirra S (2000) Experiments on curve reconstruction. In: ALENEX, pp 103–114
3. Amenta N, Bern M (1999) Surface reconstruction by Voronoi filtering. Discrete Comput Geom 22: 481–504
4. Amenta N, Bern M, Eppstein D (1998) The crust and the β-skeleton: combinatorial curve reconstruction. Graph Model Image Process 60:125–135
5. Cook W, Cunningham W, Pulleyblank W, Schrijver A (1998) Combinatorial optimization. Wiley, New York
6. Dey TK (2004) Curve and surface reconstruction. In: Goodman JE, O'Rourke J (eds) Handbook of discrete and computational geometry, 2nd edn. CRC, Boca Raton
7. Dey TK (2006) Curve and surface reconstruction: algorithms with mathematical analysis. Cambridge University Press, New York
8. Edlesbrunner H (1998) Shape reconstruction with the Delaunay complex. In: LATIN'98, theoretical informatics. Lecture notes in computer science, vol 1380. Springer, Berlin, pp 119–132
9. Giesen J (2000) Curve reconstruction, the TSP, and Menger's theorem on length. Discrete Comput Geom 24:577–603
10. Schrijver A (1986) Theory of linear and integer programming. Wiley, New York

Two-Dimensional Scaled Pattern Matching

Amihood Amir
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel
Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

Keywords

2d scaled matching; Multidimensional scaled search; Pattern matching in scaled images; Two dimensional pattern matching with scaling

Years and Authors of Summarized Original Work

2006; Amir, Chencinski

Problem Definition

Definition 1 Let T be a two-dimensional $n \times n$ array over some alphabet Σ.

1. The *unit pixels array* for T (T^{1X}) consists of n^2 unit squares, called pixels in the real plane $\Re^2$. The corners of the pixel $T[i, j]$ are $(i - 1, j-1)$, $(i, j-1)$, $(i-1, j)$, and (i, j). Hence the pixels of T form a regular $n \times n$ array that covers the area between $(0, 0)$, $(n, 0)$, $(0, n)$, and (n, n). Point $(0, 0)$ is the *origin* of the unit pixel array. The *center* of each pixel is the geometric center point of its square location. Each pixel $T[i, j]$ is identified with the value from Σ that the original array T had in that position. Say that the pixel has a color or

a character from Σ. See Fig. 1 for an example of the grid and pixel centers of a 7×7 array.

2. Let $r \in \Re, r \geq 1$. The *r-ary pixels array* for T (T^{rX}) consists of n^2 *r-squares*, each of dimension $r \times r$ whose *origin* is $(0,0)$ and covers the area between $(0,0)$, $(nr,0)$, $(0,nr)$, and (nr,nr). The corners of the pixel $T[i,j]$ are $((i-1)r,(j-1)r)$, $(ir,(j-1)r)$, $((i-1)r,jr)$, and (ir,jr). The *center* of each pixel is the geometric center point of its square location.

Notation: Let $r \in \Re$. $[r]$ denotes the *rounding* of r, i.e.,

$$[r] = \begin{cases} \lfloor r \rfloor & \text{if } r - \lfloor r \rfloor < .5; \\ \lceil r \rceil & \text{otherwise.} \end{cases}$$

Definition 2 Let T be an $n \times n$ text array, P be an $m \times m$ pattern array over alphabet Σ, and let $r \in \Re, 1 \leq r \leq \frac{n}{m}$. Say that there is an *occurrence of P scaled to r* at text location (i,j) if the following conditions hold:

Let T^{1X} be the unit pixels array of T and P^{rX} be the r-ary pixel arrays of P. Translate P^{rX} onto T^{1X} in a manner that the origin of P^{rX} coincides with location $(i-1, j-1)$ of T^{1X}. Every center of a pixel in T^{1X} which is within the area covered by $(i-1, j-1), (i-1, j-1+mr), (i-1+mr, j-1)$ and $(i-1+mr, j-1+mr)$ has the same color as the r-square of P^{rX} in which it falls.

The colors of the centers of the pixels in T^{1X} which are within the area covered by $(i-1, j-1), (i-1, j-1+mr), (i-1+mr, j-1)$ and $(i-1+mr, j-1+mr)$ define a $[mr] \times [mr]$ array over Σ. This array is denoted by $P^{s(r)}$ and called *P scaled to r*.

The above definition is the one provided in the *geometric model*, pioneered by Landau and Vishkin [15], and Fredriksson and Ukkonen [14]. Prior to the advent of the geometric model, the only discrete definition of scaling was to natural scales, as defined by Amir, Landau and Vishkin [10]:

Definition 3 Let $P[m \times m]$ be a two-dimensional matrix over alphabet Σ (not necessarily bounded). Then P scaled by s (P^s) is the $sm \times sm$ matrix where every symbol $P[i,j]$ of P is replaced by a $s \times s$ matrix whose elements all equal the symbol in $P[i,j]$. More precisely,

$$P^s[i,j] = P[\lceil \frac{i}{s} \rceil, \lceil \frac{j}{s} \rceil] .$$

Say that pattern $P[m \times m]$ *occurs* (or an occurrence of P starts) at location (k,l) of the text T if for any $i \in \{1,\ldots,m\}$ and $j \in \{1,\ldots,m\}$, $T[k+i-1, l+j-1] = P[i,j]$.

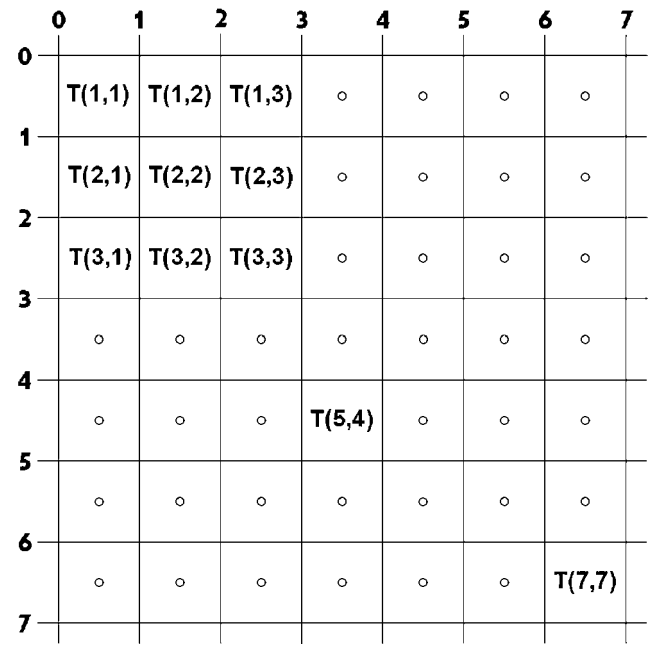
The *two dimensional pattern matching problem with natural scales* is defined as follows.

INPUT: Pattern matrix $P[i,j]$ $i = 1,\ldots m; j = 1,\ldots,m$ and Text matrix $T[i,j]$ $i = 1,\ldots,n; j = 1,\ldots,n$ where $n > m$.

OUTPUT: all locations in T where an occurrence of P scaled by s (an s-occurrence) starts, for any $s = 1,\ldots,\lfloor \frac{n}{m} \rfloor$.

The natural scales definition cannot answer normal everyday occurrences such as an image scaled to, say, 1.3. This led to the geometric model. The geometric model is a discrete adaptation, without smoothing, of scaling as used in computer graphics. The definition is pleasing in a "real-world" sense. Figure 2 shows "lenna" scaled to non-discrete scales by the geometric model definition. The results look natural.

It is possible, of course, to consider a *one dimensional* version of scaling, or scaling in *strings*. Both above definitions apply for one dimensional scaling where the text and pattern are taken to be matrices having a single row. The interest in one dimensional scaling lies because of two reasons: (1) There is a faster algorithm for one dimensional scaling in the geometric model than the restriction of the two dimensional scaling algorithm to one dimension. (2) Historically, before the geometric model was defined, there was an attempt [3] to define real scaling on strings as follows.



Two-Dimensional Scaled Pattern Matching, Fig. 1 The grid and pixel centers of a unit pixel array for a 7×7 array

Two-Dimensional Scaled Pattern Matching, Fig. 2 An original image, scaled by 1.3 and scaled by 2, using the geometric model definition of scaling

Definition 4 Denote the string $aa \cdots a$, where a is repeated r times, by a^r. The *one dimensional floor real scaled matching problem* is the following.

INPUT: A pattern $P = a_1^{r_1} a_2^{r_2} \ldots a_j^{r_j}$, of length m, and a text T of length n.

OUTPUT: All locations in the text where the substring $a_1^{c_1} a_2^{\lfloor r_2 k \rfloor} \ldots a_{j-1}^{\lfloor r_{j-1} k \rfloor} a_j^{c_j}$ appears, where $c_1 \geq \lfloor r_1 k \rfloor$ and $c_j \geq \lfloor r_j k \rfloor$.

This definition indeed handles real scaling but has a significant weakness in that a string of length m scaled to r may be significantly shorter than mr. For this reason the definition could not be generalized to two dimensions. The geometric model does not suffer from these deficiencies.

Key Results

The first results in scaled natural matching dealt with fixed finite-sized alphabets.

Theorem 1 (Amir, Landau, and Vishkin [10]) *There exists an $O(|T|\log|\Sigma|)$ worst-case time solution to the two dimensional pattern matching problem with natural scales, for fixed finite alphabet Σ.*

The main idea behind the algorithm is analyzing the text with the aid of *power columns*. Those are the text columns appearing $m-1$ columns apart, where P is an $m \times m$ pattern. This dependence on the pattern size make the power columns useless where a dictionary of different sized patterns is involved. A significantly simpler algorithm with an additional advantage of being alphabet-independent was presented in [6].

Theorem 2 (Amir and Calinescu [6]) *There exists an $O(|T|)$ worst-case time solution to the two dimensional pattern matching problem with natural scales.*

The alphabet independent time complexity of this algorithm was achieved by developing a scaling-invariant "signature" of the pattern. This idea was further developed to scaled dictionary matching.

Theorem 3 (Amir and Calinescu [6]) *Given a static dictionary of square pattern matrices. It is possible in $O(|D|\log k)$ preprocessing, where $|D|$ is the total dictionary size and k is the number of patterns in the dictionary, and $O(|T|\log k)$ text scanning time, for input text T, to find all occurrences of dictionary patterns in the text in all natural scales.*

This is identical to the time at [8], the best non-scaled matching algorithm for a static dictionary of square patterns. It is somewhat surprising that scaling does not add to the complexity of single matching nor dictionary matching.

The first algorithm to solve the scaled matching problem for real scales, was a one dimensional real scaling algorithm using Definition 4.

Theorem 4 (Amir, Butman, and Lewenstein [3]) *There exists an $O(|T|)$ worst-case time solution to the one dimensional floor real scaled matching problem.*

The first algorithm to solve the two dimensional scaled matching problem for real scales in the geometric model is the following.

Theorem 5 (Amir, Butman, Lewenstein, and Porat [4]) *Given an $n \times n$ text and $m \times m$ pattern. It is possible to find all pattern occurrences in all real scales in time $O(nm^3 + n^2 m\log m)$ and space $O(nm^3 + n^2)$.*

The above result was improved.

Theorem 6 (Amir and Chencinski [7]) *Given an $n \times n$ text and $m \times m$ pattern. It is possible to find all pattern occurrences in all real scales in time $O(n^2 m)$ and space $O(n^2)$.*

This algorithm achieves its time by exploiting geometric characteristics of nested scales occurrences and a sophisticated use of dueling [1, 16].

The assumption in both above algorithms is that the scaled occurrence of the pattern starts at the top left corner of some pixel.

It turns out that one can achieve faster times in the *one dimensional* real scaled matching problem, even in the geometric model.

Theorem 7 (Amir, Butman, Lewenstein, Porat, and Tsur [5]) *Given a text string T of length n and a pattern string P of length m, there exists an $O(n\log m + m\sqrt{nm\log m})$ worst-case time solution to the one dimensional pattern matching problem with real scales in the geometric model.*

Applications

The problem of finding approximate occurrences of a template in an image is a central one in digital libraries and web searching. The current

algorithms to solve this problem use methods of computer vision and computational geometry. They model the image in another space and seek a solution there. A deterministic worst-case algorithm in pixel-level images does not yet exist. Yet, such an algorithm could be useful, especially in raw data that has not been modeled, e.g., movies. The work described here advances another step toward this goal from the scaling point of view.

Open Problems

Finding all scaled occurrences without fixing the scaled pattern start at the top left corner of the text pixel would be important from a practical point of view. The final goal is an integration of scaling with rotation [2, 11–13] and local errors (edit distance) [9].

Cross-References

- ▸ Multidimensional Compressed Pattern Matching
- ▸ Multidimensional String Matching

Recommended Reading

1. Amir A, Benson G, Farach M (1994) An alphabet independent approach to two dimensional pattern matching. SIAM J Comput 23(2):313–323
2. Amir A, Butman A, Crochemore M, Landau GM, Schaps M (2004) Two-dimensional pattern matching with rotations. Theor Comput Sci 314(1–2):173–187
3. Amir A, Butman A, Lewenstein M (1999) Real scaled matching. Inf Process Lett 70(4):185–190
4. Amir A, Butman A, Lewenstein M, Porat E (2003) Real two dimensional scaled matching. In: Proceedings of the 8th workshop on algorithms and data structures (WADS '03), pp 353–364
5. Amir A, Butman A, Lewenstein M, Porat E, Tsur D (2004) Efficient one dimensional real scaled matching. In: Proceedings of the 11th symposium on string processing and information retrieval (SPIRE'04), pp 1–9
6. Amir A, Calinescu G (2000) Alphabet independent and dictionary scaled matching. J Algorithm 36: 34–62
7. Amir A, Chencinski E (2006) Faster two dimensional scaled matching. In: Proceedings of the 17th symposium on combinatorial pattern matching (CPM). LNCS, vol 4009. Springer, Berlin, pp 200–210
8. Amir A, Farach M (1992) Two dimensional dictionary matching. Inf Process Lett 44:233–239
9. Amir A, Landau G (1991) Fast parallel and serial multidimensional approximate array matching. Theor Comput Sci 81:97–115
10. Amir A, Landau GM, Vishkin U (1992) Efficient pattern matching with scaling. J Algorithm 13(1):2–32
11. Amir A, Tsur D, Kapah O (2004) Faster two dimensional pattern matching with rotations. In: Proceedings of the 15th annual symposium on combinatorial pattern matching (CPM '04), pp 409–419
12. Fredriksson K, Mäkinen V, Navarro G (2004) Rotation and lighting invariant template matching. In: Proceedings of the 6th Latin American symposium on theoretical informatics (LATIN'04). LNCS, pp 39–48
13. Fredriksson K, Navarro G, Ukkonen E (2002) Optimal exact and fast approximate two dimensional pattern matching allowing rotations. In: Proceedings of the 13th annual symposium on combinatorial pattern matching (CPM 2002). LNCS, vol 2373, pp 235–248
14. Fredriksson K, Ukkonen E (1998) A rotation invariant filter for two-dimensional string matching. In: Proceedings of the 9th annual symposium on combinatorial pattern matching (CPM). LNCS, vol 1448. Springer, Berlin, pp 118–125
15. Landau GM, Vishkin U (1994) Pattern matching in a digitized image. Algorithmica 12(3/4):375–408
16. Vishkin U (1985) Optimal parallel pattern matching in strings. In: Proceedings of the 12th ICALP, pp 91–113

Two-Interval Pattern Problems

Stéphane Vialette
IGM-LabInfo, University of Paris-East, Descartes, France

Keywords

2-intervals; RNA structures

Years and Authors of Summarized Original Work

2004; Vialette
2007; Cheng, Yang, Yuan

Problem Definition

The problem is concerned with finding large constrained patterns in sets of 2-intervals. Given

a single-stranded RNA molecule, a sequence of contiguous bases of the molecule can be represented as an interval on a single line, and a possible pairing between two disjoint sequences can be represented as a 2-interval, which is merely the union of two disjoint intervals. Derived from arc-annotated sequences, 2-interval representation considers thus only the bonds between the bases and the pattern of the bonds, such as hairpin structures, knots and pseudoknots. A maximum cardinality disjoint subset of a candidate set of 2-intervals restricted to certain prespecified geometrical constraints can provide a useful valid approximation for RNA secondary structure determination.

The geometric properties of 2-intervals provide a possible guide for understanding the computational complexity of finding structured patterns in RNA sequences. Using a model to represent nonsequential information allows us to vary restrictions on the complexity of the pattern structure. Indeed, two disjoint 2-intervals, i.e., two 2-intervals that do not intersect in any point, can be in precedence order ($<$), be allowed to nest ($\sqsubset$) or be allowed to cross ($\between$). Furthermore, the set of 2-intervals and the pattern can have different restrictions, e.g., all intervals have the same length or all the intervals are disjoint. These different combinations of restrictions alter the computational complexity of the problems, and need to be examined separately. This examination produces efficient algorithms for more restrictive structured patterns, and hardness results for those that are less restrictive.

Notations

Let $I = [a, b]$ be an interval on the line. Write $\mathsf{start}(I) = a$ and $\mathsf{end}(I) = b$. A *2-interval* is the union of two disjoint intervals defined over a single line and is denoted by $D = (I, J)$; I is completely to the left of J. Write $\mathsf{left(D)} = \mathsf{I}$ and $\mathsf{right(D)} = \mathsf{J}$. Two 2-intervals $D_1 = (I_1, J_1)$ and $D_2 = (I_2, J_2)$ are said to be *disjoint* (or *nonintersecting*) if both 2-intervals share no common point, i.e., $(I_1 \cup J_1) \cap (I_2 \cup J_2) = \emptyset$. For such disjoint pairs of 2-intervals, three natural binary relations, denoted $<$, $\sqsubset$ and $\between$, are of special interest:

- $D_1 < D_2$ (D_1 *precedes* D_2), if $I_1 < J_1 < I_2 < J_2$,
- $D_1 \sqsubset D_2$ (D_1 is *nested in* D_2), if $I_2 < I_1 < J_1 < J_2$, and
- $D_1 \between D_2$ (D_1 *crosses* D_2), if $I_1 < I_2 < J_1 < J_2$.

A pair of 2-intervals D_1 and D_2 is said to be *R-comparable* for some $R \in \{<, \sqsubset, \between\}$, if either $D_1 R D_2$ or $D_2 R D_1$. Note that any two disjoint 2-intervals are R-comparable for some $R \in \{<, \sqsubset, \between\}$. A set of disjoint 2-intervals $\mathcal{D}$ is said to be *$\mathcal{R}$-comparable* for some $\mathcal{R} \subseteq \{<, \sqsubset, \between\}$, $\mathcal{R} \neq \emptyset$, if any pair of distinct 2-intervals in $\mathcal{D}$ is R-comparable for some $R \in \mathcal{R}$. The nonempty subset $\mathcal{R}$ is called a *model* for $\mathcal{D}$.

The 2-interval-pattern problem asks one to find in a set of 2-intervals a largest subset of pairwise compatible 2-intervals. In the present context, compatibility denotes the fact that any two 2-intervals in the solution are (1) nonintersecting and (2) satisfy some prespecified geometrical constraints. The 2-interval-pattern problem is formally defined as follows:

Problem 1 (2-interval-pattern)
INPUT: A set of 2-intervals $\mathcal{D}$ and a model $\mathcal{R} \subseteq \{<, \sqsubset, \between\}$.
SOLUTION: A $\mathcal{R}$-comparable subset $\mathcal{D}' \subseteq \mathcal{D}$.
MEASURE: The size of the solution, i.e., $|\mathcal{D}'|$.

According to the above definition, any solution for the 2-interval-pattern problem for some model $\mathcal{R} \subseteq \{<, \sqsubset, \between\}$ corresponds to an RNA structure constrained by $\mathcal{R}$. For example, a solution for the 2-interval-pattern problem for the $\mathcal{R} = \{<, \sqsubset\}$ model corresponds to a pseudoknot-free structure (a *pseudoknot* in an RNA sequence $S = s_1, s_2, \ldots, s_n$ is composed of two interleaving nucleotide pairings (s_i, s_j) and $(s_{i'}, s_{j'})$ such that $i < i' < j < j'$).

Some additional definitions are needed for further algorithmic analysis. Let $\mathcal{D}$ be a set of 2-intervals. The *width* (respectively *height*, *depth*) is the size of a maximum cardinality $\{<\}$-comparable (respectively $\{\sqsubset\}$-comparable, $\{\between\}$-comparable) subset $\mathcal{D}' \subseteq \mathcal{D}$. The *interleaving distance* of a 2-interval $D_i \in \mathcal{D}$ is defined to

be the distance between the two intervals of D_i, i.e., $\mathsf{start}(\mathsf{right}(\mathsf{D_i})) - \mathsf{end}(\mathsf{left}(\mathsf{D_i}))$. The *total interleaving distance* of the set of 2-intervals $\mathcal{D}$, written $\mathcal{L}(\mathcal{D})$, is the sum of all interleaving distances, i.e., $\mathcal{L}(\mathcal{D}) = \sum_{D_i \in \mathcal{D}} \mathsf{start}(\mathsf{right}(\mathsf{D_i})) - \mathsf{end}(\mathsf{left}(\mathsf{D_i}))$. The *interesting coordinates* of $\mathcal{D}$ are defined to be the set $X(\mathcal{D}) = \bigcup_{D_i \in \mathcal{D}} \{\mathsf{end}(\mathsf{left}(\mathsf{D_i})), \mathsf{start}(\mathsf{right}(\mathsf{D_i}))\}$. The *density* of $\mathcal{D}$, written $d(\mathcal{D})$, is the maximum number of 2-intervals in $\mathcal{D}$ over a single point. Formally, $d(\mathcal{D}) = \max_{x \in X(\mathcal{D})} \{D \in \mathcal{D} : \mathsf{end}(\mathsf{left}(\mathsf{D})) \leq \mathsf{x} < \mathsf{start}(\mathsf{right}(\mathsf{D}))\}$.

Constraints

The structure of the set of all (simple) intervals involved in a set of 2-intervals $\mathcal{D}$ turns out to be of particular importance for algorithmic analysis of the 2-interval-pattern problem. The *interval ground set* of $\mathcal{D}$, denoted $\mathcal{I}(\mathcal{D})$, is the set of all intervals involved in $\mathcal{D}$, i.e., $\mathcal{I}(\mathcal{D}) = \{\mathsf{left}(\mathsf{D_i}) : \mathsf{D_i} \in \mathcal{D}\} \cup \{\mathsf{right} < (\mathsf{D} : \mathsf{i}) : \mathsf{D_i} \in \mathcal{D}\}$. In [7, 20], four types of interval ground sets were introduced.

1. *Unlimited*: no restriction on the structure.
2. *Balanced*: each 2-interval $D_i \in \mathcal{D}$ is composed of two intervals having the same length, i.e., $|\mathsf{left}(\mathsf{D_i})| = |\mathsf{right}(\mathsf{D_i})|$.
3. *Unit*: the interval ground set $\mathcal{I}(\mathcal{D})$ is solely composed of unit length intervals.
4. *Disjoint*: no two distinct intervals in the interval ground set $\mathcal{I}(\mathcal{D})$ intersect.

Observe that a unit 2-interval set is balanced, while the converse is not necessarily true. Furthermore, for most applications, one may assume that a disjoint 2-interval set is unit. Observe that in this latter case, a set of 2-intervals reduces to a graph $G = (V, E)$ equipped with a numbering of its vertices from 1 to $|V|$, and hence the 2-interval-pattern problem for disjoint interval ground sets reduces to finding a constrained maximum matching in a linear graph. Considering additional restrictions such as:

- Bounding the width, the height or the depth of either the input set of 2-intervals or the solution subset
- Bounding the interleaving distances

is also of interest for practical applications.

Key Results

The different combinations of the models and interval ground sets alter the computational complexity of the 2-interval-pattern problem. The main results are summarized in Tables 1 (time complexity and hardness) and 2 (approximation for hard instances).

Theorem 1 *The 2-interval-pattern problem is approximable (APX) hard for models $\mathcal{R} = \{<, \sqsubset, \between\}$ and $\mathcal{R} = \{\sqsubset, \between\}$, and is nondeterministic polynomial-time (NP) complete – in its natural decision version – for model $\mathcal{R} = \{<, \between\}$, even when restricted to unit interval ground sets.*

Notice here that the 2-interval-pattern problem for model $\mathcal{R} = \{<, \between\}$ is not APX-hard. Two hard cases of the 2-interval-pattern turn out to be polynomial-time-solvable when restricted to disjoint-interval ground sets.

Theorem 2 *The 2-interval-pattern problem for a disjoint-interval ground set is solvable in*

- *$O(n\sqrt{n})$ time for model $\mathcal{R} = \{<, \sqsubset, \between\}$ (trivial reduction to the standard maximum matching problem)*

Two-Interval Pattern Problems, Table 1 Complexity of the 2-interval-pattern problem for all combinations of models and interval ground sets. For the polynomial-time cases, $n = |\mathcal{D}|$, $\mathcal{L} = \mathcal{L}(\mathcal{D})$ and $d = d(\mathcal{D})$

	Interval ground set $\mathcal{I}(\mathcal{D})$	
Model $\mathcal{R}$	Unlimited, balanced, unit	Disjoint
$\{<, \sqsubset, \between\}$	APX-hard [1]	$O(n\sqrt{n})$ [15]
$\{<, \between\}$	NP-complete [3]	**unknown**
$\{\sqsubset, \between\}$	APX-hard [19]	$O(n \log n + \mathcal{L})$ [8]
$\{<, \sqsubset\}$	$O(n \log n + nd)$ [8]	
$\{<\}$	$O(n \log n)$ [19]	
$\{\sqsubset\}$	$O(n \log n)$ [3]	
$\{\between\}$	$O(n \log n + \mathcal{L})$ [8]	

- $O(n \log n + \mathcal{L})$ *time for model* $\mathcal{R} = \{\sqsubset, \between\}$

The complexity of the 2-interval-pattern problem for model $\mathcal{R} = \{<, \between\}$ and a disjoint-interval ground set is still unknown. Three cases of the 2-interval-pattern problem are polynomial-time-solvable, regardless of the structure of the interval ground sets.

Theorem 3 *The 2-interval-pattern problem is solvable in*

- $O(n \log n + nd)$ *time for model* $\mathcal{R} = \{<, \sqsubset\}$
- $O(n \log n)$ *time for models* $\mathcal{R} = \{<\}$ *and* $\mathcal{R} = \{\sqsubset\}$
- $O(n \log n + \mathcal{L})$ *time for model* $\mathcal{R} = \{\between\}$

One may now turn to approximating hard instances of the 2-interval-pattern problem. Surprisingly enough, no significant differences (in terms of approximation guarantees) have yet been found for the 2-interval-pattern problem between the model $\mathcal{R} = \{<, \sqsubset, \between\}$ and the model $\mathcal{R} = \{\sqsubset, \between\}$ (the approximation algorithms are, however, different).

Theorem 4 *The 2-interval-pattern problem for model* $\mathcal{R} = \{<, \sqsubset, \between\}$ *or model* $\mathcal{R} = \{\sqsubset, \between\}$ *is approximable within ratio*

- *4 for unlimited-interval ground sets, and*
- $2 + \epsilon$ *for unit-interval ground sets.*

The 2-interval-pattern problem for model $\mathcal{R} = \{<, \between\}$ *is approximable within ratio* $1 + 1/\epsilon$, $\epsilon \geq 2$ *for all models.*

A practical 3-approximation algorithm for model $\mathcal{R} = \{<, \sqsubset, \between\}$ (resp. $\mathcal{R} = \{\sqsubset, \between\}$) and unit interval ground set that runs in $\mathcal{O}(n \lg n)$ (resp. $\mathcal{O}(n^2 \lg n)$) time has been proposed in [1] (resp. [7]). For model $\mathcal{R} = \{<, \between\}$, a more practical 2-approximation algorithm that runs in $\mathcal{O}(n^3 \lg n)$ time has been proposed in [10]. Notice that Theorem 4 holds true for the weighted version of the 2-interval-pattern problem [7] except for models $\mathcal{R} = \{<, \sqsubset, \between\}$ and $\mathcal{R} = \{\sqsubset, \between\}$ and unit interval ground set where the best approximation ratio is $2.5 + \epsilon$ [5].

Applications

Sets of 2-intervals can be used for modeling stems in RNA structures [20, 21], determining DNA sequence similarities [13] or scheduling jobs that are given as groups of nonintersecting segments in the real line [1, 9]. In all these applications, one is concerned with finding a maximum cardinality subset of nonintersecting 2-intervals. Some other classical combinatorial problems are also of interest [5]. Also, considering sets of t-intervals (each element is the union of at most t disjoint intervals) and their corresponding intersection graph has proved to be useful.

It is computationally challenging to predict RNA structures including pseudoknots [14]. Practical approaches to cope with intractability are either to restrict the class of pseudoknots under consideration [18] or to use heuristics [6, 17, 19]. The general problem of establishing a general representation of structured patterns, i.e., *macroscopic describers* of RNA structures, was considered in [20]. Sets of 2-intervals provide such a natural geometric description.

Constructing a relevant 2-interval set from a RNA sequence is relatively easy: stable stems are selected, usually according to a simplified thermodynamic model without accounting for loop energy [2, 16, 19–21]. Predicting a reliable RNA structure next reduces to finding a maximum subset of nonconflicting 2-intervals, i.e., a subset of disjoint 2-intervals. Considering in addition a model $\mathcal{R} \subseteq \{<, \sqsubset, \between\}$ allows us to vary restrictions on the complexity of the pattern structure. In [21], the treewidth of the intersection graph of the set of 2-intervals is considered for speeding up the computation.

For sets of 2-intervals involved in practical applications, restrictions on the interval ground set are needed. Unit interval ground sets were considered in [7]. Of particular importance in the context of molecular biology (RNA structures

Two-Interval Pattern Problems, Table 2 Performance ratios for hard instances of the 2-interval-pattern problem. LP stands for *Linear Programming* and N/A stands for *Not Applicable*

	Interval ground set $\mathcal{I}(\mathcal{D})$			
Model $\mathcal{R}$	Unlimited	Balanced	Unit	Disjoint
$\{<, \sqsubset, \between\}$	4 LP [1]	4 $\mathcal{O}(n \lg n)$ [7]	$2 + \epsilon\mathcal{O}(n^2 + n^{\mathcal{O}(\log 1/\epsilon)})$ [13]	N/A
$\{\sqsubset, \between\}$	4 LP [7]	4 $\mathcal{O}(n^2 \lg n)$ [7]	$2 + \epsilon\mathcal{O}(n^2 + n^{\mathcal{O}(\log 1/\epsilon)})$ [13]	N/A
$\{<, \between\}$	$1 + 1/\epsilon\mathcal{O}(n^{2\epsilon+3})$, $\epsilon \geq 2$ [14]			

and DNA sequence similarities) are balanced interval ground sets, where each 2-interval is composed of two equally length intervals.

Open Problems

A number of problems related to the 2-interval-pattern problem remain open. First, improving the approximation ratios for the various flavors of the 2-interval-pattern problem is of particular importance. For example, the existence of a fast approximation algorithm with good performance guarantee for the 2-interval-pattern problem for model $\mathcal{R} = \{<, \sqsubset, \between\}$ remains an apparently challenging open problem. A related open research area is concerned with balanced-interval ground sets. In particular, no evidence has shown yet that the 2-interval-pattern problem becomes easier to approximate for balanced-interval ground sets. This question is of special importance in the context of RNA structures where most 2-intervals are balanced.

A number of important question are still open for model $\mathcal{R} = \{<, \between\}$. First, it is still unknown whether the 2-interval-pattern problem for disjoint-interval ground sets and model $\mathcal{R} = \{<, \between\}$ is polynomial-time-solvable. Observe that this problem trivially reduces to the following graph problem: Given a graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$, find a maximum cardinality matching $\mathcal{M} \subseteq E$ such that for any two distinct edges $\{i, j\}$ and $\{k, l\}$ of $\mathcal{M}$, $i < j$, $k < l$ and $i < k$, either $j < k$ or $j < l$. Another open question concerns the approximation of the 2-interval-pattern problem for balanced interval ground set. Is this special case better approximable than the general case?

A last direction of research is concerned with the parameterized complexity of the 2-interval-pattern problem. For example, it is not known whether the 2-interval-pattern problem for models $\mathcal{R} = \{<, \sqsubset, \between\}$, $\mathcal{R} = \{\sqsubset, \between\}$ or $\mathcal{R} = \{<, \between\}$ is fixed-parameter-tractable when parameterized by the size of the solution. Also, investigating the parameterized complexity for parameters such as the maximum number of pairwise crossing intervals in the input set or the treewidth of the corresponding intersection 2-interval graph, which are expected to be relatively small for most practical applications, is of particular interest.

Cross-References

- RNA Secondary Structure Prediction by Minimum Free Energy
- RNA Secondary Structure Prediction Including Pseudoknots

Recommended Reading

1. Bar-Yehuda R, Halldorsson M, Naor J, Shachnai H, Shapira I (2002) Scheduling split intervals. In: Proceedings of the 13th annual ACM-SIAM symposium on discrete algorithms (SODA), pp 732–741
2. Billoud B, Kontic M, Viari A (1996) Palingol a declarative programming language to describe nucleic acids' secondary structures and to scan sequence database. Nucleic Acids Res 24:1395–1403
3. Blin G, Fertin G, Vialette S (2007) Extracting 2-intervals subsets from 2-interval sets. Theor Comput Sci 385(1–3):241–263
4. Blin G, Fertin G, Vialette S (2004) New results for the 2-interval pattern problem. In: Proceedings of the 15th annual symposium on combinatorial pattern matching (CPM). Lecture notes in computer science, vol 3109. Springer, Berlin
5. Butman A, Hermelin D, Lewenstein M, Rawitz D (2007) Optimization problems in multiple-interval

graphs. In: Proceedings of the 9th annual ACM-SIAM symposium on discrete algorithms (SODA), ACM-SIAM 2007, pp 268–277
6. Chen J-H, Le S-Y, Maize J (2000) Prediction of common secondary structures of RNAs: a genetic algorithm approach. Nucleic Acids Res 28:991–999
7. Crochemore M, Hermelin D, Landau G, Rawitz D, Vialette S (2008) Approximating the 2-interval pattern problem. Theor Comput Sci (special issue for Alberto Apostolico)
8. Erdong C, Linji Y, Hao Y (2007) Improved algorithms for 2-interval pattern problem. J Comb Optim 13(3):263–275
9. Halldorsson M, Karlsson R (2006) Strip graphs: recognition and scheduling. In: Proceedings of the 32nd international workshop on graph-theoretic concepts in computer science (WG). Lecture notes in computer science, vol 4271. Springer, Berlin, pp137–146
10. Jiang M (2007) A 2-approximation for the preceding-and-crossing structured 2-interval pattern problem. J Comb Optim 13:217–221
11. Jiang M (2007) Improved approximation algorithms for predicting RNA secondary structures with arbitrary pseudoknots. In: Proceedings of the 3rd international conference on algorithmic aspects in information and management (AAIM), Portland. Lecture notes in computer science, vol 4508. Springer, pp 399–410
12. Jiang M (2007) A PTAS for the weighted 2-interval pattern problem over the preceding-and-crossing model. In: Xu Y, Dress AWM, Zhu B (eds) Proceedings of the 1st annual international conference on combinatorial optimization and applications (COCOA), Xi'an. Lecture notes in computer science, vol 4616. Springer, pp 378–387
13. Joseph D, Meidanis J, Tiwari P (1992) Determining DNA sequence similarity using maximum independent set algorithms for interval graphs. In: Proceedings of the 3rd Scandinavian workshop on algorithm theory (SWAT). Lecture notes in computer science. Springer, Berlin, pp 326–337
14. Lyngsø R, Pedersen C: RNA pseudoknot prediction in energy-based models. J Comput Biol 7:409–427 (2000)
15. Micali S, Vazirani V (1980) An O(sqrt|V||E|) algorithm for finding maximum matching in general graphs. In: Proceedings of the 21st annual symposium on foundation of computer science (FOCS). IEEE, pp 17–27
16. Nussinov R, Pieczenik G, Griggs J, Kleitman D (1978) Algorithms for loop matchings. SIAM J Appl Math 35:68–82
17. Ren J, Rastegart B, Condon A, Hoos H (2005) HotKnots: heuristic prediction of rna secondary structure including pseudoknots. RNA 11:1194–1504
18. Rivas E, Eddy S (1999) A dynamic programming algorithm for RNA structure prediction including pseudoknots. J Mol Biol 285:2053–2068
19. Ruan J, Stormo G, Zhang W (2004) An iterated loop matching approach to the prediction of RNA secondary structures with pseudoknots. Bioinformatics 20:58–66
20. Vialette S (2004) On the computational complexity of 2-interval pattern matching. Theor Comput Sci 312:223–249
21. Zhao J, Malmberg R, Cai L (2006) Rapid ab initio RNA folding including pseudoknots via graph tree decomposition. In: Proceedings of the workshop on algorithms in bioinformatics. Lecture notes in computer science, vol 4175. Springer, Berlin, pp 262–273

U

Undirected Feedback Vertex Set

Jiong Guo
Department of Mathematics and Computer Science, University of Jena, Jena, Germany

Keywords

Odd cycle transversal

Years and Authors of Summarized Original Work

2005; Dehne, Fellows, Langston, Rosamond, Stevens Guo, Gramm, Hüffner, Niedermeier, Wernicke

Problem Definition

The UNDIRECTED FEEDBACK VERTEX SET (UFVS) problem is defined as follows:

Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.

Task: Find a *feedback vertex set* $F \subseteq V$ with $|F| \leq k$ such that each cycle in G contains at least one vertex from F. (The removal of all vertices in F from G results in a forest.)

Karp [11] showed that UFVS is NP-complete. Lund and Yannakakis [12] proved that there exists some constant $\epsilon > 0$ such that it is NP-hard to approximate the optimization version of UFVS to within a factor of $1 + \epsilon$. The best-known polynomial-time approximation algorithm for UFVS has a factor of 2 [1, 4]. There is a simple and elegant randomized algorithm due to Becker et al. [3] which solves UFVS in $O(c{\cdot}4^k{\cdot}k\,n)$ time on an n-vertex and m-edge graph by finding a feedback vertex set of size k with probability at least $1 - (1 - 4^{-k})^{c4^k}$ for an arbitrary constant c. An exact algorithm for UFVS with a running time of $O(1.7548^n)$ was recently found by Fomin et al. [9]. In the context of parameterized complexity [8, 13], Bodlaender [5] and Downey and Fellows [7] were the first to show that the problem is *fixed-parameter tractable*, i.e., that the combinatorial explosion when solving it can be confined to the parameter k. The currently best fixed-parameter algorithm for UFVS runs in $O(c^k{\cdot}mn)$ for a constant c [6, 10] (see [6] for the so far best running time analysis leading to a constant $c = 10.567$). This algorithm is the subject of this entry.

Key Results

The $O(c^k{\cdot}mn)$-time algorithm for the UNDIRECTED FEEDBACK VERTEX SET is based on the so-called "iterative compression" technique, which was introduced by Reed et al. [14]. The central observation of this technique is quite simple but fruitful: To derive a fixed-parameter algorithm for a minimization problem, it suffices to give a fixed-parameter "compression routine"

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

that, given a size-$(k+1)$ solution, either proves that there is no size-k solution or constructs one. Starting with a trivial instance and iteratively applying this compression routine a linear number of rounds to larger instances, one obtains a fixed-parameter algorithm of the problem. The main challenge of applying this technique to UFVS lies in showing that there is a fixed-parameter compression routine.

The compression routine from [6, 10] works as follows:

1. Consider all possible partitions (X, Y) of the size-$(k+1)$ feedback vertex set F with $|X| \leq k$ under the assumption that set X is entirely contained in the new size-k feedback vertex set F' and $Y \cap F' = \emptyset$
2. For each partition (X, Y), if the vertices in Y induce cycles, then answer "no" for this partition; otherwise, remove the vertices in X. Moreover, apply the following data reduction rules to the remaining graph:
   - Remove degree-1 vertices.
   - If there is a degree-2 vertex v with two neighbors v_1 and v_2, where $v_1 \notin Y$ or $v_2 \notin Y$, then remove v and connect v_1 and v_2. If this creates two parallel edges between v_1 and v_2, then remove the vertex of v_1 and v_2 that is not in Y and add it to any feedback vertex set for the reduced instance.

   Finally, exhaustively examine every vertex set S with size at most $k - |X|$ of the reduced graph as to whether S can be added to X to form a feedback vertex set of the input graph. If there is one such vertex set, then output it together with X as the new size-k feedback vertex set.

The correctness of the compression routine follows from its brute-force nature and the easy to prove correctness of the two data reduction rules. The more involved part is to show that the compression routine runs in $O(c^k \cdot m)$ time: There are 2^k+1 partitions of F into the above sets (X, Y) and one can show that, for each partition, the reduced graph after performing the data reduction rules has at most $d \cdot k$ vertices for a constant d; otherwise, there is no size-k feedback vertex set for this partition. This then gives the $O(c^k \cdot m)$-running time. For more details on the proof of the $d \cdot k$-size bound see [6, 10].

Given as input a graph G with vertex set $\{v_1, \ldots, v_n\}$, the fixed-parameter algorithm from [6, 10] solves UFVS by iteratively considering the subgraphs $G_i := G[\{v_1, \ldots, v_i\}]$. For $i = 1$, the optimal feedback vertex set is empty. For $i > 1$, assume that an optimal feedback vertex set X_i for G_i is known. Obviously, $X_i \cup \{v_{i+1}\}$ is a solution set for G_{i+1}. Using the compression routine, the algorithm can in $O(c^k \cdot m)$ time either determine that $X_i \cup \{v_{i+1}\}$ is an optimal feedback vertex set for G_{i+1}, or, if not, compute an optimal feedback vertex set for G_{i+1}. For $i = n$, we thus have computed an optimal feedback vertex set for G in $O(c^k \cdot mn)$ time.

Theorem 1 Undirected Feedback Vertex Set *can be solved in $O(c^k \cdot mn)$ time for a constant c.*

Applications

The Undirected Feedback Vertex Set is of fundamental importance in combinatorial optimization. One typical application, for example, appears in the context of combinatorial circuit design [1]. For applications in the areas of constraint satisfaction problems and Bayesian inference, see Bar-Yehuda et al. [2].

Open Problems

It is open to explore the practical performance of the described algorithm. Another research direction is to improve the running time bound given in Theorem 1. Finally, it remains a long-standing open problem whether the Feedback Vertex Set on *directed* graphs

is fixed-parameter tractable. The answer to this question would represent a significant breakthrough in the field.

Acknowledgments Supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4

Recommended Reading

1. Bafna V, Berman P, Fujito T (1999) A 2-approximation algorithm for the undirected feedback vertex set problem. SIAM J Discret Math 3(2):289–297
2. Bar-Yehuda R, Geiger D, Naor J, Roth RM (1998) Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. SIAM J Comput 27(4): 942–959
3. Becker A, Bar-Yehuda R, Geiger D (2000) Randomized algorithms for the loop cutset problem. J Artif Intell Res 12:219–234
4. Becker A, Geiger D (1994) Approximation algorithms for the Loop Cutset problem. In: Proceedings of the 10th conference on uncertainty in artificial intelligence. Morgan Kaufman, San Fransisco, pp 60–68
5. Bodlaender HL (1994) On disjoint cycles. Int J Found Comput Sci 5(1):59–68
6. Dehne F, Fellows MR, Langston MA, Rosamond F Stevens K (2005) An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. In: Proceedings of the 11th COCOON. LNCS, vol 3595. Springer, Berlin, pp 859–869. Long version to appear in: J Discret Algorithm
7. Downey RG, Fellows MR (1992) Fixed-parameter tractability and completeness. Congr Numerant 87:161–187
8. Downey RG, Fellows MR (1999) Parameterized complexity. Springer, Heidelberg
9. Fomin FV, Gaspers S, Pyatkin AV (2006) Finding a minimum feedback vertex set in time $O(1.7548^n)$. In: Proceedings of the 2th IWPEC. LNCS, vol 4196. Springer, Berlin, pp 184–191
10. Guo J, Gramm J, Hüffner F, Niedermeier R, Wernicke S (2006) Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. J Comput Syst Sci 72(8):1386–1396
11. Karp R (1972) Reducibility among combinatorial problems. In: Miller R, Thatcher J (eds) Complexity of computer computations. Plenum Press, New York, pp 85–103
12. Lund C, Yannakakis M (1993) The approximation of maximum subgraph problems. In: Proceedings of the 20th ICALP. LNCS, vol 700. Springer, Berlin, pp 40–51
13. Niedermeier R (2006) Invitation to fixed-parameter algorithms. Oxford University Press, Oxford
14. Reed B, Smith K, Vetta A (2004) Finding odd cycle transversals. Oper Res Lett 32(4):299–301

Unified View of Graph Searching and LDFS-Based Certifying Algorithms

Derek G. Corneil[1] and Michel Habib[2]
[1]Department of Computer Science, University of Toronto, Toronto, ON, Canada
[2]LIAFA, Université Paris Diderot, Paris Cedex 13, France

Keywords

BFS; Cocomparability graphs; DFS; Graph searching; Hamiltonian path; LBFS; LDFS; Minimum pathcover

Years and Authors of Summarized Original Work

2008; Corneil, Krueger
2013; Corneil, Dalton, Habib

Problem Definition

The notion of searching a graph, in particular visiting each vertex in a systematic preordained fashion, is as old as graph theory itself. Indeed, Euler's paper in 1736 [13] presented conditions on the vertex degrees of a graph that would certify the presence or absence of a path (or circuit) of edges visiting each edge exactly once. Later it was shown by Fleury that an easy algorithm to find such a path (or circuit) can be achieved using depth-first search (DFS) [14]. In the late nineteenth century, C. P. Trémaux [22] and G. Tarry [28] presented DFS-based algorithms for maze traversal; similarly, breadth-first search (BFS) algorithms were used to find the shortest possible successful maze traversals.

In the 1960s and 1970s, these searches were used in many of the early graph algorithms for problems such as distance and diameter determination, network flows, planar graph recognition, and connected and 2-connected components; see [4, 27]. A "generic" (GENS) search, as defined by Tarjan [27], is one in which an unvisited vertex adjacent to a visited vertex must be chosen before an arbitrary unvisited vertex. Note that this criterion includes the standard graph searches, but does not include some useful vertex orderings such as nonincreasing vertex degree. We caution the reader that by BFS we follow Golumbic [16] and refer to the distance layering where unvisited neighbors of the currently visited vertex are placed at the end of a queue data structure. Note that in [4] BFS is defined as distance layering, but their implementation of distance layering uses a queue. Throughout this note, we assume that our graphs are connected.

In a seminal 1976 paper, Rose, Tarjan, and Lueker [24] introduced a variation of BFS, called lexicographic breadth-first search (LBFS), and showed that an arbitrary LBFS search could be used to achieve a linear time algorithm to recognize chordal graphs (there is no induced cycle of size strictly greater than 3). After the appearance of this paper, there were a few new applications of LBFS, mostly on applications on graph families related to chordal graphs. In the 1990s there was a marked increase in the application of LBFS in which a previous vertex ordering (usually a previous LBFS ordering) was used to break ties when there was more than one vertex eligible to be visited next. Such tiebreaking is referred to as a "+-sweep," as defined below.

Definition 1 Given a search $\mathcal{S}$ and vertex ordering σ of graph G, a **plus $\mathcal{S}$ sweep with respect to** σ (denoted $\mathcal{S}^+(G,\sigma)$) is the vertex ordering where the next vertex to be visited is the rightmost (as ordered by σ) T vertex (where T denotes the set of tied vertices).

Such "multi-sweep" LBFS algorithms were used for the recognition of interval graphs (for definitions of, and basic results on, various graph classes mentioned in this paper, see [3]), unit interval graphs, and cographs as well as finding dominating pairs (a pair of vertices (x, y) such that every $x - y$ path P dominates G in the sense that every vertex of G is either on P or has a neighbor on P) for asteroidal triple-free graphs. See [5] for an overview of these applications of LBFS. More recently, applications of LBFS have been found for graphs in general for such problems as modular decomposition [29] and split decomposition [15].

The proofs of correctness of multi-sweep LBFS algorithms typically are based on the following "4-vertex ordering characterization of LBFS":

Theorem 1 ([2, 16]) *A vertex total ordering σ could be produced by an LBFS if and only if for every triple of vertices $\{a, b, c\}$ where $a <_\sigma b <_\sigma c, ac \in E$, and $ab \notin E$ there exists vertex d such that $d <_\sigma a, db \in E$, and $dc \notin E$.*

Having seen the importance of Theorem 1 to the development of multi-sweep LBFS algorithms, a natural question and the question that is the basis of the Graph Searching Paper [6] is:

> Do other standard graph searches have a similar "4-vertex ordering characterization"?

Key Results

The first reaction to the question posed above is to try to understand exactly what is the structure of the search imposed by the $\{a, b, c\}$ vertices. The relevant question is:

> In the presence of the ac edge, how could b have been visited before c?

Since we are dealing with "generic" searches, some vertex d which is adjacent to b must have been visited before b since otherwise a would have to be chosen before b. If the search we are considering does not impose any further conditions on which unvisited vertices are eligible to be chosen, then the existence of d with $db \in E$ and $d <_\sigma b$ is a "4-vertex ordering characterization of GENS search." The full statement of the main theorem proved in [6] is:

Theorem 2 ([6]) *For $\mathcal{S}$ a graph search in {GENS, BFS, DFS, MNS, LBFS, LDFS} a total ordering σ of the set of vertices of the given graph could have been produced by $\mathcal{S}$ if and only if for every triple $a <_\sigma b <_\sigma c$ in σ where $ac \in E, ab \notin E$ there exists vertex d satisfying the requirements stated in the following table:*

Search $\mathcal{S}$	Requirements on d	
	Location	Adjacencies
GENS	$d < b$	$db \in E$
BFS	$d < a$	$db \in E$
DFS	$a < d < b$	$db \in E$
MNS	$d < b$	$db \in E, dc \notin E$
LBFS	$d < a$	$db \in E, dc \notin E$
LDFS	$a < d < b$	$db \in E, dc \notin E$

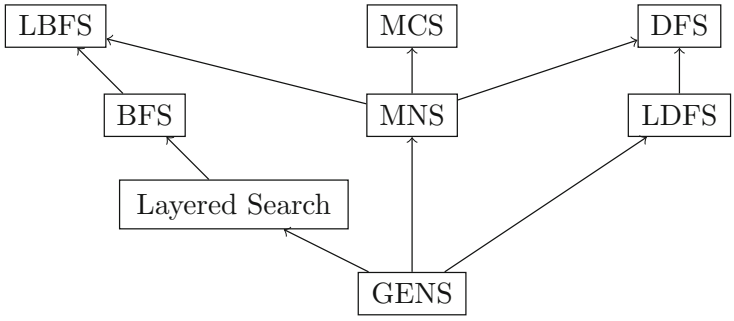
Note that the hierarchy among these different searches (and layered search) is shown in Fig. 1.

In the case of BFS, the characterization states that b must have a neighbor d where $d <_\sigma a$. In effect, this location for d reflects the role played by the queue in the BFS algorithm. Similarly, for DFS the location of d is between a and b reflecting the role played by the stack in the DFS algorithm. Note that the locations of d imposed by BFS and DFS capture the full range allowed by GENS search thereby exhibiting a type of duality between BFS and DFS.

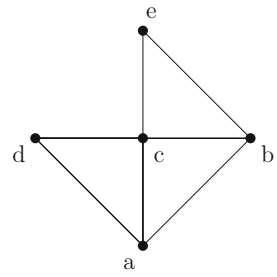
Now look at the difference between the characterizations of both BFS and LBFS. Both have the same location requirement for vertex d; however, BFS requires d to be a neighbor of b, whereas LBFS strengthens this condition so that d has to be a **private neighbor** of b with respect to c. This means that b's neighborhood in the set of visited vertices is **maximal with respect to set inclusion**. This raises the question of what happens to both DFS and GENS search if we add the "lexicographic" property – i.e., as with LBFS, that d has to be a private neighbor of b with respect to c, and thus that b's neighborhood in the set of visited vertices is maximal with respect to set inclusion. In the case of the "lexical" version of GENS search, this search was already known as the maximal neighbor search (MNS) [25], in particular, a search that chooses any vertex that has a maximal (by set inclusion) neighborhood in the set of visited vertices. Interestingly, this vertex ordering was presented in [24] where they showed that any search that obeys this property would produce a perfect elimination ordering (PEO) if the given graph is chordal. Thus, they concluded that both maximum cardinality search and LBFS suffice. Turning to LDFS, we see that d has the same location requirement as DFS, and adding the "lexical" property shows that LDFS is also a restricted version of MNS, and thus, it too is guaranteed to produce a PEO on chordal graphs. Note that in [6] all of these conditions are shown to be characterizations of the specific searches.

To illustrate the differences and relationships among these various searches, consider the graph in Fig. 2:

Regarding complexity issues, all searches mentioned in Theorem 2, except LDFS, have a linear time implementation (see [17] for LBFS); the current best LDFS implementation for arbitrary graphs uses van Emde Boas



Unified View of Graph Searching and LDFS-Based Certifying Algorithms, Fig. 1 Summary of the hereditary relationships proved in Theorem 2. An arc from search S to search S' indicates that S' is a restriction of S



$a\,b\,d\,c\,e \in BFS \setminus \{LBFS, DFS\}$

$a\,b\,c\,d\,e \in \{LBFS, DFS\} \setminus LDFS$

$a\,b\,e\,c\,d \in DFS \setminus \{LDFS, BFS\}$

$a\,b\,c\,e\,d \in LDFS \setminus BFS$

$c\,a\,b\,e\,d \in \{MNS \cap BFS\} \setminus LBFS$

$b\,a\,c\,e\,d \in \{MNS \cap DFS\} \setminus LDFS$

Unified View of Graph Searching and LDFS-Based Certifying Algorithms, Fig. 2 Sample graph and illustrative searches

trees [26] and runs in time $O(\max(n^2, n + m\log\log n))$. The key question arising from [6] is:

Are there any applications of LDFS?

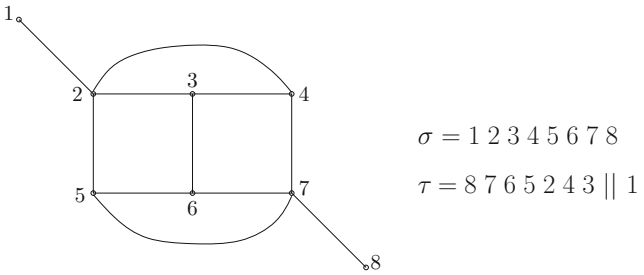
Problem Definition (cont.)

The first few attempts to find such an application quickly failed. The first was to see if LDFS could enjoy the same success as LBFS in building recognition algorithms for various restricted families of graphs (apart from chordal graphs); in all cases easily found counterexamples thwarted the various attempts. The second approach was to determine if LDFS$^+$ could be helpful in finding Hamilton paths (HP) or more generally minimum path covers (MPC) where the goal is to find a minimum cardinality set of subpaths of given graph G such that each vertex belongs to exactly one such path. Unfortunately, LDFS$^+$ fails when applied to an interval ordering (G is an interval graph if and only if there is an **interval ordering**, σ, of the vertices such that for all triples $a <_\sigma b <_\sigma c$ where $ac \in E$, then ab must also belong to E). To see this, consider a vertex universal to two disjoint paths on three vertices. From examples, it seems, however, that DFS$^+$ will find an HP, if one exists.

In fact, [1] and [11] independently showed that using the rightmost neighbor (RMN) sweep on an interval ordering yields an MPC of the given interval graph. Note that RMN when presented with an ordering σ greedily builds paths by starting at the rightmost unvisited vertex of σ and proceeding to its rightmost unvisited neighbor if such a vertex exists; if not, a new path is started at the rightmost unvisited vertex. (Note that this backtracking is different than the DFS$^+$ restarting.) Building off this algorithm, Dalton [10] presented a simple algorithm that certifies the correctness of the computed set of paths by either finding a set of vertices S (called a "scattering set") where the number of connected components of $G \setminus S$ equals $|S|$ plus the number of paths in the path cover or concludes that the given vertex ordering is not an interval ordering.

The next step was to try to lift this simple MPC algorithm to the superclass of cocomparability graphs. Note that a graph is a cocomparability graph if and only if its complement $\overline{G}$ has a transitive orientation of its edges. This orientation condition in $\overline{G}$ immediately translates into a vertex ordering characterization of cocomparability graphs. In particular, G is a cocomparability graph if and only if there is a **cocomp ordering**, σ, of the vertices such that for all triples $a <_\sigma b <_\sigma c$ where $ac \in E$, at least one of ab and bc must also belong to E.

Although there were polynomial time algorithms that solved the MPC problem on cocomparability graphs, all of these algorithms solved the "bump number" problem on the poset associated with the given graph and used the fact that any linear extension that minimizes the bump number contains the set of paths in a minimum path cover. The goal of this research was to find an MPC cocomparability graph algorithm that is directly graph theoretical and hopefully extends the interval graph MPC algorithm mentioned above. Examples immediately showed that applying RMN to an arbitrary cocomp ordering does not work, so many attempts were made to



Unified View of Graph Searching and LDFS-Based Certifying Algorithms, Fig. 3 LBFS counterexample

use multi-sweep LBFS to yield such a cocomp ordering. (Note that if σ is a cocomp ordering and $\tau = \text{LBFS}^+(\sigma)$, then τ is also a cocomp ordering.) This approach continued until the graph shown in Fig. 3 was discovered. On this graph every LBFS cocomp σ ordering fails, in the sense that RMN applied to σ does not produce a Hamiltonian path! A sample LBFS and the resulting RMN (which consists of two paths) are included in Fig. 3.

> Having seen the failure of LBFS, is there any chance that LDFS could work?

Key Results (cont.)

If there is such a role for LDFS, the counterexample for interval graphs mentioned previously shows that LDFS could not be expected to produce a minimum path cover itself; possibly LDFS could be used as a preprocessing step. If so, the simplest possible algorithm would be:

1. Let π be an arbitrary cocomp ordering.
2. Let σ be $\text{LDFS}^+(\pi)$.
3. Let τ be $\text{RMN}(\sigma)$.
4. If τ is not a Hamiltonian path, then from τ, use Dalton's algorithm to construct a separator S that certifies τ; otherwise, conclude π is not a cocomp ordering.

First of all, as with LBFS, LDFS when applied as a $+$-sweep on a given cocomp ordering returns a cocomp ordering. In this algorithm the hope is that an LDFS cocomp ordering would capture the "interval structure of cocomparability graphs," at least from the perspective of the MPC problem.

Somewhat surprisingly, this algorithm worked on all attempted examples. In an attempt to understand the structure exposed by an LDFS cocomp ordering, there are two points. First of all, why we do not see the LDFS structure in interval graphs? From the vertex ordering characterization of interval graphs, we see that there can never be an ordered triple of vertices $a < b < c$ with edge ac and nonedge ab, and thus, every interval ordering is simultaneously an example of every search mentioned in Theorem 2. Secondly, since an interval graph is chordal, every LBFS and LDFS must be a perfect elimination ordering implying that every vertex is simplicial in the graph formed on it and all vertices before it in the ordering. By considering a C_4, this property will not hold for LDFS cocomp orderings. There is however a crucial observation of the structure guaranteed by a nonsimplicial vertex in an LDFS cocomp ordering.

Lemma 1 ([7]) *Let σ be an LDFS cocomp ordering of cocomparability graph G. If z is a nonsimplicial vertex in σ as witnessed by $x <_\sigma y <_\sigma z$ where $xz, yz \in E, xy \notin E$, then there exists vertex w, $x <_\sigma w <_\sigma y$ where $xw, wy \in E, wz \notin E$.*

Proof By the LDFS vertex ordering characterization applied to the triple $\{x, y, z\}$, vertex w exists and satisfies all conditions of the lemma, except possibly $xw \in E$; if this is not the case, then the triple $\{x, w, z\}$ violates σ being a cocomp ordering.

This lemma plays a critical role in the proof of correctness of the MPC algorithm stated above.

Open Problems

We first list a number of recent results that have grown out of the work presented in [6, 7]:

- Köhler and Mouatadid [20] have recently shown that LDFS on a cocomparability graph can be done in linear time, thereby avoiding the log log$n$ off linear factor in the MPC paper [7].
- Mertzios and Corneil [23] have "lifted" the $O(n^4)$ longest path algorithm on interval graphs [18] to achieve the same result and time bound for cocomparability graphs. As with the MPC algorithm, a LDFS cocomp ordering was required.
- A similar technique of using an LDFS cocomparability ordering as a preprocessing step for a simple linear time interval graph algorithm has resulted in a linear time algorithm for the maximum independent set (and minimum vertex cover) problems on cocomparability graphs [8]. Note that the algorithm also produces a minimum cardinality clique cover in order to certify the maximum independent set produced by the algorithm. This algorithm also uses the linear time LDFS cocomp ordering algorithm presented in [20]. Very recently Köhler and Mouatadid [19] have presented a linear time algorithm that computes a maximum weighted independent set of a cocomparability graph; this algorithm works on any cocomp ordering and, in particular, does not require an LDFS cocomp ordering.
- In [8] the authors also characterized the search orderings that are "cocomp ordering preserving" in the sense that when used as a $+$-sweep, the output is a cocomp ordering when the input is a cocomp ordering. They showed that dfgreedy is such a preserving search and can be used to simplify the current best recognition algorithm for permutation graphs.
- In his PhD thesis, Dusart [12] studied the maximal clique lattice of a cocomparability graph and showed that a graph G is a cocomparability graph if and only if the set of maximal cliques of G satisfies specific lattice properties. Furthermore, he defined a new cocomp ordering preserving search called local MNS to compute a maximal interval subgraph of G. The new characterization together with MNS yields linear time algorithms to compute the simplicial vertices, the clique separators, and associated components of a cocomparability graph.
- Recently a new model of graph searching called "tiebreaking label search" (TBLS) [9] has been announced. This model builds off the vertex ordering characterization model appearing in [6] as well as the General Label Search formalism of Krueger, Simonet, and Berry [21]. The TBLS model incorporates the $+$-sweep use of graph searches, restricts labels to be sets of integers, and presents some new vertex ordering characterizations.

We now turn to some new directions for further research. From a graph algorithm perspective, the most interesting question is whether the results on cocomparability graphs can be easily extended to asteroidal triple-free graphs, an inclusive family that has received considerable attention. Further results, both structural and algorithmic are expected for cocomparability graphs and their associated posets. We expect that graph searching will continue to play a major role in these developments.

Recommended Reading

1. Arikati SR, Rangan CP (1990) Linear algorithm for optimal path cover problem on interval graphs. Inf Process Lett 35(3):149–153
2. Brandstädt A, Dragan FF, Nicolai F (1997) LexBFS-orderings and powers of chordal graphs. Discret Math 171(1–3):27–42
3. Brandstädt A, Le VB, Spinrad JP (1999) Graph classes, a survey. SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics, Philadelphia
4. Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms, 3rd edn. MIT, Cambridge
5. Corneil DG (2004) Lexicographic breadth first search – a survey. In: WG, Bad Honnef. Lecture notes in computer science, vol 3353. Springer, pp 1–19
6. Corneil DG, Krueger R (2008) A unified view of graph searching. SIAM J Discret Math 22(4): 1259–1276

7. Corneil DG, Dalton B, Habib M (2013) LDFS-based certifying algorithm for the minimum path cover problem on cocomparability graphs. SIAM J Comput 42(3):792–807
8. Corneil DG, Dusart J, Habib M, Kőhler E (2014) On the power of graph searching for cocomparability graphs. Submitted for publication
9. Corneil DG, Dusart J, Habib M, Mamcarz A, de Montgolfier F (2014) A new model for graph search. Under revision
10. Dalton B (2011) On minimum path cover in interval graphs. Master's thesis, University of Toronto
11. Damaschke P (1993) Paths in interval graphs and circular arc graphs. Discret Math 112(1–3):49–64
12. Dusart J (2014) Graph searches with applications to cocomparability graphs. PhD thesis, University of Paris Diderot
13. Euler L (1736) Solutio problematis ad geometriam situs pertinentis. Comment Academiae Sci I Petropolitanae 8:128–140
14. Fleury (1883) Deux problèmes de géométrie de situation. Journal de mathématiques élémentaires 257–261
15. Gioan E, Paul C, Tedder M, Corneil DG (2014) Practical and efficient split-decomposition via graph-labelled trees. Algorithmica 69(4): 789–843
16. Golumbic MC (2004) Algorithmic graph theory and perfect graphs. Annals of discrete mathematics, vol 57. Elsevier, Amsterdam/Boston
17. Habib M, McConnell RM, Paul C, Viennot L (2000) LexBFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. Theor Comput Sci 234(1–2):59–84. http://dblp.uni-trier.de/rec/bibtex/journals/tcs/HabibMPV00
18. Ioannidou K, Mertzios GB, Nikolopoulos SD (2011) The longest path problem has a polynomial solution on interval graphs. Algorithmica 6(2):320–341
19. Köhler E, Mouatadid L (2014) A linear time algorithm for computing a maximum weight independent set on cocomparability graphs. Submitted for publication
20. Köhler E, Mouatadid L (2014) Linear time LDFS on cocomparability graphs. In: SWAT, Copenhagen, pp 319–330
21. Krueger R, Simonet G, Berry A (2011) A general label search to investigate classical graph search algorithms. Discret Appl Math 159(2–3): 128–142
22. Lucas E (1882) Récréations mathématiques. Gauthier-Vilars, Paris
23. Mertzios GB, Corneil DG (2012) A simple polynomial algorithm for the longest path problem on cocomparability graphs. SIAM J Discret Math 26(3):940–963. http://dblp.uni-trier.de/rec/bibtex/journals/siamdm/MertziosC12
24. Rose DJ, Tarjan RE, Lueker GS (1976) Algorithmic aspects of vertex elimination on graphs. SIAM J Comput 5(2):266–283. http://dblp.uni-trier.de/rec/bibtex/journals/siamcomp/RoseTL76
25. Shier DR (1984) Some aspects of perfect elimination orderings in chordal graphs. Discret Appl Math 7:325–331
26. Spinrad J (–) Efficient implementation of lexicographic depth first search. Submitted for publication
27. Tarjan RE (1972) Depth-first search and linear graph algorithms. SIAM J Comput 1(2): 146–160
28. Tarry G (1895) Le problème des labyrinthes. Nouvelles Annales de Math 14: 187–189
29. Tedder M (2011) Applications of lexicographic breadth first search to modular decomposition, split decomposition, and circle graphs. PhD thesis, University of Toronto

Uniform Covering of Rings and Lines by Memoryless Mobile Sensors

Paola Flocchini
School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON, Canada

Keywords

Barrier covering; Scattering; Uniform deployment

Years and Authors of Summarized Original Work

2008; Flocchini, Prencipe, Santoro
2008; Cohen, Peleg

Problem Definition

The Model

A *mobile robotic sensor* (or simply *sensor*) is modeled as a computational unit with sensorial capabilities: it can perceive the spatial environment within a fixed distance $V > 0$, called *visibility range*, it has its own local working memory, and it is capable of performing local computations [6, 7].

Each sensor is a point with its own local coordinate system, which might not be consistent with the ones of the other sensors. The sensor can move in any direction, but it may be stopped before reaching its destination, e.g., because of limits to its motion energy; however, it is assumed that the distance traveled in a move by a sensor is not infinitesimally small (unless it brings the sensor to its destination).

The sensors have no means of direct communication to other sensors. Thus, any communication occurs in a totally implicit manner, by observing the other sensors' positions. Moreover, they are *autonomous* (i.e., without a central control) *identical* (i.e., they execute the same protocol), and *anonymous* (i.e., without identifiers that can be used during the computation).

The sensors can be *active* or *inactive*. When *active*, a sensor performs a *Look-Compute-Move* cycle of operations: it first observes the portion of the space within its visibility range obtaining a snapshot of the positions of the sensors in its range at that time (*Look*); using the snapshot as an input, the sensor then executes the algorithm to determine a destination point (*Compute*); finally, it moves toward the computed destination, if different from the current location (*Move*). After that, it becomes *inactive* and stays idle until the next activation. Sensors are *oblivious*: when a sensor becomes active, it does not remember any information from previous cycles.

Depending on the degree of synchronization among the cycles of different sensors, three submodels are traditionally identified: *synchronous*, *semi-synchronous*, and *asynchronous*. In the *synchronous* (FSYNC) and in the *semi-synchronous* (SSYNC) models, there is a global clock tick reaching all sensors simultaneously, and a sensor's cycle is an instantaneous event that starts at a clock tick and ends by the next. In FSYNC, at each clock tick all sensors become active, while in SSYNC some sensors might not be active in each cycle. In the *asynchronous* model (ASYNC), there is no global clock and the sensors do not have a common notion of time. Furthermore, the duration of each activity (or inactivity) is finite but unpredictable. As a result, sensors can be seen while moving, and computations can be made based on obsolete observations.

The Problem

The (distributed) *uniform covering* problem refers to sensors, randomly dispersed in a *bounded* region of space, that must scatter themselves throughout the region so to "cover" it satisfying some optimization criteria. Consider the case of a circular rim $\mathcal{R}$ (i.e., a ring), and let $S = \{s_0, \ldots, s_{n-1}\}$ be the sensors initially arbitrarily placed in different points on $\mathcal{R}$, with s_i preceding s_{i+1} clockwise (the index operations are modulo n). We emphasize that these names are used for presentation purposes only, and are not known to the sensors. If the sensors agree on the notion of clockwise, we say that they have a *common orientation*. Let $d = L_{\mathcal{R}}/n$ where $L_{\mathcal{R}}$ is the length of the ring. In the following, unless otherwise stated, the sensors are assumed to have visibility range $V \geq 2d$. Let $d_i(t)$ be the distance between sensors s_i and s_{i+1} at time t; when no ambiguity arises, we shall omit the time and simply indicate the distance as d_i. The sensors are said to have reached an *exact uniform covering* (*exact covering* for simplicity) at time t if $d_i(t) = d$ for all $0 \leq i \leq n-1$. Given $\epsilon > 0$, the sensors are said to have reached an *ϵ-approximate covering* at time t if $d - \epsilon \leq d_i(t) \leq d + \epsilon$ for all $0 \leq i \leq n-1$.

Key Results

The Ring

Exact Uniform Covering

There is a strong impossibility result that stresses the importance of having common orientation. If the sensors have only a local notion of left and right, but do not share a *common orientation* of the ring, the exact covering problem is unsolvable. This result holds even if the sensors had unbounded memory and visibility, and under a SSYNC scheduler.

Theorem 1 ([5]) *Let the sensors be on a ring $\mathcal{R}$. In absence of common orientation, there is no*

deterministic exact covering algorithm even if the sensors have unbounded persistent memory, their visibility range is unlimited, and the scheduling is SSYNC.

To see why this is the case, consider the following setting. Let n be even; partition the sensors in two sets, $S_1 = \{s_1, \ldots, s_{n/2}\}$ and $S_2 = S \backslash S_1$, and place the sensors of S_1 and S_2 on the vertexes of two regular $(n/2)$-gons on $\mathcal{R}$, rotated of an angle $\alpha < 360°/n$. Furthermore, all sensors have their local coordinate axes rotated so that they all have the same view of the world. In other words, the sensors in S_1 share the same orientation, while those in S_2 share the opposite orientation of $\mathcal{C}$. If activating only the sensors in S_1, an exact covering (resp. *no* exact covering) on $\mathcal{R}$ is reached at time step t_{i+1}, then the same is true also activating only the ones in S_2. Clearly, in such a case, activating both sets no exact covering would be reached at time step t_{i+1}, and the system would be an analogous configuration as the one of time step t_i, with different angles. Using this property, it is easy to design an adversary that will force any algorithm to never succeed in solving the problem; its behavior would be as follows: (i) If activating only the sensors in S_1 (resp. S_2) no exact covering on $\mathcal{R}$ is reached, then activate all sensors in S_1 (resp. S_2), while all sensors in S_2 (resp. S_1) are inactive; (ii) otherwise, activate all sensors. Go to (i).

On the other hand, assuming common orientation and knowledge of the final inter-distance d among sensors, a simple algorithm that solves the exact covering in ASYNC is for each sensor to move toward the point at distance d from its clockwise successor (if visible). We remind that $V \geq 2d$.

Protocol RINGCOVERINGEXACT (for sensor s_i)
Assumptions: Orientation, knowledge of d.

1. If s_{i+1} is not visible, move distance d clockwise.
2. else, if $d_i > d$ move toward point x at distance d from s_{i+1}.

Theorem 2 ([5]) *The exact covering of the ring problem is solvable in* ASYNC*, with common orientation and knowledge of the final inter-distance.*

Approximate Covering

Assuming common orientation but no knowledge of the final inter-distance among sensors, an ϵ-approximate covering is still possible for any $\epsilon > 0$, but no exact covering algorithm is known. Also this algorithm is very simple: the sensors asynchronously and independently *Look* in both directions, then they position themselves in the middle between the closest observed sensors (if any). Correctness is shown by proving that the minimum distance between any two neighboring sensors eventually grows, while the maximum distance eventually shrinks in such a way that there is a time when all sensors are within $d \pm \epsilon$ distance.

Theorem 3 ([5]) *The approximate covering of the ring problem is solvable in* ASYNC *with common orientation.*

Algorithm RINGCOVERINGAPPROX (for sensor s_i)
Assumptions: Orientation

- If no sensor is visible clockwise (resp. counterclockwise), let $d_i = V$ (resp. $d_{i-1} = V$).
- If $d_i \leq d_{i-1}$ do not move.
- If $d_i > d_{i-1}$ move distance $\frac{d_i + d_{i-1}}{2} - d_{i-1}$ clockwise.

Note that the covering problem has been also studied in discrete rings [4].

The Line

The case of a line segment is quite different from the one of the ring, and perhaps surprisingly, it is not easier. Let $S = \{s_0, \ldots, s_{n-1}\}$ be the sensors initially arbitrarily placed in different points on a line $\mathcal{L}$ with s_0 and s_{n-1} being two special immobile sensors delimiting the segment to be covered and with s_i preceding s_{i+1} $(0 < i < n-2)$. Let $d = L_{\mathcal{L}}/(n-1)$, where $L_{\mathcal{L}}$ denotes

the length of the segment. *Exact covering* and ϵ-*approximate covering* are defined analogously to the case of the ring.

Exact Uniform Covering

With common orientation and known final inter-distance, an algorithm has been recently shown for oriented sensors in ASYNC [3]. The algorithm works even if the visibility range is just enough to sense the final inter-distance ($V = d$). Let $\delta \leq \frac{d}{2}$ be a fixed positive (arbitrarily small) constant the sensors agree upon.

Protocol CORRIDORCOVERINGEXACT (for sensor s_i)
Assumptions: Orientation, knowledge of d, $V = d$

- If s_{i-1} is not visible, move distance $\frac{d}{2}$ to the left.
- else, let $a := d - d_{i-1}$
  If $d_i \geq d$ and $a > 0$, move distance $min(\frac{d}{2} - \delta, a)$ to the right.

Theorem 4 ([3]) *The exact covering of the line problem is solvable in* ASYNC *with common orientation and knowledge of the final inter-distance.*

With fixed visibility, a distributed algorithm has been proposed for FSYNC in a discrete setting, to solve the slightly different problem of barrier coverage [2].

Approximate Covering

Approximate covering has been studied in a slightly different visibility model where each sensor is able to perceive up to the next sensor on the line [1]. In other words, in each direction, a sensor sees the closest sensor (if it exists), regardless of its distance, but its visibility is blocked by it (*neighbor visibility*). For presentation purposes, a global linear coordinate system (not known to the sensors) is used here with $s_0(t) = 0$ and $s_{n-1}(t) = 1$. For the sensors to be spread uniformly, sensor s_i should then occupy position $\frac{i}{n-1}$. The following is a simple approximate covering algorithm.

Protocol CORRIDORSPREAD (for sensor s_i)
Assumptions: SSYNC, neighbour visibility

- If no sensor is visible in either direction, do nothing.
- Otherwise, move toward point $x = \frac{1}{2}(s_{i+1} + s_{i-1})$.

The idea of the convergence proof in FSYNC is sketched below. Let $\mu_i[t]$ be the *shift* of the s_i's location at time t from its final position. According to the protocol, the position of sensor s_i changes from $s_i(t)$ to $s_i(t+1) = \frac{1}{2}(s_{i-1}(t) + s_{i+1}(t))$ for $1 \leq i \leq n-2$, while sensors s_0 and s_{n-1} never move. Therefore, the shifts changes with time as $\mu_i[t+1] = \frac{1}{2}(\mu_{i+1}[t] + \mu_{i-1}[t])$. Considering the *progress* measure, $\psi[t] = \Sigma_{i=1}^{i=n} \mu^2[t]$, it can be shown that $\psi[t]$ is a decreasing function of t unless the sensors are already equally spread; more precisely, it is shown that every $O(n^2)$ cycle, $\psi[t]$ is at least halved thus reaching approximate covering. More complex but analogous reasoning is followed for SSYNC.

Theorem 5 ([1]) *The approximate covering of the line problem is solvable in* SSYNC *with* neighbor visibility.

With a simple modification of the algorithm, the result above can be extended to any fixed visibility $V > d$, provided that d is known, as described below [3].

Protocol CORRIDORSPREAD2 (for sensor s_i)
Assumptions: SSYNC, d known, $V > d$

- If only one sensor $s_j \in \{s_{i+1}, s_{i-1}\}$ is visible to s_i and $d' = dist(s_i, s_j) < d$: move distance $\frac{d-d'}{2} + \frac{V-d}{2}$ away from s_j
- If both s_{i+1}, s_{i-1} are visible and $d_1 = dist(s_{i-1}, s_i) < d_2 = dist(s_{i+1}, s_i)$ (resp. $d_1 = dist(s_{i+1}, s_i) < d_2 = dist(s_{i-1}, s_i)$): move $\frac{d_2-d_1}{2}$ toward s_{i+1} (resp. toward s_{i-1})

Applications

Uniform covering problems are important in many applications; covering of a circular rim occurs, for example, when the sensors have to surround a dangerous area and can only move along its outer perimeter. On the other hand, coverings of the line (often called *barrier coverings*) guarantee that any intruder attempting to cross the perimeter of a protected region (e.g., crossing an international border) is detected by one or more of the sensors. These problems are studied under a variety of assumption; the majority of the studies uses sensors provided with memory, explicit communication devices, global localization capabilities (e.g., GPS), and centralized approaches. The advantage of memoryless sensors are self-stabilization and tolerance to loss of sensors, the use of local coordinate systems has clear advantages over the full strength of a GPS; finally, decentralized solutions offer better fault tolerance.

Open Problems

It is known that the exact covering of the ring is impossible without orientation in SSYNC, but the impossibility does not extend to FSYNC where, however, no algorithm is known. Moreover, the only existing exact covering algorithm in ASYNC assumes orientation, which is needed, and knowledge of the inter-distance d, which is possibly not needed, so a tighter result might be possible. Finally, approximate covering is achieved in the ring in SSYNC assuming orientation, which is not shown to be necessary, furthermore, no solution exists for ASYNC.

In the case of the line, the only impossibility result for exact covering [3] holds for fully disoriented sensors (not even able to locally distinguish between their two directions) and with small visibility range $V = d$. As for approximate covering, the only known result in this model is for SSYNC, and it is not known whether an algorithm exists for the ASYNC model.

Recommended Reading

1. Cohen R, Peleg D (2008) Local spreading algorithms for autonomous robot systems. Theor Comput Sci 399:71–82
2. Eftekhari Hesari M, Kranakis E, Krizanc D, Morales Ponce O, Narayanan L, Opatrny J (2013) Distributed algorithms for barrier coverage using relocatable sensors. In: ACM symposium on principles of distributed computing (PODC), Montreal, Canada, pp 383–392
3. Eftekhari Hesari M, Flocchini P, Narayanan L, Opatrny J, Santoro N (2014) Distributed barrier coverage with relocatable sensors. In: 21th international colloquium on structural information and communication complexity (SIROCCO), Takayama, Japan
4. Elor Y, Bruckstein AM (2011) Uniform multi-agent deployment on a ring. Theor Comput Sci 412:783–795
5. Flocchini P, Prencipe G, Santoro N (2008) Self-deployment algorithms for mobile sensors on a ring. Theor Comput Sci 402(1):67–80
6. Flocchini P, Prencipe G, Santoro N (2011) Computing by mobile robotic sensors. In: Nikoletseas S, Rolim J (eds) Theoretical aspects of distributed computing in sensor networks, chap 21. Springer, Heidelberg ISBN:978-3-642-14849-1
7. Flocchini P, Prencipe G, Santoro N (2012) Distributed computing by oblivious mobile robots. Morgan & Claypool, San Rafael

Unique *k*-SAT and General *k*-SAT

Timon Hertli
Department of Computer Science, ETH Zürich, Zürich, Switzerland

Keywords

3-SAT; Boolean satisfiability; k-SAT; SAT

Years and Authors of Summarized Original Work

2011; Hertli

Problem Definition

A Boolean formula F is said to be in conjunctive normal form (CNF) if it is a conjunction of

disjunction of literals. If furthermore every disjunction (called clause) is over at most k literals, F is said to be in k-CNF. k-SAT, the decision problem whether a k-CNF formula admits a satisfying assignment, is one of the most prominent NP-complete problems. A special case of k-SAT is (promise) unique k-SAT, where the k-CNF is additionally promised to have either a unique or no satisfying assignment.

Suppose F has n variables. The trivial algorithm tries all 2^n satisfying assignments. For k-SAT and especially 3-SAT, there have been many successive improvements [3–9, 11]. The best of them are *randomized* in the sense that they always correctly report unsatisfiability but might fail to report satisfiability with probability $\frac{1}{3}$, say.

Problem 1 (k-SAT)

INPUT: A k-CNF formula F.
OUTPUT: "No" if F is not satisfiable. "Yes" with probability at least $\frac{2}{3}$ if F is satisfiable.

Problem 2 (Unique k-SAT)

INPUT: A k-CNF formula F with at most one satisfying assignment.
OUTPUT: "No" if F is not satisfiable. "Yes" with probability at least $\frac{2}{3}$ if F is satisfiable.

It is conjectured that unique k-SAT and k-SAT have the same exponential complexity; however, this could only be shown for $k \to \infty$ [1]. Especially for PPSZ [9], the fastest known (randomized) algorithm for unique k-SAT, the analysis results in a gap between k-SAT and unique k-SAT for $k = 3, 4$. Furthermore, the PPSZ algorithm has been derandomized for unique k-SAT [10] but not for general k-SAT.

Notation For a CNF formula F over a variable set V, denote by $\mathrm{sat}(F)$ the set of satisfying assignments of F on V. For x a variable and b a Boolean value, define $F^{[x \mapsto b]}$ the restriction of F by $x \mapsto b$, i.e., the formula obtained by replacing x by b in F.

Key Results

The bounds of the PPSZ algorithm for unique k-SAT hold for general k-SAT also if $k = 3, 4$ [2]. This makes PPSZ the fastest known k-SAT algorithm for all k. In the analysis of [2], the PPSZ algorithm is slightly modified.

Theorem 1 *There is a randomized algorithm for 3-SAT running in time* $O(2^{0.387n})$.

Theorem 2 *There is a randomized algorithm for 4-SAT running in time* $O(2^{0.555n})$.

Algorithm 1 PPSZ(k-CNF formula F)

$V \leftarrow$ variables of F
Choose β uniformly at random from all assignments on V
Choose π uniformly at random from all permutations of V
Let α be a partial assignment over V, initially the empty assignment
for all $x \in V$ in the order prescribed by π **do**
  while there is an $\log n$-implied assignment $y \mapsto a$ of F **do**
    $F \leftarrow F^{[y \mapsto a]}$
    $\alpha(y) \leftarrow a$
  end while
  if $\alpha(x)$ not fixed yet **then**
    $F \leftarrow F^{[x \mapsto \beta(x)]}$
    $\alpha(x) \leftarrow \beta(x)$
  end if
end for
return If α satisfies F, return 'satisfiable', otherwise return 'failure'.

If F is not satisfiable, then PPSZ will never find a satisfying assignment and thus is always correct. Hence, let F be a satisfiable k-CNF formula over n variables V. The PPSZ algorithm tries to find a satisfying assignment of F by iteratively setting variables as follows: Go through the variables one by one, in random order. If a variable x is not set at its step, then its value will be guessed uniformly at random. Between steps, we might infer the value of some variables in subexponential time: Setting x to a is called $\log n$-implied (by F) if there is a set of $\log n$ clauses G in F such that all satisfying assignments of G set x to a. $\log n$-implication can be checked in subexponential time by brute

force; the algorithm fixes all $\log n$-implications accordingly between steps.

If a variable is determined by $\log n$-implication, it is called *forced*; otherwise, it is called *guessed*. The key result of [9] for unique k-SAT is the following: If F is in k-CNF and has a unique satisfying assignment α, then given that $\beta = \alpha$ (i.e., all guesses are according to α), a variable is forced with a certain probability R_k. By Jensen's inequality one can then show that α is found with probability $2^{-(1-R_k)n}$. We have $R_3 = 2 - 2\ln 2 \approx 0.613$ and $R_4 \approx 0.445$. Repeating PPSZ inversely proportional to its success, probability will match the above theorems for unique 3-SAT and unique 4-SAT.

If there are multiple satisfying assignments, there is no bound on the probability that a variable is forced. For example, the empty CNF formula that always evaluates to true will never have a forced variable, as being forced depends on certain assignments not being satisfying. However, the following can be done: Given a satisfiable CNF formula F, call a variable x *frozen* if it has the same value in all satisfying assignments of F, and call x *non-frozen* otherwise. If x is frozen, the same bound on the probability that it is forced holds by the arguments of [9]. If x is non-frozen, then it can be set both ways and the resulting formula remains satisfiable. The remaining problem is that the probability for frozen variables depends on a fixed satisfying assignment and a uniform permutation; however, depending on the permutation, certain assignments will be more or less likely. This leads to a correlation issue that has to be solved by balancing the correlation and the benefit of non-frozen variables by careful bookkeeping.

Let V_f be the frozen variables of F and V_n be the non-frozen variables of F. The *likelihood* of an assignment α in F, $\mathrm{lkhd}(F, \alpha)$, is recursively defined as follows: If α does not satisfy F, then $\mathrm{lkhd}(F, \alpha) = 0$. If α is the unique satisfying assignment of F, then $\mathrm{lkhd}(\alpha) = 1$. Otherwise, let $\mathrm{lkhd}(\alpha) = \frac{1}{|V_f|+2|V_n|}\left(\sum_{x \in V} \mathrm{lkhd}(\alpha, F^{[x \mapsto \alpha(x)]})\right)$. The likelihood simulates how likely an assignment would be returned by PPSZ in an ideal setting.

With this, the cost F is defined as follows: For a non-frozen variable x, $\mathrm{cost}(F, x) = 1 - R_k$. For a frozen variable x, first defined for a satisfying assignment α, $\mathrm{cost}(F, x, \alpha)$ is the probability that x is guessed if executing PPSZ conditioned on $\beta = \alpha$. Then $\mathrm{cost}(F, x) = \sum_{\alpha \in \mathrm{sat}(F)} \mathrm{lkhd}(F, \alpha)\mathrm{cost}(F, x, \alpha)$. Observe that $\mathrm{cost}(F, x) \leq 1 - R_k$, as frozen variables are guessed with probability at most $1 - R_k$. In total we define $\mathrm{cost}(F) = \sum_{x \in V} \mathrm{cost}(F, x) \leq (1 - R_k)n$. The following theorem relates the cost to the probability that PPSZ finds an assignment:

Theorem 3

$$\Pr(\textit{PPSZ finds some satisfying assignment of } F) \geq 2^{-c(F)}.$$

This theorem immediately implies Theorems 1 and 2. The theorem is by induction on the number of variables of F. After a single PPSZ step, the cost decreases in expectation; the more the more frozen variables there are. On the other hand, the more non-frozen variables there are, the higher the probability is to retain a satisfiable formula. Balancing these factors and applying Jensen's inequality gives the theorem. It is noteworthy that the proof relies on the inequality $0.613 \approx R_3 \leq \frac{1}{2\ln 2} \approx 0.721$, meaning that if PPSZ would be improved beyond this bound, the unique case might indeed be better.

Open Problems

- Is the exponential complexity of k-SAT and unique k-SAT the same? Here this has been shown for the specific case of the PPSZ algorithm.
- Does PPSZ perform even better on formulas with exponentially many satisfying assignments?
- Can PPSZ be derandomized for general k-SAT?

Cross-References

- Backtracking Based k-SAT Algorithms
- Derandomization of k-SAT Algorithm
- Exact Algorithms for General CNF SAT
- Exact Algorithms for k SAT Based on Local Search
- Exponential Lower Bounds for k-SAT Algorithms

Recommended Reading

1. Calabro C, Impagliazzo R, Kabanets V, Paturi R (2008) The complexity of unique k-SAT: an isolation lemma for k-CNFs. J Comput Syst Sci 74(3):386–393
2. Hertli T (2011) 3-SAT faster and simpler—unique-SAT bounds for PPSZ hold in general. In: 2011 IEEE 52nd annual symposium on foundations of computer science—FOCS 2011, Palm Springs. IEEE Computer Society, Los Alamitos, pp 277–284
3. Iwama K, Tamaki S (2004) Improved upper bounds for 3-SAT. In: Proceedings of the fifteenth annual ACM-SIAM symposium on discrete algorithms, New Orleans. ACM, New York, pp 328–329 (electronic)
4. Kullmann O (1997) Worst-case analysis, 3-SAT decision and lower bounds: approaches for improved SAT algorithms. In: Satisfiability problem: theory and applications, Piscataway, 1996. DIMACS series in discrete mathematics and theoretical computer science, vol 35. American Mathematical Society, Providence, pp 261–313
5. Makino K, Tamaki S, Yamamoto M (2011) Derandomizing HSSW algorithm for 3-SAT. In: Computing and combinatorics, Dallas. Lecture notes in computer science, vol 6842. Springer, Heidelberg, pp 1–12
6. Monien B, Speckenmeyer E (1985) Solving satisfiability in less than 2^n steps. Discret Appl Math 10(3):287–295
7. Moser RA, Scheder D (2011) A full derandomization of Schoening's k-SAT algorithm. In: Proceedings of the 43rd annual ACM symposium on theory of computing, San Jose. ACM, pp 245–252
8. Paturi R, Pudlák P, Zane F (1999) Satisfiability coding lemma. Chic J Theor Comput Sci Article 11, 19 (electronic)
9. Paturi R, Pudlák P, Saks ME, Zane F (2005) An improved exponential-time algorithm for k-SAT. J ACM 52(3):337–364 (electronic)
10. Rolf D (2005) Derandomization of PPSZ for unique-k-SAT. In: Theory and applications of satisfiability testing, St Andrews. Lecture notes in computer science, vol 3569. Springer, Berlin, pp 216–225
11. Schöning U (1999) A probabilistic algorithm for k-SAT and constraint satisfaction problems. In: Proceedings of the 40th annual symposium on foundations of computer science, New York City. IEEE Computer Society, Los Alamitos, pp 410–414

Universal Sequencing on an Unreliable Machine

Julián Mestre[1,2] and Nicole Megow[3]
[1]Department of Computer Science, University of Maryland, College Park, MD, USA
[2]School of Information Technologies, The University of Sydney, Sydney, NSW, Australia
[3]Institut für Mathematik, Technische Universität Berlin, Berlin, Germany

Keywords

Availability periods; Machine speed; Min-sum objective; Scheduling; Sequencing; Universal solution; Unreliable machine; Worst-case guarantee

Years and Authors of Summarized Original Work

2012; Epstein, Levin, Marchetti-Spaccamela, Megow, Mestre, Skutella, Stougie
2013; Megow, Mestre

Problem Definition

Given a set of jobs $J = \{1, 2, \ldots, n\}$ with processing times $p_j \in \mathbb{R}_+$ and weights $w_j \in \mathbb{R}_+$, the task is to find a schedule for all jobs on a single machine that minimizes $\sum w_j C_j$, where C_j is the completion time of job j.

Under the standard scheduling assumption of an ideal machine that runs at constant speed, an optimal schedule is obtained by sequencing the jobs in nonincreasing order of the ratio w_j/p_j; this is known as Smith's Rule [12]. Unfortunately, as we shall see shortly, this sequence may per-

form arbitrarily bad when the machine is not ideal.

This note is concerned with the setting in which the machine may change its processing speed over time or it may fully break down and is unavailable until it is fixed. Given a sequence, the jobs are processed in this order no matter how the machine behaves. In case of a machine breakdown, the job that is currently running is preempted and resumes processing when the machine becomes available again at a later time. The aim is to compute a *universal sequence* that, for any given machine behavior, is a good approximation of an optimal schedule for that particular machine behavior.

Definition 1 A sequence π is a *universal c-approximation* if for any machine behavior the total weighted completion times of π is at most a factor c larger than the objective value of an optimal solution for this machine behavior.

To illustrate this definition consider the following toy instance with two jobs: $p_1 = w_1 = 2$ and $p_2 = w_2 = N \gg 2$. There are only two possible sequences: $(1, 2)$ and $(2, 1)$. Both sequences are optimal on an ideal machine and are consistent with Smith's Rule. Now suppose our machine breaks down at $t = N + 1$ and stays offline for $T = N^2$ units of time. The cost of $(1, 2)$ on this faulty machine is $4 + N(N + 2 + T) = \Theta(N^3)$, while the cost of $(2, 1)$ is $N^2 + 2(N + 2 + T) = \Theta(N^2)$. This example shows that Smith's Rule can produce a sequence that is not a universal $O(1)$-approximation. In fact, it is not clear that such a universal sequence should always exist.

Key Results

Epstein et al. [3] initiated the study of universal sequencing. They showed that universal $O(1)$-approximate sequences do indeed exist and established tight lower bounds on the universal approximation ratio that can be achieved. Their study was subsequently furthered by Megow and Mestre [10] who showed that the best universal schedule can be approximated in polynomial time up to any desired level of accuracy.

Bounding the Performance of a Universal Sequence

The key observation needed to bound the performance of a universal sequence is that approximating the min-sum objective value on a machine with unknown processing behavior is equivalent to approximating the total weight of uncompleted jobs at any point in time on an ideal machine. To that end, let $W^\pi(t)$ denote, for any $t \geq 0$, the total weight of outstanding jobs at time t in the schedule obtained for job sequence π on an ideal machine. Define $W^*(t) := \min_\pi W^\pi(t)$ for all $t \geq 0$.

Lemma 1 *Let π be a sequence of jobs. Then, the objective value of the corresponding schedule is at most c times the value of an optimum schedule for any machine behavior, if and only if*

$$W^\pi(t) \leq c \cdot W^*(t) \qquad \text{for all } t \geq 0.$$

A Universal Sequencing Algorithm

The universal sequencing algorithm computes the job sequence iteratively backwards. In each iteration it solves the subproblem of finding a set of jobs that has maximum total processing time and total weight within a given bound. This bound is doubled in each iteration.

This approach is related to, but not equivalent to, an algorithm of Hall et al. [6] for online scheduling on ideal machines – the doubling there happens in the time horizon. Indeed, *doubling* strategies have been applied successfully in the design of approximation and online algorithms for various problems; see, e.g., the survey by Chrobak and Kenyon-Mathieu [1].

Doubling Algorithm:

1. For $i \in \{0, 1, \ldots, \lceil \log w(J) \rceil\}$, find a subset J_i^* of jobs of maximum total processing time $p(J_i^*)$, such that the total weight satisfies $w(J_i^*) \leq 2^i$.
2. Construct a permutation π as follows. Start with an empty sequence of jobs. For $i = \lceil \log w(J) \rceil$ down to 0, append the jobs in $J_i^* \setminus \bigcup_{k=0}^{i-1} J_k^*$ in any order at the end of the sequence.

Finding the subsets of jobs J_i^* is a KNAPSACK problem and, thus, NP-hard [8]. Using straightforward dynamic programming, the algorithm runs in pseudo-polynomial time and achieves a performance guarantee of 4 as shown below. However, FPTASes for the knapsack problem can be adopted such that the *Doubling Algorithm* runs in polynomial time loosing an arbitrarily small constant in the performance guarantee.

Theorem 1 *For every scheduling instance, the* Doubling Algorithm *produces a universal* 4*-approximation for all machine behaviors.*

Proof By Lemma 1 it is sufficient to show that $W^\pi(t) \leq 4W^*(t)$ for all $t \geq 0$. Let $t \geq 0$ and let i be minimal such that $p(J_i^*) \geq p(J) - t$. By construction of π, only jobs j in $\bigcup_{k=0}^{i} J_k^*$ can have a completion time $C_j^\pi > t$. Thus,

$$W^\pi(t) \leq \sum_{k=0}^{i} w(J_k^*) \leq \sum_{k=0}^{i} 2^k = 2^{i+1} - 1. \tag{1}$$

In case $i = 0$, the claim is trivially true since $w_j \geq 1$ for any $j \in J$, and thus, $W^*(t) = W^\pi(t)$. Suppose $i \geq 1$; then by our choice of i, it holds that $p(J_{i-1}^*) < p(J) - t$. Therefore, in any sequence π', the total weight of jobs completing after time t is larger than 2^{i-1}, because otherwise we get a contradiction to the maximality of $p(J_{i-1}^*)$. That is, $W^*(t) > 2^{i-1}$. Together with (1) this concludes the proof. □

This result is best possible for universal sequencing on a single machine.

Theorem 2 *For any $c < 4$, there exists an instance for which there is no universal c-approximation.*

This can be shown through a connection to the *online bidding problem* and the corresponding lower bounds shown by Chrobak et al. [2].

Randomized Universal Schedules

It is possible to obtain a better approximation ratio if we select the sequence at random and slightly relax the universality requirement.

Definition 2 A probability distribution over sequences is a *randomized universal c-approximation* if for any machine behavior the *expected* total weighted completion times of a sequence chosen according to the distribution is at most a factor c larger than the objective value of an optimal solution for this machine behavior.

By randomizing the "doubling parameter" in the Doubling Algorithm, the algorithm can achieve an approximation ratio of $e \approx 2.718$, which is best possible for randomized strategies.

Theorem 3 *For every scheduling instance, a randomized variant of the* Doubling Algorithm *produces a randomized universal e-approximation for all machine behaviors. Furthermore, for any $c < e$, there exists an instance for which there is no randomized universal c-approximation.*

Generalizations

Global Cost Functions

The universality of the sequence constructed by the *Doubling Algorithm* can be driven even further. Consider the generalized min-sum objective $\min \sum w_j f(C_j)$ for any nondecreasing, nonnegative, differentiable cost function f.

Theorem 4 *The Doubling Algorithm computes a universal* 4*-approximation (randomized e-approximation) for all machine behaviors and all considered cost functions f simultaneously.*

Precedence Constraints

A natural generalization of the universal sequencing problem requires that jobs must be sequenced in compliance with given *precedence constraints*. To a certain extent the *Doubling Algorithm* can be adopted to this more general problem setting. Essentially, the knapsack-related subroutine must respect the precedence constraints, *and* it must ensure that prepending the subsets found in different iterations, starting in the end, does not violate the precedence order.

This corresponds to solving a so-called *partially ordered knapsack* (POK) problem on the reverse of the given partial order.

Theorem 5 *The Doubling Algorithm computes a universal* 4*-approximation (randomized e-approximation) for the universal scheduling problem respecting given precedence constraints if the POK problem for the given partial order can be solved in polynomial time.*

In general, POK is strongly NP-hard [7] and hard to approximate [5]. However, FPTASes exist for special partial orders, including directed out-trees, two-dimensional orders, and the complement of chordal bipartite orders [7, 9].

Release Dates

If jobs have *release dates*, we cannot hope for a universal sequence with bounded approximation ratio unless the scheduler is allowed to preempt jobs. We can think of a universal sequence as a priority order of the jobs guiding a preemptive list scheduling procedure: At any point in time, we work on the job of highest priority that has not been finished yet and that has already been released. Unfortunately, even with this flexibility, the problem is significantly harder.

Theorem 6 *There exists an instance with n jobs with release dates and unit weights, where the performance guarantee of any universal schedule is* $\Omega(\log n / \log \log n)$.

The proof relies on the classical theorem of Erdős and Szekeres [4] on the existence of long increasing/decreasing subsequences of a given sequence of distinct real numbers.

Despite this negative result, there is a non-trivial algorithm that produces a universal 5-approximate sequencing for the class of instances with release dates in which the processing time of each job is proportional to its weight.

Instance-Sensitive Performance Guarantee

Theorem 1 says that the *Doubling Algorithm* produces for every instance a universal 4-approximation. Theorem 2 proves that this is best possible since there are particular instances that do not admit a sequence with a smaller approximation ratio. Many instances, however, admit better-than-4-approximate universal sequences, yet the Doubling Algorithm is only guaranteed to find a 4-approximation. This motivates the problem of finding the best possible universal sequence on an *instance-by-instance* basis.

Theorem 7 *For any fixed* $\epsilon > 0$ *and* $c > 1$*, there is a polynomial time algorithm that given an instance either finds a* $(c + \epsilon)$*-approximate universal sequence or determines that there is no universal* c*-approximation for this particular instance.*

Applications

The unreliable machine scheduling problem addresses the demand for high-quality scheduling solutions in the dynamic real-world environments of manufacturing processes or in operating systems. The machine could be, for example, a computer server that slows down due to unpredictable third-party usage or an aging production unit prone to unexpected breakdowns. Another setting where the model is applicable is where a higher authority may give priority to another batch of jobs, thus delaying the execution of our jobs. In general, universally good performance regardless of the actual machine behavior is desirable in highly automated systems in which changing the schedule at an arbitrary point in time is too costly or technically infeasible.

Open Problems

The worst-case performance of universal sequences is quite well understood. While the analysis in the model without release dates is tight, it remains open in the setting with release dates if it is possible to obtain a universal $o(n)$-approximation for general instances. The best known lower bound is $\Omega(\log n / \log \log n)$.

While the worst-case analysis assumes arbitrary machine behaviors, it would be interesting to develop and analyze more realistic speed functions. For example, it is reasonable to assume that when a machine breaks down, then it will be repaired or replaced within a certain (possibly fixed) amount of time; or in a stochastic model, the availability periods between breakdowns may be assumed to be exponentially distributed. What improvements in the approximation guarantee do such restrictions allow?

A different approach in aiming for more practice-relevant guarantees is to relax the strict universality requirement. In many situations, changing the scheduling sequence is possible to a certain extent at some extra cost. A very interesting problem is to quantify the amount of adaptivity an algorithm needs to achieve a certain performance guarantee. Ideally, there is a parameter describing the adaptivity that allows to scale between the nonadaptive 4-approximation (Theorem 1) and a fully adaptive $(1 + \epsilon)$-approximation, given by a PTAS that constructs an individual scheduling solution for a specific machine behavior [11].

Cross-References

- List Scheduling
- Minimum Weighted Completion Time
- Robust Scheduling Algorithms

Recommended Reading

1. Chrobak M, Kenyon-Mathieu C (2006) Sigact news online algorithms column 10: competitiveness via doubling. SIGACT News 37(4):115–126
2. Chrobak M, Kenyon C, Noga J, Young N (2008) Incremental medians via online bidding. Algorithmica 50(4):455–478
3. Epstein L, Levin A, Marchetti-Spaccamela A, Megow N, Mestre J, Skutella M, Stougie L (2012) Universal sequencing on a single unreliable machine. SIAM J Comput 41(3): 565–586
4. Erdős P, Szekeres G (1935) A combinatorial problem in geometry. Compos Math 2: 463–470
5. Hajiaghayi M, Jain K, Lau L, Mandoiu I, Russell A, Vazirani V (2006) Minimum multicolored subgraph problem in multiplex PCR primer set selection and population haplotyping. In: Proceedings of second IWBRA, Atlanta, GA, pp 758–766
6. Hall L, Schulz A, Shmoys D, Wein J (1997) Scheduling to minimize average completion time: off-line and on-line approximation algorithms. Math Oper Res 22:513–544
7. Johnson D, Niemi K (1983) On knapsacks, partitions, and a new dynamic programming technique for trees. Math Oper Res 8(1):1–14
8. Karp R (1972) Reducibility among combinatorial problems. In: Complexity of computer computations (Proceedings of symposium on IBM Thomas J. Watson Research Center, Yorktown Heights), Plenum, pp 85–103
9. Kolliopoulos S, Steiner G (2007) Partially ordered knapsack and applications to scheduling. Discret Appl Math 155(8):889–897
10. Megow N, Mestre J (2013) Instance-sensitive robustness guarantees for sequencing with unknown packing and covering constraints. In: Proceedings of ITCS, Berkeley, CA, pp 495–504
11. Megow N, Verschae J (2013) Dual techniques for scheduling on a machine with varying speed. In: Proceedings of ICALP, Riga, pp 745–756
12. Smith WE (1956) Various optimizers for single-stage production. Naval Res Logist Q 3:59–66

Upward Graph Drawing

Walter Didimo
Department of Engineering, University of Perugia, Perugia, Italy

Keywords

Directed graphs; Flow networks; Graph planarity; Hierarchical drawings

Years and Authors of Summarized Original Work

1994; Bertolazzi, Di Battista, Liotta, Mannino

Problem Definition

Upward graph drawing is concerned with computing two-dimensional layouts of directed graphs where all edges flow in the upward direction. Namely, given a directed graph $G(V, E)$ (also called a *digraph* for short), an *upward drawing* of G is a drawing such that: (i) each vertex $v \in V$ is mapped to a distinct point p_v of the plane and (ii) each edge $(u, v) \in E$ is drawn as a simple curve from p_u and p_v, monotonically increasing in the upward direction.

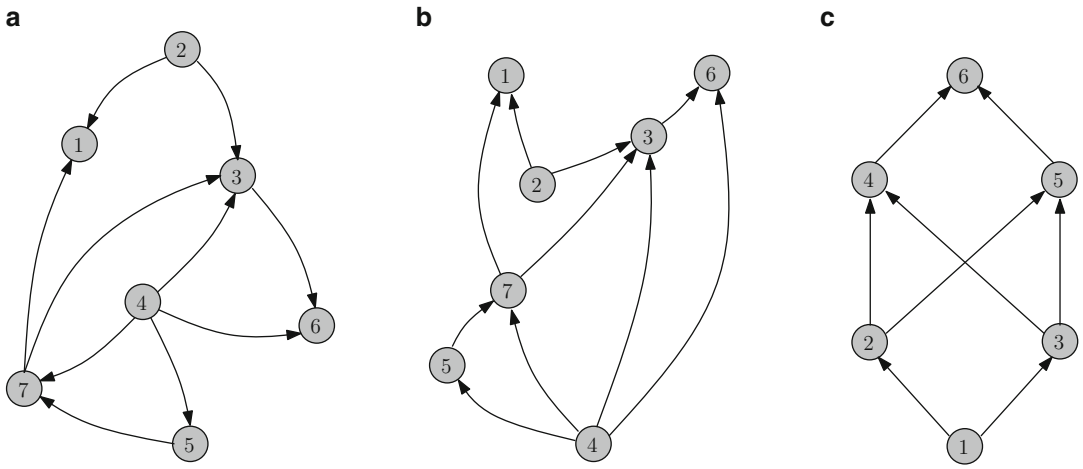
Clearly, G admits an upward drawing only if it does not contain directed cycles; if we allow edge crossings, acyclicity is also a sufficient condition for the existence of an upward drawing. Instead, if G is planar and we require that also the upward drawing of G is crossing-free, acyclicity is only a necessary condition, and the upward drawability of G becomes a much more intriguing problem. An upward drawing with no edge crossing is called an *upward planar drawing*; deciding whether a planar digraph G admits such a drawing is recognized as the *upward planarity testing* problem. This problem can be studied in two different settings:

- **Variable embedding setting.** The existence of an upward planar drawing of G is checked over all possible planar embeddings of G.
- **Fixed embedding setting.** The existence of an upward planar drawing of G is checked for a given planar embedding of G, i.e., the drawing must preserve the given embedding.

Both these settings have been widely studied in the literature. In the next section we briefly survey few seminal results on the upward planarity testing problem, and then we concentrate on the first and most popular polynomial-time algorithm for the fixed embedding setting. Figure 1 shows a planar digraph G with a given planar embedding, an embedding-preserving upward planar drawing of G, and a planar digraph G that does not admit upward planar drawings.

Key Results

Let $G(V, E)$ be a planar digraph. We will assume that G is connected (indeed, a digraph admits an upward planar drawing if and only if each of its connected components admits an upward



Upward Graph Drawing, Fig. 1 (**a**) A planar digraph G with a given planar embedding. (**b**) An upward planar drawing of G that preserves the embedding of G. (**c**) A planar digraph G that has no upward planar drawing

planar drawing). A *source* (resp. a *sink*) of G is a vertex with only outgoing (resp. incoming) edges. An *internal* vertex of G is a vertex with both outgoing and incoming edges. We denote by S, T, and I the set of sources, sinks, and internal vertices of G, respectively. Digraph G is a *planar st-digraph* if it has only one source s and one sink t and a planar embedding where s and t belong to the same face. Di Battista and Tamassia proved different equivalent characterizations of upward planar drawable digraphs, as stated in the following result [7]:

Theorem 1 ([7]) *Let G be a planar digraph. The following properties are equivalent:*

(a) G admits an upward planar drawing;
(b) G admits an upward planar drawing with straight-line edges;
(c) G is the spanning subgraph of a planar st-digraph.

Using Theorem 1, Garg and Tamassia focused on straight-line drawings and showed that the upward planarity testing problem in the variable embedding setting is NP hard [9]. As a consequence of this hardness result, polynomial-time algorithms in the variable embedding setting have been devised for restricted classes of planar digraphs, like single-source digraphs [6] and series-parallel digraphs [8], while exponential-time algorithms have been proposed for more general planar digraphs (see, e.g., [1, 8]).

Conversely, Bertolazzi et al. showed that the upward planarity testing problem can be solved in polynomial time in the fixed embedding setting [2]. In the following we describe this breakthrough result, which inspired several subsequent papers on the subject.

Polynomial-Time Upward Planarity Testing

Let $G(V, E)$ be an embedded planar digraph, and still denote by S, T, and I the number of sources, sinks, and internal vertices of G, respectively. The result in [2] is based on an elegant combinatorial characterization of the planar embedded digraphs that are upward planar drawable. We first recall few basic definitions.

Digraph G is *bimodal* if for every vertex $v \in I$, the outgoing edges of v are consecutive in the cyclic clockwise order around v (which implies that also the incoming edges of v are consecutive in the cyclic clockwise order around v). It is immediate to see that if a digraph G admits an embedding-preserving upward planar drawing, G is necessarily bimodal.

Let f be a face of G, and let $a = (e_1, v, e_2)$ be a triplet such that $v \in V$ is a vertex of the boundary of f and e_1, e_2 are two edges incident to v that are consecutive on the boundary of f (e_1 and e_2 may coincide if G is not biconnected). Triplet a is called an *angle at v in face f*, or simply an *angle of f*, or an *angle at v*. If both e_1 and e_2 are outgoing edges of v, we call a a *source-switch* angle of f; if both e_1 and e_2 are incoming edges of v, we call a a *sink-switch* angle of f. Denote by $S(f)$ and $T(f)$ the number of source-switch angles and the number of sink-switch angles of f, respectively. It can be easily observed that $S(f) = T(f)$. The *capacity* of f is defined as $cap(f) = \frac{S(f)+T(f)}{2} - 1$ if f is an internal face of G and as $cap(f) = \frac{S(f)+T(f)}{2} + 1$ if f is the external face of G. The number of sources and sinks in the digraph is nicely related to the face capacities, as stated by the following theorem.

Theorem 2 ([2]) *If G is a bimodal embedded planar digraph and F is the set of faces of G, then $\sum_{f \in F} cap(f) = |S| + |T|$.*

Now, given any upward planar drawing Γ, denote by $L(v)$ the number of geometric angles larger than π at vertex v in Γ and by $L(f)$ the number of geometric angles larger than π in face f in Γ. The following result establishes which kinds of angles in Γ can occur around the vertices and inside the faces of the digraph:

Theorem 3 ([2]) *Let G be an embedded planar digraph and let Γ be an embedding-preserving upward planar drawing of G. We have that:*

(i) $L(v) = 0$ for each $v \in I$ and $L(v) = 1$ for each $v \in S \cup T$;
(ii) $L(f) = cap(f)$, for each $f \in F$.

Motivated by Theorem 3, for any given embedded planar digraph G, one can look for an assignment of the angles of G to the faces of G, with these properties:

(a) For each source or sink v, exactly one angle at v is assigned to a face incident to v.
(b) For each face f, the number of angles assigned to f equals $cap(f)$.

Such an assignment is called an *upward-consistent assignment* of G. The following result translates the upward planarity testing problem into the problem of deciding whether G admits an upward-consistent assignment.

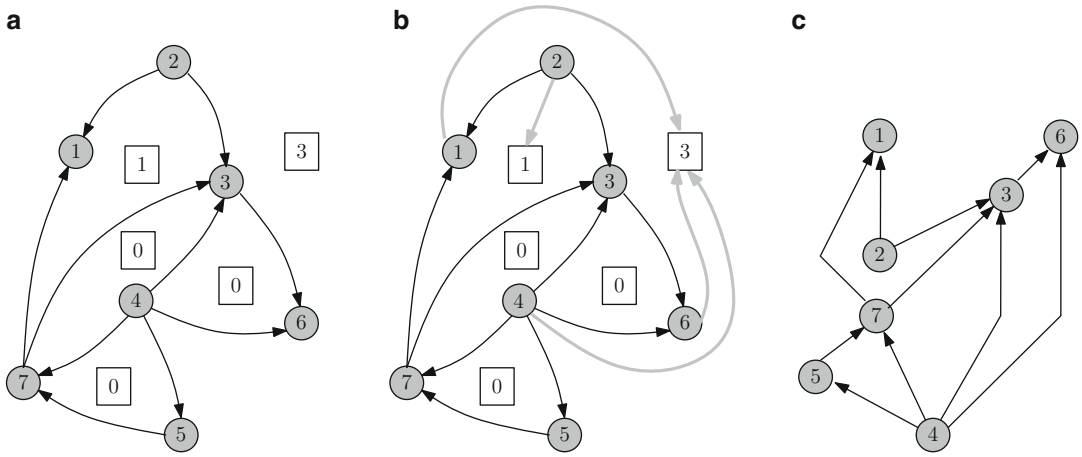
Theorem 4 ([2]) *Let G be an acyclic bimodal embedded planar digraph. G admits an embedding-preserving upward planar drawing if and only if G admits an upward-consistent assignment.*

In [2] it is proved that an upward-consistent assignment can be used to construct in linear time an upward planar drawing where each angle assigned to a face corresponds to a geometric angle larger than π. This is done by exploiting Theorem 1; namely, G is first augmented to an st-planar digraph G', then an upward drawing of G' is computed, and finally the dummy edges are removed from the drawing of G', thus obtaining an upward planar drawing of G.

Deciding whether G admits an upward-consistent assignment, and in case finding one, can be done using a network flow model. Namely, construct a bipartite flow network $N(G)$ having a node $n(v)$ for each source or sink v of G, called a *vertex-node*, and a node $n(f)$ for each face f of G, called a *face node*. Each vertex-node $n(v)$ supplies flow 1, while each face-node $n(f)$ demands a flow equal to $cap(f)$. Also, $N(G)$ has a directed arc $(n(v), (f))$ if v is a source or a sink that belongs to the boundary of f in G. A unit of flow on an arc $(n(v), (f))$ indicates that an angle at v in f must be assigned to f. Each feasible flow in $N(G)$ defines an upward-consistent assignment of G. Using standard flow algorithms, testing whether $N(G)$ has a feasible flow, and in case computing one, can be done in $O(n + r^2)$, where n is the number of vertices of G and $r = |S| + |T|$. Figure 2 illustrates the algorithmic approach described above for the upward planarity testing problem.

The next theorem summarizes the main result of [2].

Theorem 5 ([2]) *Let G be an acyclic bimodal embedded planar digraph with n vertices, and let r be the total number of sources and sinks of G.*



Upward Graph Drawing, Fig. 2 (**a**) An embedded planar digraph G; each face is represented by a *small box* reporting its capacity. (**b**) An upward consistent assignment of G; a *light gray arrow* indicates the assignment of an angle to a face. (**c**) An embedding-preserving upward planar drawing constructed from the upward-consistent assignment; the angles of G assigned to a face correspond to geometric angles larger than π in the drawing

There exists an $O(n + r^2)$-time algorithm that tests whether G admits an embedding-preserving upward planar drawing of G and that computes such a drawing if the test is positive.

Applications

Upward drawings can be effectively used to represent PERT networks, ISA hierarchies in knowledge-representation diagrams, and subroutine call charts. A generalized model, called *quasi-upward drawing*, strongly enlarges the range of application domains of upward graph drawing, making it possible to also represent cyclic digraphs [1] by allowing an edge to break its upward monotonicity in a finite number of points. Petri nets are examples of diagrams that can be represented as quasi-upward drawings; Petri nets are widely used to describe distributed systems.

Efficient C++ graph drawing libraries, like GDToolkit [5] and OGDF [3], implement advanced upward graph drawing algorithms.

Experimental Results

Extensive experimental studies on upward planarity testing are described in [1, 4]. Other references on experimental work about upward graph drawing algorithms can be found in [3, 5].

Cross-References

- Bend Minimization for Orthogonal Drawings of Plane Graphs
- Planarity Testing
- Sugiyama Algorithm

Recommended Reading

1. Bertolazzi P, Di Battista G, Didimo W (2002) Quasi-upward planarity. Algorithmica 32(3):474–506
2. Bertolazzi P, Di Battista G, Liotta G, Mannino C (1994) Upward drawings of triconnected digraphs. Algorithmica 12(6):476–497
3. Chimani M, Gutwenger C, Jünger M, Klau GW, Klein K, Mutzel P (2013) The open graph drawing framework (OGDF). In: Tamassia R (ed) Handbook of graph drawing and visualization. CRC Press - Taylor & Francis Group Boca Raton, FL, USA
4. Chimani M, Zeranski R (2013) Upward planarity testing: a computational study. In: Proceedings of Graph Drawing (GD'13), Bordeaux. Volume 8242 of LNCS. Springer, pp 13–24
5. Di Battista G, Didimo W (2013) GDToolkit. In: Tamassia R (ed) Handbook of graph drawing and visualization. CRC Press - Taylor & Francis Group Boca Raton, FL, USA
6. Di Battista G, Eades P, Tamassia R, Tollis IG (1998) Graph drawing: algorithms for the visualization of graphs. Prentice Hall, Upper Saddle River, New Jersey, USA
7. Di Battista G, Tamassia R (1988) Algorithms for plane representations of acyclic digraphs. Theor Comput Sci 61:175–198
8. Didimo W, Giordano F, Liotta G (2009) Upward spirality and upward planarity testing. SIAM J Discret Math 23(4):1842–1899
9. Garg A, Tamassia R (1992) On the computational complexity of upward and rectilinear planarity testing. SIAM J Comput 31(2):601–625

Utilitarian Mechanism Design for Single-Minded Agents

Piotr Krysta[1] and Berthold Vöcking[2]
[1]Department of Computer Science, University of Liverpool, Liverpool, UK
[2]Department of Computer Science, RWTH Aachen University, Aachen, Germany

Keywords

Forward (combinatorial; multi-unit) auction

Years and Authors of Summarized Original Work

2005; Briest, Krysta, Vöcking

Problem Definition

This problem deals with the design of efficiently computable incentive compatible, or truthful, mechanisms for combinatorial optimization

problems with selfish one-parameter agents and a single seller. The focus is on approximation algorithms for NP-hard mechanism design problems. These algorithms need to satisfy certain monotonicity properties to ensure truthfulness.

A *one parameter agent* is an agent who as her private data has some *resource* as well as a *valuation*, i.e., the maximum amount of money she is willing to pay for this resource. Sometimes, however, the resource is assumed to be known to the mechanism. The scenario where a single seller offers these resources to the agents is primarily considered. Typically, the seller aims at maximizing the social welfare or her revenue. The work by Briest, Krysta and Vöcking [6] will mostly be considered, but also other existing models and results will be surveyed.

Utilitarian Mechanism Design

A famous example of mechanism design problems is given by combinatorial auctions (CAs), in which a single seller, auctioneer, wants to sell a collection of goods to potential buyers. A wider class of problems is encompassed by a *utilitarian mechanism design (maximization) problem* Π defined by a finite set of *objects* $\mathcal{A}$, a set of feasible outputs $O_\Pi \subseteq \mathcal{A}^n$ and a set of n agents. Each agent declares a set of objects $S_i \subseteq \mathcal{A}$ and a valuation function $v_i : \mathcal{P}(\mathcal{A}) \times \mathcal{A}^n \to \mathbb{R}$ by which she values all possible outputs. Given a vector $S = (S_1, \ldots, S_n)$ of declarations one is interested in output $o^* \in O_\Pi$ maximizing the *social welfare*, i.e., $o^* \in \operatorname{argmax}_{o \in O_\Pi} \sum_{i=1}^n v_i(S_i, o)$. In CAs, an object a corresponds to a subset of goods. Each agent declares all the subsets she is interested in and the prices she would be willing to pay. An output specifies the sets to be allocated to the agents.

Here, a limited type of agents called *single-minded* is considered, introduced by Lehmann et al. [10]. Let $R_\preceq \subseteq \mathcal{A}^2$ be a reflexive and transitive relation on $\mathcal{A}$, such that there exists a special object $\varnothing \in \mathcal{A}$ with $\varnothing \preceq a$ for any $a \in \mathcal{A}$ to model the situation in which some agent does not contribute to the solution at all. For $a, b \in \mathcal{A}(a, b) \in R_\preceq$ will be denoted by $a \preceq b$. The single-minded agent i declares a *single* object a_i and is fully defined by her *type* (a_i, v_i), with $a_i \in \mathcal{A}$ and $v_i > 0$. The valuation function introduced earlier reduces to

$$v_i(a_i, o) = \begin{cases} v_i, & \text{if } a_i \preceq o_i \\ 0, & \text{else.} \end{cases}$$

Agent i is called *known* if object a_i is known to the mechanism [11]. Here, mostly unknown agents will be considered. Intuitively, each a_i corresponds to an object agent i offers to contribute to the solution, v_i describes her valuation of any output o that indeed selects a_i. In CAs, relation $R_\preceq$ is set inclusion: an agent interested in set S will is also satisfied by S' with $S \subseteq S'$. For ease of notation let $(a, v) = ((a_1, v_1), \ldots, (a_n, v_n))$, $(a_{-i}, v_{-i}) = ((a_1, v_1), \ldots, (a_{i-1}, v_{i-1}), (a_{i+1}, v_{i+1}), \ldots, (a_n, v_n))$ and $((a_i, v_i), (a_{-i}, v_{-i})) = (a, v)$.

Mechanism

A *mechanism* $M = (A, p)$ consists of an algorithm A computing a solution $A(a, v) \in O_\Pi$ and an n-tuple $p(a, v) = (p_1(a, v), \ldots, p_n(a, v)) \in \mathbb{R}_+^n$ of payments collected from the agents. If $a_i \preceq A(a, v)_i$, agent i is *selected*, and let $S(A(a, v)) = \{i \,|\, a_i \preceq A(a, v)_i\}$ be the set of selected agents. Agent i's type is her private knowledge. Thus, the types declared by agents may not match their true types. To reflect this, let (a_i^*, v_i^*) refer to agent i's true type and (a_i, v_i) be the declared type. Given an output $o \in O_\Pi$, the *utility* of agent i is $u_i(a, v) = v_i(a_i^*, o) - p_i(a, v)$. Each agent's goal is to maximize her utility. To achieve this, she will try to manipulate the mechanism by declaring a false type if this could result in higher utility. A mechanism is called *truthful*, or *incentive compatible*, if no agent i can gain by lying about her type, i.e., given declarations (a_{-i}, v_{-i}), $u_i((a_i^*, v_i^*), (a_{-i}, v_{-i})) \geq u_i((a_i, v_i), (a_{-i}, v_{-i}))$ for any $(a_i, v_i) \neq (a_i^*, v_i^*)$.

Monotonicity

A sufficient condition for truthfulness of approximate mechanisms for single-minded CAs was

first given by Lehmann et al. [10]. Their results can be adopted for the considered scenario. An algorithm A is *monotone* with respect to $R_{\preceq}$ if

$$i \in S(A((a_i, v_i), (a_{-i}, v_{-i}))) \Rightarrow i \in S(A((a_i', v_i'), (a_{-i}, v_{-i})))$$

for any $a_i' \preceq a_i$ and $v_i' \geq v_i$. Intuitively, one requires that a winning declaration (a_i, v_i) remains winning if an object a_i', smaller according to $R_{\preceq}$, and a higher valuation v_i' are declared. If declarations (a_{-i}, v_{-i}) are fixed and object a_i declared by i, algorithm A defines a *critical value* θ_i^A, i.e., the minimum valuation v_i that makes (a_i, v_i) winning, i.e., $i \in S(A((a_i, v_i), (a_{-i}, v_{-i})))$ for any $v_i > \theta_i^A$ and $i \notin S(A((a_i, v_i), (a_{-i}, v_{-i})))$ for any $v_i < \theta_i^A$. The *critical value payment scheme* p^A associated with A is defined by $p_i^A(a, v) = \theta_i^A$, if $i \in S(A(a, v))$, and $p_i^A(a, v) = 0$, otherwise. The critical value for any fixed agent i can be computed, e.g., by performing binary search on interval $[0, v_i]$ and repeatedly running algorithm A to check if i is selected. Also, mechanism $M_A = (A, p^A)$ is *normalized*, i.e., agents that are not selected pay 0. Algorithm A is *exact*, if for declarations (a, v), $A(a, v)_i = a_i$ or $A(a, v)_i = \varnothing$ for all i. In analogy to [10] one obtains the following.

Theorem 1 *Let A be a monotone and exact algorithm for some utilitarian problem Π and single-minded agents. Then mechanism $M_A = (A, p^A)$ is truthful.*

Additional Definitions

In the *unsplittable flow problem* (UFP), an undirected graph $G = (V, E)$, $|E| = m$, $|V| = n$, with edge capacities b_e, $e \in E$, and a set K of $k \geq 1$ commodities described by terminal pairs $(s_i, t_i) \in V \times V$ and a demand d_i and a value c_i are given. One assumes that $\max_i d_i \leq \min_e b_e$, $d_i \in [0, 1]$ for each $i \in K = \{1, \ldots, k\}$, and $b_e \geq 1$ for all $e \in E$. Let $B = \min_e\{b_e\}$. A feasible solution is a subset $K' \subseteq K$ and a single flow s_i-t_i-path for each $i \in K'$, such that the demands of K' can simultaneously and unsplittably be routed along the paths and the capacities are not exceeded. The goal in *UFP*, called *B-bounded UFP*, is to maximize the total value of the commodities in K'. A generalization is allocating bandwidth for multicast communication, where commodity is a set of terminals that should be connected by a multicast tree.

Key Results

Monotone Approximation Schemes

Let Π be a given utilitarian (maximization) problem. Given declarations (a, v), let $Opt(a, v)$ denote an optimal solution to Π on this instance and $w(Opt(a, v))$ the corresponding social welfare. Assuming that A_Π is a pseudopolynomial exact algorithm for Π an algorithm A_Π^k and monotone FPTAS for Π is defined in Fig. 1.

Theorem 2 *Let Π be a utilitarian mechanism design problem among single-minded agents, A_Π monotone pseudopolynomial algorithm for Π with running time $poly(n, V)$, where*

Algorithm A_Π^k:
1 $\alpha_k := \frac{n}{\varepsilon \cdot 2^k}$;
2 **for** $i = 1, \ldots, n$ **do**
3 $v_i' := \min\{v_i, 2^{k+1}\}$;
4 $v_i'' := \lfloor \alpha_k \cdot v_i' \rfloor$;
5 return $A_\Pi(a, v'')$;

Algorithm A_Π^{FPTAS}
1 $V := \max_i v_i$, $Best := (\emptyset, \ldots, \emptyset)$, $best := 0$;
2 **for** $j = 0, \ldots, \lceil \log(1 - \varepsilon)^{-1} n \rceil + 1$ **do**
3 $k := \lfloor \log(V) \rfloor - j$;
4 **if** $w_k(A_\Pi^k(a, v)) > best$ **then**
5 $Best := A_\Pi^k(a, v)$; $best := w_k(A_\Pi^k(a, v))$;
6 return $Best$;

Utilitarian Mechanism Design for Single-Minded Agents, Fig. 1 A monotone FPTAS for utilitarian problem Π and single-minded agents

$V = \max_i v_i$, and assume that $V \le w(Opt(a, v))$ for declaration (a, v). Then A_{Π}^{FPTAS} is a monotone FPTAS for Π.

Theorem 2 can also be applied to minimization problems. Section "Applications" describes how these approximation schemes can be used for forward multi-unit auctions and job scheduling with deadlines.

Truthful Primal-Dual Mechanisms

For an instance $G = (V, E)$ of UFP defined above, let S_i be the set of all s_i-t_i-paths in G, and $S = \bigcup_{i=1}^{k} S_i$. Given $S \in S_i$, let $q_S(e) = d_i$ if $e \in S$, and $q_S(e) = 0$ otherwise. UFP is the following integer linear program (ILP)

$$\max \quad \sum_{i=1}^{k} c_i \cdot \left(\sum_{S \in S_i} x_S \right) \tag{1}$$

$$\text{s.t.} \quad \sum_{S : S \in S, e \in S} q_S(e) x_S \le b_e \quad \forall e \in E \tag{2}$$

$$\sum_{S \in S_i} x_S \le 1 \quad \forall i \in \{1, \ldots, k\} \tag{3}$$

$$x_S \in \{0, 1\} \quad \forall S \in S\,. \tag{4}$$

The linear programming (LP) relaxation is the same linear program with constraints (4) replaced with $x_S \ge 0$ for all $S \in S$. The corresponding dual linear program is

$$\min \quad \sum_{e \in E} b_e y_e + \sum_{i=1}^{k} z_i \tag{5}$$

$$\text{s.t.} \quad z_i + \sum_{e \in S} q_S(e) y_e \ge c_i \quad \forall i \in \{1, \ldots, k\}\ \forall S \in S_i \tag{6}$$

$$z_i, y_e \ge 0 \quad \forall i \in \{1, \ldots, k\}\ \forall e \in E. \tag{7}$$

Based on these LPs, Fig. 2 specifies a primal-dual mechanism for routing, called Greedy-1. Greedy-1 ensures feasibility by using y_e's: if an added set exceeded the capacity b_e of some $e \in E$, then this would imply the stopping condition already in the previous iteration. Using the weak duality of LPs the following result can be shown.

Theorem 3 *Greedy-1 outputs a feasible solution, and it is a $(\frac{e\gamma B}{B-1}(m)^{1/(B-1)})$-approximation algorithm if there is a polynomial time algorithm that finds a γ-approximate set S_i in line 4.*

In case of UFP $\gamma = 1$, as the shortest s_i-t_i-path computation finds set S_i in line 4 of Greedy-1. For multicast routing, this problem corresponds to the NP-hard Steiner tree problem, for which one can take $\gamma = 1.55$. Greedy-1 can easily be shown to be monotone in demands and valuations as required in Theorem 1. Thus it implies a truthful mechanism for allocating network resources. The commodities correspond to bidders, the terminal nodes of bidders are known, but the bidders might lie about their demands and valuations. In the multicast routing the set of terminals for each bidder is known but the demands and valuations are unknown.

Utilitarian Mechanism Design for Single-Minded Agents, Fig. 2 Truthful mechanism for network (multicast) routing. $e \approx 2.718$ is Euler number

Algorithm Greedy-1:

1 $\mathcal{T} := \emptyset$; $K := \{1, \ldots, k\}$;
2 **forall** $e \in E$ **do** $y_e := 1/b_e$;
3 **repeat**
4 **forall** $i \in K$ **do** $S_i := \operatorname{argmin} \left\{ \sum_{e \in S} y_e \;\middle|\; S \in S_i \right\}$;
5 $j := \operatorname{argmax} \left\{ \dfrac{c_i}{d_i \sum_{e \in S_i} y_e} \;\middle|\; i \in K \right\}$;
6 $\mathcal{T} := \mathcal{T} \cup \{S_j\}$; $K := K \setminus \{j\}$;
7 **forall** $e \in S_j$ **do** $y_e := y_e \cdot \left(e^{B-1} m\right)^{q_{S_j}(e)/(b_e - 1)}$;
8 **until** $\sum_{e \in E} b_e y_e \ge e^{B-1} m$ **or** $K = \emptyset$;
9 return $\mathcal{T}$.

Utilitarian Mechanism Design for Single-Minded Agents, Fig. 3 Truthful mechanism for multi-unit CAs among unknown single-minded bidders. For CAs without multisets: $q_S(e) \in \{0, 1\}$ for each $e \in U$, $S \in \mathcal{S}$

Algorithm Greedy-2:

$\mathcal{T} := \emptyset$;
forall $e \in U$ **do** $y_e := 1/b_e$;
repeat
  $S := \text{argmax}\left\{ \frac{c_S}{\sum_{e \in U} q_S(e) y_e} \;\middle|\; S \in \mathcal{S} \setminus \mathcal{T} \right\}$;
  $\mathcal{T} := \mathcal{T} \cup \{S\}$;
  forall $e \in S$ **do** $y_e := y_e \cdot (e^B m)^{q_S(e)/b_e}$;
until $\sum_{e \in U} b_e y_e \geq e^B m$;
return $\mathcal{T}$.

Corollary 1 *Given any $\epsilon > 0$, $B \geq 1 + \epsilon$, Greedy-1 is a truthful $O(m^{1/(B-1)})$-approximation mechanism for UFP (unicast routing) as well as for the multicast routing problem, where the demands and valuations of the bidders are unknown.*

When B is large, $\Omega(\log m)$, then the approximation factor in Corollary 1 becomes constant. Azar et al. [4] presented further results in case of large B. Awerbuch et al. [3] gave randomized online truthful mechanisms for uni- and multicast routing, obtaining an expected $O(\log(\mu m))$-approximation if $B = \Omega(\log m)$, where μ is the ratio of the largest to smallest valuation. Their approximation holds in fact with respect to the revenue of the auctioneer, but they assume that the demands are known to the mechanism. Bartal et al. [5] give a truthful $O(B \cdot (m/\theta)^{1/(B-2)})$-approximation mechanism for UFP with unknown valuations and demands, where $\theta = \min_i \{d_i\}$.

Greedy-1 can be modified to give truthful mechanisms for multi-unit CAs among unknown single-mined bidders. (In the case of *unknown* single-minded bidders, the bidders have as private data not only their valuations (as in the case of *known* single-minded bidders) but also the sets they demand.) Archer et al. [2] used randomized rounding to obtain a truthful mechanism for multi-unit CAs, but only in a probabilistic sense and only for known bidders. Multi-unit CA among single-minded bidders is a special case of ILP (1)–(4), where $|S_i| = 1$ for each $i \in K$, and $q_S(e) \in \{0, 1\}$ for each $e \in U$, $S \in \mathcal{S}$ (E is U in CAs). A bid of bidder $i \in K$ is $(a_i, v_i) = (S, c_S)$, $S \in S_i$, and $c_S = c_i$ is the valuation. The relation $R_{\preceq}$ is $\subseteq$. Algorithm Greedy-2 in Fig. 3 is exact and monotone for CAs with unknown single-minded bidders, as needed in Theorem 1.

Theorem 4 *Algorithm Greedy-2 is a truthful $O(m^{\frac{1}{B}})$-approximation mechanism for multi-unit CAs among unknown single-minded bidders.*

Bartal et al. [5] presented a truthful mechanism for this problem among unknown single-minded bidders which is $O(B \cdot m^{1/(B-2)})$-approximate. (It works in fact for more general bidders.)

Applications

Applications of the techniques described above are presented and a short survey of other results.

Applications of Monotone Approximation Schemes

In a *forward multi-unit auction* a single auctioneer wants to sell m identical items to n possible buyers (bidders). Each single-minded bidder specifies the number of items she is interested in and a price she is willing to pay. Elements in the introduced notation correspond to the requested and allocated numbers of items. Relation $R_{\preceq}$ describes that bidder i requesting q_i items will be satisfied also by any larger number of items. Mu'alem and Nisan [11] give a 2-approximate monotone algorithm for this problem. Theorem 2 gives a monotone FPTAS for multi-unit auctions among unknown single-minded bidders. This FP-

TAS is truthful with respect to agents where both the number of items and price are private.

In *job scheduling with deadlines (JSD)*, each agent i has a job with running time t_i, deadline d_i and a price v_i she is willing to pay if her job is processed by deadline d_i. Element a_i is defined as $a_i = (t_i, d_i)$. Output for agent i is a time slot for processing i's job. For two elements $a_i = (t_i, d_i)$ and $a'_i = (t'_i, d'_i)$ one has $a_i \preceq a'_i$ if $t_i \leq t'_i$ and $d_i \geq d'_i$. Theorem 2 leads to a monotone FPTAS, which, however, is not exact (see Theorem 1) with respect to deadlines, and so it is a truthful mechanism only if the deadlines are known. The techniques of Theorem 2 apply also to minimization mechanism design problems with a single buyer, such as reverse multi-unit auctions, scheduling to minimize tardiness, constrained shortest path and minimum spanning tree problems [6].

Applications of the primal dual algorithms

The applications of the primal dual algorithms are combinatorial auctions and auctions for unicast and multicast routing. As these applications are tied very much to the algorithms, they have already been presented in section "Key Results".

Survey of Other Results

First truthful mechanisms for single-minded CAs were designed by Lehmann et al. [10], where they introduced the concept of single-minded agents, identified the role of monotonicity, and used greedy algorithms to design truthful mechanisms. Better approximation ratios of these greedy mechanisms were proved by Krysta [9] with the help of LP duality. A tool-box of techniques for designing truthful mechanisms for CAs was given by Mu'alem and Nisan [11].

The previous section presented a monotone FPTAS for job scheduling with deadlines where jobs are selfish agents and the seller offers the agents the facilities to process their jobs. Such scenarios when jobs are selfish agents to be scheduled on (possibly selfish) machines have been investigated further by Andelman and Mansour [1], see also references therein.

So far social welfare was mostly assumed as the objective, but for a seller probably more important is to maximize her revenue. This objective turns out to be much harder to enforce in mechanism design. Such truthful (in probabilistic sense) mechanisms were obtained for auctioning unlimited supply goods among one-parameter agents [7, 8]. Another approach to maximizing seller's revenue is known as optimal auction design [12]. A seller wants to auction a single good among agents and each agent has a private value for winning the good. One assumes that the seller knows a joint distribution of those values and wants to maximize her expected revenue [13, 14].

Cross-References

Mechanisms that approximately maximize revenue for unlimited-supply goods as of Goldberg, Hartline and Wright [8] are presented in entry ▸ Competitive Auction.

Recommended Reading

1. Andelman N, Mansour Y (2006) A sufficient condition for truthfulness with single parameter agents. In: Proceedings of the 8th ACM conference on electronic commerce (EC), Ann Arbor, June 2006
2. Archer A, Papadimitriou CH, Talwar K, Tardos E (2003) An approximate truthful mechanism for combinatorial auctions with single parameter agents. In: Proceedings of the 14th annual ACM–SIAM symposium on discrete algorithms (SODA), Baltimore, pp 205–214
3. Awerbuch B, Azar Y, Meyerson A (2003) Reducing truth-telling online mechanisms to online optimization. In: Proceedings of the 35th annual ACM symposium on theory of computing (STOC), San Diego
4. Azar Y, Gamzu I, Gutner S (2007) Truthful unsplittable flow for large capacity networks. In: Proceedings of the 19th annual ACM symposium on parallelism in algorithms and architectures (SPAA), pp 320–329
5. Bartal Y, Gonen R, Nisan N (2003) Incentive compatible multi unit combinatorial auctions. In: Proceedings of the 9th conference on theoretical aspects of rationality and knowledge (TARK), ACM Press, pp 72–87. http://doi.acm.org/10.1145/846241.846250
6. Briest P, Krysta P, Vöcking B (2005) Approximation techniques for utilitarian mechanism design. In: Proceedings of the 37th annual ACM symposium on theory of computing (STOC), pp 39–48

7. Fiat A, Goldberg AV, Hartline JD, Karlin AR (2002) Competitive generalized auctions. In: Proceedings of the 34th annual ACM symposium on theory of computing (STOC), pp 72–81
8. Goldberg AV, Hartline JD, Wright A (2001) Competitive auctions and digital goods. In: Proceedings of the 12th annual ACM–SIAM symposium on discrete algorithms (SODA), pp 735–744
9. Krysta P (2005) Greedy approximation via duality for packing, combinatorial auctions and routing. In: Proceedings of the 30th international conference on mathematical foundations of computer science (MFCS). Lecture notes in computer science, vol 3618, pp 615–627
10. Lehmann DJ, O'Callaghan LI, Shoham Y (1999) Truth revelation in approximately efficient combinatorial auctions. In: Proceedings of the 1st ACM conference on electronic commerce (EC), pp 96–102
11. Mu'alem A, Nisan N (2002) Truthful approximation mechanisms for restricted combinatorial auctions. In: Proceedings of the 18th national conference on artificial intelligence. AAAI, pp 379–384
12. Myerson RB (1981) Optimal auction design. Math Oper Res 6:58–73
13. Ronen A (2001) On approximating optimal auctions (extended abstract). In: Proceedings of the 3rd ACM conference on electronic commerce (EC), pp 11–17
14. Ronen A, Saberi A (2002) On the hardness of optimal auctions. In: Proceedings of the 43rd annual IEEE symposium on foundations of computer science (FOCS), pp 396–405

V

Vector Bin Packing

David S. Johnson
Department of Computer Science, Columbia University, New York, NY, USA
AT&T Laboratories, Algorithms and Optimization Research Department, Florham Park, NJ, USA

Keywords

Approximation algorithms; Bin packing; Resource constraints; Shared hosting platforms; Worst-case analysis

Years and Authors of Summarized Original Work

1976; Garey, Graham, Johnson, Yao
1977; Kou, Markowski
1977; Maruyama, Chang, Tang
1981; de la Vega, Lueker
1987; Yao
1990; Csirik, Frenk, Labbé, Zhang
1997; Woeginger
2001; Caprara, Toth
2004; Chekuri, Khanna
2009; Bansal, Caprara, Sviridenko
2010; Stillwell, Schanzenback, Vivien, Casanova
2011; Panigrahy, Talwar, Uyeda, Wieder

Problem Definition

In the *vector bin packing problem*, we are given an integral dimension $d \geq 1$ and a list $L = (x_1, x_2, \ldots, x_n)$ of items, where each item is a d-dimensional tuple $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d})$ with rational entries $x_{i,j} \in [0, 1]$. The goal is to assign the items to a minimum number of multidimensional *bins*, where if X is the set of items assigned to a bin, we must have, for each j, $1 \leq j \leq d$,

$$\sum_{x_i \in X} x_{i,j} \leq 1.$$

Note that when $d = 1$, the vector bin packing problem reduces to the classic (one-dimensional) *bin packing* problem.

One potential application of the vector bin packing problem is that of assigning jobs to servers in a shared hosting platform, where each job may require a specific number of cycles per second and specific amounts of memory, bandwidth, and other resources [12]. Here the servers

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

would correspond to the bins, the dimension d is the number of resources, the items are the jobs, and $x_{i,j}$ is the fraction of the total amount of a server's jth resource that job x_i requires.

In the early literature, this problem was often called the *multidimensional bin packing* problem. That term, however, is now more typically reserved for the related problem where the items are d-dimensional rectangular parallelepipeds (rectangles, when $d = 2$), the bins are d-dimensional unit cubes, and the items assigned must not only be assigned to bins but also to specific positions in the bins, in such a way that no point in any bin is in the interior of more than one item. With vector bin packing, in contrast, the dimensions are all independent and there is no geometric interpretation of the items.

Key Results

As a generalization of bin packing, vector bin packing is clearly NP-hard in the strong sense, and so most of the research on this problem has been directed toward the study of approximation algorithms for it. This will be the primary topic in this entry. Most of the theoretical results concerning these algorithms can be expressed in terms of *asymptotic worst-case ratios*. For a given algorithm A and a list of items L, let $A(L)$ denote the number of bins used by A for L. Let *OPT(L)* denote the optimal number of bins for list L. We define the asymptotic worst-case ratio $R_A^\infty(d)$ for algorithm A on d-dimensional instances as follows.

$$R_A^N(d) = \max\left\{\frac{A(L)}{OPT(L)} : L \text{ is a list of } d\text{-dimensional items with } OPT(L) = N\right\}$$

$$R_A^\infty(d) = \limsup_{N\to\infty} R_A^N(d)$$

Generalizations of Classical Bin Packing Algorithms

Generalizing First Fit and First Fit Decreasing

Several classic one-dimensional bin packing algorithms have been generalized to vector bin packing. Imagine we have a potentially infinite sequence of empty bins $B_1, B_2, \ldots$, and let $X_{h,j}$ denote the total amount of resource j used by the items currently assigned to B_h. In the generalized "First Fit" algorithm, the first item goes in bin B_1, and thereafter each item goes into the lowest-index bin into which it can be legally placed, subject to the resource constraints. In generalized "Best Fit," each item is assigned to a bin with the maximum value of $\sum_{j=1}^{d} X_{h,j}$ among those to which it can legally be added, ties broken in favor of the smallest index h.

As in the one-dimensional case, a plausible way to improve the above two online algorithms is to first reorder the list in decreasing order, and then apply the packing algorithm. Now, however, there are a variety of ways to define "decreasing order," each leading to different algorithms. For example, in FFDmax items are ordered by nonincreasing value of $\max_{j=1}^{d} x_{i,j}$ and then FF is applied. Similarly, in FFDsum, the items are ordered by nonincreasing value of $\sum_{j=1}^{d} x_{i,j}$ and, in FFDprod, they are ordered by nonincreasing value of $\prod_{j=1}^{d} x_{i,j}$. In FFDlex, they are ordered so that x_i precedes $x_{i'}$ only if either $x_{i,j} = x_{i',j}$, $1 \le j \le d$, or there is a $j^* \le d$ such that $x_{i,j} = x_{i',j}$, $1 \le j < j^*$ and $x_{i,j^*} < x_{i',j^*}$. The algorithms BFDmax, BFDsum, BFCprod, and BFDlex are defined analogously, with Best Fit being used to pack the reordered list instead of First Fit.

Call an algorithm "reasonable" if it produces packings in which no two bins can be combined, that is, are such that all the items contained in the two would collectively fit together in a single bin [9]. All of the above algorithms are easily seen to be reasonable, and, indeed, any vector bin packing algorithm has a "reasonable" counterpart that uses no more bins and spends at most $O(n^2 d)$ additional time (in a final pass that

combines legally combinable pairs of bins as long as such pairs exist). A general upper bound on asymptotic worst-case behavior is the following.

Theorem 1 ([9]) *If A is a reasonable vector bin packing algorithm, then for all $d \geq 1$,*

$$R_A^\infty(d) \leq d + 1.$$

Unfortunately, none of the above algorithms are much better.

Theorem 2 ([9, 11]) *For each of the 10 algorithms defined above and all $d \geq 1$,*

$$R_A^\infty(d) \geq d.$$

Tighter bounds have been proved for two of the algorithms.

Theorem 3 ([6]) *For all $d \geq 1$, $R_{\mathrm{FF}}^\infty(d) = d + \frac{7}{10}$.*

Theorem 4 ([6]) *For all $d \geq 1$, $d + \frac{d-1}{d(d+1)} \leq R_{\mathrm{FFDmax}}^\infty(d) \leq d + \frac{1}{3}$.*

Note that the classic one-dimensional bin packing results of [8] yield $R_{\mathrm{FF}}^\infty(1) = 17/10$ (the precise specialization of Theorem 3) and $R_{\mathrm{FFD}}^\infty(1) = 11/9$ (a tighter result than the specialization of Theorem 4). Matching upper and lower bounds are not known for $R_{\mathrm{FFDmax}}^\infty(d)$ for any $d > 1$. In special cases, however, the lower bounds can be improved. It was observed in [6] that the lower bounds for $d \in \{2, 3\}$ could be increased to $d + 11/60$ using ideas from [8]. And Csirik et al. [4] showed that for odd $d \geq 5$, the lower bound of Theorem 2 could be increased by $1/(d(d+1)(d+2))$.

Generalizing the de la Vega and Lueker Asymptotic Approximation Scheme

In [5], de la Vega and Lueker devised an "asymptotic polynomial-time approximation scheme" (APTAS) for one-dimensional bin packing, that is, a collection of polynomial-time algorithms A_ϵ with $R_{A_\epsilon}^\infty(1) \leq 1 + \epsilon$ for all $\epsilon > 0$. In that same paper, they also showed how to generalize the algorithms to provide a collection of vector bin packing algorithms B_ϵ such that for each ϵ and each integer $d \geq 1$, $R_{B_\epsilon}^\infty(d) \leq d + \epsilon$.

The algorithm B_ϵ is quite simple. Divide L into d sublists, $L_1, L_2, \ldots, L_d$, where L_j consists of all those items x for which j is the index of the dimension with the largest entry in the corresponding tuple, ties broken arbitrarily. Then apply $A_{\epsilon/d}$ to each list L_j separately, viewed as an instance of one-dimensional bin packing with the size of item x_i being $x_{i,j}$, and output the union of the d packings. Unfortunately, although the running times for the B_ϵ's are linear in dn, they contain additive constants that are potentially exponential in $(d/\epsilon)^2$, and so they may not be practical for small ϵ.

In contrast, FF, FFDmax, and all their variants mentioned in the previous section have straightforward $O(dn^2)$ implementations, and, although the data structures that allow them to be sped up to $O(n \log n)$ when $d = 1$ do not extend to higher dimensions, speedups should be possible by using d-dimensional dynamic range searching procedures to identify the set of bins that can contain the next item to be packed [11].

Hardness of Approximation Results

In [14], Yao observed that, under a standard decision tree model of computation, any vector bin packing algorithm A that has $R_A^\infty(d) < d$ for all d cannot have $o(n \log n)$ running time. This is not much of a constraint, however, since almost all the algorithms that have been proposed for Vector Bin Packing are slower than this. For those algorithms, a weaker bound applies. Assuming P $\neq$ NP, no polynomial-time vector bin packing algorithm A can have $R_A^\infty(d) < \sqrt{d} - \epsilon$ for all d and any $\epsilon > 0$. This follows from a straightforward reduction of graph coloring to vector bin packing and a result of Zuckerman [15] for the former [3]. Under the same assumption, there can be no APTAS for any fixed $d \geq 2$ [13].

Algorithms with $R_A^\infty(d) < d$

Chekuri and Khanna [3] devised the first polynomial-time algorithms to guarantee $R_A^\infty(d) < d$ for all sufficiently large d.

Theorem 5 ([3]) *For any* $\epsilon > 0$ *there is a polynomial-time algorithm* C_ϵ *such that*

$$R^{\infty}_{C_\epsilon}(d) < \epsilon \cdot d + \ln(\epsilon^{-1}) + 2.$$

The algorithm works in three phases. The first considers the following linear program (LP). Let $z_{i,k}$ be a decision variable with value 1 if item x_1 is packed in bin j. The LP's constraints are

$$\sum_{k=1}^{m} z_{i,k} = 1, \quad 1 \le i \le n \tag{1}$$

$$\sum_{i=1}^{n} x_{i,j} \cdot z_{i,k} \le 1, \quad 1 \le k \le m, 1 \le j \le d \tag{2}$$

$$z_{i,k} \ge 0, \quad 1 \le i \le n, 1 \le k \le m \tag{3}$$

This is the LP relaxation of an integer program (with $z_{i,k} \in \{0, 1\}$) which has a feasible solution if and only if our list can be packed into m bins. The first set of constraints insures that each item is packed into exactly one bin. The second set insures that all resource constraints are satisfied by the packing.

Let M be the least value of m such that this LP is feasible. Then we clearly must have $OPT(L) \ge M$. Moreover we can in polynomial time determine M and a basic feasible solution for the corresponding LP, by using binary search and a polynomial time LP-solver. In this basic feasible solution, there will be at most $n + dM$ positive variables (the number of nontrivial constraints). Since each of the n items x_i by (1) must be assigned to at least one bin, at least one of the variables $z_{i,k}$ must be positive for each i, meaning that at most dM of the items can be assigned to more than one bin. That leaves $n - dM$ items assigned to exactly one bin, and consequently our LP solution yields a feasible packing of these items into $M \le OPT(L)$ bins, which is the output of our first phase. The remaining dM or fewer items will be packed into additional bins in two additional phases as follows.

Let $k = \lceil 1/\epsilon \rceil$. While there are at least k unpacked items that will fit in a single bin, find such a set and pack them all in a new bin (Phase 2). Otherwise, find a maximum size set of unpacked items that will fit in a bin, assign them to a new bin, and repeat until all items are packed (Phase 3). Note that in both of these phases the next set of items to be packed can be found in time $O(n^k kd)$, which is polynomial for fixed ϵ. Thus the overall time for the algorithm is itself polynomial for fixed ϵ. Phase 2 creates at most $dM/k < \epsilon d \cdot OPT(L)$ bins. Phase 3 can be interpreted as implementing the Greedy algorithm for Set Covering, as applied to the instance in which the elements to be covered are the items left to be packed after Phase 2, the sets are the collections of those items which will fit in a bin, and no set has size exceeding $k - 1$. Thus, by standard results about Greedy Set Covering (see [7] for example), the number of bins added in this phase is less than $(\ln(k-1)+1) \cdot OPT(L) < (\ln(1/\epsilon) + 1) \cdot OPT(L)$. Adding up the above three terms yields the claimed theorem.

Note that if we set $\epsilon = 1/d$ in the above, we get a series of algorithms $C_{1/d}$ with $R^{\infty}_{C_{1/d}}(d) \le \ln(d) + 3$, where the running time of each is polynomial in n, although exponential in d. A slight improvement to this has recently been obtained by Bansal et al. [1]. They devise algorithms $D_{d,\epsilon}$ that run in polynomial time for fixed d and ϵ (although exponential in both) that have $R^{\infty}_{D_{d,\epsilon}} \le \ln(d + \epsilon) + 1 + \epsilon$, which, for $d \ge 2$, already beats $\ln(d) + 3$ when $\epsilon = 1$.

Experimental Results

There have been several experimental studies of approximation algorithms for vector bin packing [2, 10–12]. These studies were for the most part limited to $d \le 10$ and $n \le 500$, which may well make sense in the context of the proposed applications, and used distinct sets of randomly generated test instances. In two cases ([10] and [12]), the algorithms were compared using objective functions other than the number of bins packed. Nevertheless, certain common conclusions emerge. The FFD algorithms in particular yielded substantially better packings than worst-case analysis suggests. Both [11] and [12] suggest, however, that a different class of algorithms, ones that attempt to keep the bins as "balanced"

as possible, may perform even better. An example of such an algorithm is the "norm-based greedy" algorithm of [11], which packs the bins one-by-one, at each step adding to the current bin B_h that item x_i that fits and yields the smallest weighted L^2 norm for the resulting "gap vector" $(X_{h,1} - x_{i,1}, X_{h,2} - x_{i,2}, \ldots, X_{h,d} - x_{i,d})$. For more details, see [11]. As for the algorithms described above with $R_A^\infty(d) < d$ for large d, the only ones with hopes of feasible running times are the algorithms $C_{1/d}$ from [3] when n^d is of manageable size. Limited experiments from [12] indicate that the packings these algorithms produce are not competitive.

Cross-References

- ▸ Approximation Schemes for Bin Packing
- ▸ Bin Packing
- ▸ Graph Coloring
- ▸ Greedy Set-Cover Algorithms

Recommended Reading

1. Bansal N, Caprara A, Sviridenko M (2009) A new approximation method for set covering problems, with applications to multidimensional bin packing. SIAM J Comput 39:1256–1278
2. Caprara A, Toth P (2001) Lower bounds and algorithms for the 2-dimensional vector packing problem. Discret Appl Math 111:231–262
3. Chekuri C, Khanna S (2004) On multidimensional packing problems. SIAM J Comput 33:837–851
4. Csirik WF, Frenk JBG, Labbé M, Zhang S (1990) On the multidimensional vector packing. Acta Cybern 9:361–369
5. de la Vega J, Lueker GS (1981) Bin packing can be solved within $1 + \epsilon$ in linear time. Combinatorica 1:349–355
6. Garey MR, Graham RL, Johnson DS, Yao AC-C (1976) Resource constrained scheduling as generalized bin packing. J Comb Theory (A) 21:257–298
7. Johnson DS (1974) Approximation algorithms for combinatorial problems. J Comput Syst Sci 9:256–278
8. Johnson DS, Demers A, Ullman JD, Garey MR, Graham RL (1974) Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM J Comput 3:299–325
9. Kou LT, Markowski G (1977) Multidimensional bin packing algorithms. IBM J Res Dev 21:443–448
10. Maruyama K, Chang SK, Tang DT (1977) A general packing algorithm for multidimensional resource requirements. Int J Comput Inf Sci 6:131–149
11. Panigrahy R, Talwar K, Uyeda L, Wieder U (2011) Heuristics for vector bin packing. Unpublished manuscript. (Available on the web)
12. Stillwell M, Schanzenbach D, Vivien F, Casanova H (2010) Resource allocation algorithms for virtualized service hosting platforms. J Parallel Distrib Comput 70:962–974
13. Woeginger GJ (1997) There is no asymptotic PTAS for two-dimensional vector packing. Inf Proc Lett 64:293–297
14. Yao AC-C (1980) New algorithms for bin packing. J ACM 27:207–227
15. Zuckerman D (2007) Linear degree extractors and the inapproximability of max clique and chromatic number. Theory Comput 3:103–128

Vector Scheduling Problems

Tjark Vredeveld
Department of Quantitative Economics, Maastricht University, Maastricht, The Netherlands

Keywords

Approximation schemes; Vector scheduling

Years and Authors of Summarized Original Work

2004; Chekuri, Khanna

Problem Definition

Vector scheduling is a multidimensional extension of traditional machine scheduling problems. Whereas in traditional machine scheduling a job only uses a single resource, normally time, in vector scheduling a job uses several resources. In traditional scheduling, the load of a machine is the total resource consumption by the jobs that it serves. In vector scheduling, we define the load of a machine as the maximum resource usage over all resources of the jobs that are served by this machine. In the setting that we consider here, the

makespan, which is normally defined to be the time by which all jobs are completed, is equal to the maximum machine load.

To define the vector scheduling problem that we consider more formally, we let $\|\mathbf{x}\|_\infty$ denote the standard ℓ_∞-norm of the vector $\mathbf{x}$. In the vector scheduling problem, the input consists of a set J of n jobs, where each job j is associated with a d-dimensional vector $\mathbf{p_j} \in [0,1]^d$, and m identical machines. The goal is to find an assignment of the jobs to the m machines such that $\max_{1\le i\le m} \| \sum_{j\in M_i} \mathbf{p_j}\|_\infty$ is minimized, where M_i denotes the set of jobs that are assigned to machine i.

The traditional machine scheduling problem corresponds to the case $d = 1$, and this is known to be strongly NP-hard [8]. For $d = 1$, Graham's well-known list scheduling algorithm has a performance guarantee of 2 [9] and Hochbaum and Shmoys developed at PTAS [10]. For general vector scheduling, Graham's list scheduling algorithm can be extended to the d-dimensional case, having a performance guarantee of $d + 1$. In this entry we focus on the work of Chekuri and Khann [5], who developed a PTAS for fixed d and gave a polylogarithmic approximation factor for the case of general d.

Key Results

Constant Dimension *d*

Chekuri and Khann [5] designed an approximation scheme that runs in polynomial time whenever the dimension d of the job vectors is constant.

Theorem 1 *For any $\epsilon > 0$, there exists an $(1+\epsilon)$-approximation algorithm that has a running time of $O((nd/\epsilon)^{O(s)})$, where s is in $O\left((\frac{\log(d/\epsilon)}{\epsilon})^d\right)$.*

The proof of this theorem is a nontrivial generalization of the ideas that Hochbaum and Shmoys used for the 1-dimensional case [10]. In this primal-dual approach, the main idea is to view the scheduling problem as a bin-packing problem in which the jobs need to be packed into a number of bins of a certain capacity B. If all jobs fit into m bins, then the makespan is bounded by B. Hochbaum and Shmoys gave an algorithm that determines whether the jobs fit into m bins of capacity $(1+\epsilon)B$ or the jobs need to be packed into at least $m+1$ bins of capacity B. Chekuri and Khanna extended this idea by using d-dimensional bins, where the jobs assigned to one bin should have a total resource usage of at most B in any of the d dimensions. By standard scaling techniques, we assume w.l.o.g. that $B = 1$.

Like in the 1-dimensional case, Chekuri and Khanna divide the jobs in small and large jobs, where the size of a job is based on the ℓ_∞ norm. They first do a preprocessing step in which each coordinate of the vectors is set to 0 whenever it is too small compared to the maximum value of the coordinates in the same vector. To find an $(1+\epsilon)$-approximation, the algorithm performs two stages. In the first stage all large jobs will be assigned to the machines, and in the second stage all small jobs will be assigned to the machines. Whereas in the 1-dimensional case the assignment of the small jobs can be done greedily on top of the large jobs, for $d \ge 2$ the interaction between the two stages needs to be taken into account.

To accommodate this interaction, Chekuri and Khanna define a *capacity configuration* as a d-tuple $(c_1,\ldots,c_d)$ such that c_k is an integer between 0 and $\lceil 1/\epsilon\rceil$. A set of jobs S can be feasible, scheduled on one machine according to a capacity configuration $(c_1,\ldots,c_d)$ when for any dimension k, it holds that $\left(\sum_{j\in S}\mathbf{p_j}\right)_k \le a_k\cdot\epsilon$, i.e., in each dimension k the resource usage is not more than $c_k\cdot\epsilon$. The number of distinct capacity configurations is given by $t = (1+\lceil 1/\epsilon\rceil)^d$.

A capacity configuration describes approximately how a machine is filled. As there are m machines available to process the jobs, a *machine configuration* can be described by a t-tuple $(m_1,\ldots,m_t)$, satisfying $m_i \ge 0$ and $\sum_i m_i = m$, where m_i denotes the number of machines of the ith capacity configuration. The number

of distinct machine configurations is certainly bounded from above by m^t.

After the preprocessing of the vectors and the splitting of the jobs in small and large, it needs to be determined whether all large jobs can be scheduled according to a machine configuration M. As a first step, all nonzero elements of the large vectors are rounded (down) to the begin points of geometrically increasing intervals. Moreover, as the vectors are in some sense large, not too many vectors can be scheduled on one machine. Therefore, using a dynamic programming approach, one can approximately determine whether the set of large vectors can be scheduled according to machine configuration M.

When the set of large jobs are scheduled such that a certain machine i is scheduled according to a capacity configuration $(c_1, \ldots, c_d)$, then the small jobs on this machine need to be scheduled according to the *empty capacity configuration*, i.e., the capacity configuration $(1 + \lceil 1/\epsilon \rceil) \cdot (1, 1, \ldots, 1) - (c_1, \ldots, c_d)$. Given a machine configuration M, we let $\bar{M}$ denote the corresponding machine configuration as the one obtained by taking the empty capacity configurations for each of the machines in M.

To see whether the small jobs can be scheduled according to a machine configuration $\bar{M}$, Chekuri and Khanna present an integer programming (ILP) formulation that assigns the vectors to the machines. Moreover, they show that solving the LP relaxation of this ILP formulation and distributing the fractionally assigned vectors equally over the machines result in a solution in which each dimension of each machine is only overloaded by a factor of $(1 + \epsilon)$.

Once they have found a machine configuration M according to which the large jobs can be scheduled and corresponding machine configuration $\bar{M}$ according to which the small jobs can be scheduled, Chekuri and Khanna have shown that all jobs can be scheduled such that the load of any machine does not exceed $1 + \epsilon$. If for all machine configurations M and corresponding machine configuration $\bar{M}$ the large jobs cannot be scheduled according to M or the small jobs cannot be scheduled according to $\bar{M}$, then the vectors cannot be scheduled such that the load of each machine is at most 1.

General Dimension *d*

For the general case in which the dimension d of the vectors is not restricted to be a constant, Chekuri and Khanna present several approximation algorithms. Also for the general case, they assume that all vectors can be scheduled such that the makespan is bounded by 1. For two algorithms, they use as a subroutine an approximation algorithm for finding a set of vectors S that maximizes the volume of these vectors, i.e., the sum of all coordinates of all these vectors $\sum_{j \in S} \sum_{k=1}^{d} (\mathbf{p_j})_k$, restricted to $\| \sum_{j \in S} \mathbf{p_j} \|_\infty \leq 1$. This resulted in the following results.

Theorem 2 *There exists a polynomial-time $O(\log^2 d)$-approximation algorithm for the vector scheduling problem.*

Theorem 3 *There exists a $O(\log d)$-approximation algorithm for the vector scheduling problem that runs in time polynomial in n^d.*

These approximation results are good when d is small compared to the number of machines m. On the other hand, Chekuri and Khanna also give a randomized algorithm, which just assigns each job uniformly at random to one of the machines, obtaining a performance guarantee that is better when d is large compared to m.

Theorem 4 *There exists a randomized algorithm that has a performance guarantee of $O(\log d m / \log \log d m)$ with high probability.*

Finally, there is also a hardness result for the vector scheduling problem.

Theorem 5 *For any constant $\rho > 1$, there is no polynomial-time approximation algorithm with a performance guarantee of ρ, unless NP = ZPP.*

Extensions

Epstein and Tassa [6, 7] extended the vector scheduling problem to deal with more general objective functions. Instead of defining the load of a machine as the maximum resource usage over all resources of the jobs that are

served by this machine, they defined the load as the sum of the vectors $\mathbf{p_j}$ assigned to the machine. That is, the load itself is now also a d-dimensional vector. Letting $\mathbf{l_i} = \sum_{j \in M_i} \mathbf{p_j}$ denote the load of machine i, then in [6] they gave PTASes for several objective functions of the form $F(S) = f(g(\mathbf{l_1}), \ldots, g(\mathbf{l_m}))$. Note that the vector scheduling problem as discussed in this entry is equal to the case that $f = g = \max$. In [7], they extended their results to the more general case where the function g may vary per machine, i.e., $F(S) = f(g_1(\mathbf{l_1}), \ldots, g_m(\mathbf{l_m}))$.

Bonifaci and Wiese [4] extended the vector scheduling problem to the ℓ_p norm and the case of unrelated machines. That is, a job j has a d-dimensional resource usage $\mathbf{p_{ij}}$ on machine i. They considered the case in which the number of types of machines is constant: on the same type of machine, a certain job has the same resource usage. Moreover, they restricted themselves to the case of having only a constant number of resources, that is, the vectors $\mathbf{p_{ij}}$ are d dimensional for a constant d. For this setting, they developed a PTAS.

The PTAS of Chekuri and Khanna has a running time that is doubly exponential in d. Bansal, Vredeveld, and Van der Zwaan [2] showed that this double exponential dependence on d is necessary. For $\epsilon < 1$, they showed that unless the exponential time hypothesis fails, there is no $(1 + \epsilon)$-approximation algorithm with running time $\exp(o(\lfloor 1/\epsilon \rfloor^{d/3}))$. Moreover, they showed that unless NP has subexponential algorithms, no $(1+\epsilon)$-approximation algorithm exists with running time $\exp(\lfloor 1/\epsilon \rfloor^{o(d)})$. These lower bounds even hold for the case that ϵm more machines are allowed, for sufficiently small $\epsilon > 0$. Moreover, they also gave a $(1+\epsilon)$-approximation algorithm with running time $\exp((1/\epsilon)^{O(d \log \log d)} + nd)$, which is the first efficient approximation scheme (EPTAS) for the problem with constant d.

Open Problems

The gap between the lower bounds and upper bounds on the running time of $(1 + \epsilon)$-approximation algorithms has almost been closed. The question remains whether this is also the case when instead of the ℓ_∞-norm, the ℓ_p-norm is minimized. Furthermore, it would be interesting to know whether one can obtain better running times when the vectors are highly structured. These highly structured vectors may occur, for example, in applications of real-time scheduling; see, e.g., [1, 3].

Cross-References

- Approximation Schemes for Bin Packing
- Approximation Schemes for Makespan Minimization
- Vector Bin Packing

Recommended Reading

1. Bansal N, Rutten C, van der Ster S, Vredeveld T, van der Zwaan R (2014) Approximating real-time scheduling on identical machines. In: LATIN 2014, Montevideo. LNCS, vol 8392. Springer, Heidelberg, pp 550–561
2. Bansal N, Vredeveld T, van der Zwaan R (2014) Approximating vector scheduling: almost matching upper and lower bounds. In: LATIN 2014, Montevideo. LNCS, vol 8392. Springer, Heidelberg, pp 47–59
3. Baruah S, Bonifaci V, D'Angelo G, Marchetti-Spaccamela A, van der Ster S, Stougie L (2011) Mixed criticality scheduling of sporadic task systems. In: Proceedings of the 19th annual European symposium on algorithms (ESA), Saarbrücken. LNCS, vol 6942. Springer, Heidelberg, pp 555–566
4. Bonifaci V, Wiese A (2012) Scheduling unrelated machines of few different types. CoRR abs/1205.0974
5. Chekuri C, Khanna S (2004) On multidimensional packing problems. SIAM J Comput 33(4):837–851
6. Epstein L, Tassa T (2003) Vector assignment problems: a general framework. J Algorithms 48(2):360–384
7. Epstein L, Tassa T (2006) Vector assignment schemes for asymmetric settings. Acta Inform 42(6–7):501–514
8. Garey M, Johnson D (1978) "Strong" NP-completeness results: motivation, examples, and implications. J ACM 25:499–508
9. Graham R (1966) Bounds for certain multiprocessor anomalies. Bell Syst Tech J 45:1563–1581
10. Hochbaum DS, Shmoys DB (1987) Using dual approximation algorithms for scheduling problems theoretical and practical results. J ACM 34(1): 144–162

Vertex Cover Kernelization

Jianer Chen
Department of Computer Science, Texas A&M University, College Station, TX, USA

Keywords

Vertex cover data reduction; Vertex cover preprocessing

Years and Authors of Summarized Original Work

2004; Abu-Khzam, Collins, Fellows, Langston, Suters, Symons

Problem Definition

Let G be an undirected graph. A subset C of vertices in G is a *vertex cover* for G if every edge in G has at least one end in C. The (parametrized) VERTEX COVER problem is for each given instance (G, k), where G is a graph and $k \geq 0$ is an integer (the parameter), to determine whether the graph G has a vertex cover of at most k vertices.

The VERTEX COVER problem is one of the six "basic" NP-complete problems according to Garey and Johnson [4]. Therefore, the problem cannot be solved in polynomial time unless P = NP. However, the NP-completeness of the problem does not obviate the need for solving it because of its fundamental importance and wide applications. One approach was initiated based on the observation that in many applications, the parameter k is small. Therefore, by taking the advantages of this fact, one may be able to solve this NP-complete problem effectively and practically for instances with a small parameter. More specifically, algorithms of running time of the form $f(k)p(n)$ have been studied for VERTEX COVER, where $p(n)$ is a low-degree polynomial of the number $n = |G|$ of vertices in G and $f(k)$ is a function independent of n.

There has been an impressive sequence of improved algorithms for the VERTEX COVER problem. A number of new techniques have been developed during this research, including kernelization, folding, and refined branch-and-search. In particular, the *kernelization* method is the study of polynomial time algorithms that can significantly reduce the instance size for VERTEX COVER. The following are some concepts related to the kernelization method:

Definition 1 Two instances (G, k) and (G', k') of VERTEX COVER are *equivalent* if the graph G has a vertex cover of size $\leq k$ if and only if the graph G' has a vertex cover of size $\leq k'$.

Definition 2 A *kernelization algorithm* for the VERTEX COVER problem takes an instance (G, k) of VERTEX COVER as input and produces an equivalent instance (G', k') for the problem, such that $|G'| \leq |G|$ and $k' \leq k$.

The kernelization method has been used extensively in conjunction with other techniques in the development of algorithms for the VERTEX COVER problem. Two major issues in the study of kernelization method are (1) effective reductions of instance size; and (2) the efficiency of kernelization algorithms.

Key Results

A number of kernelization techniques are discussed and studied in the current paper.

Preprocessing Based on Vertex Degrees

Let (G, k) be an instance of VERTEX COVER. Let v be a vertex of degree larger than k in G. If a vertex cover C does not include v, then C must contain all neighbors of v, which implies that C contains more than k vertices. Therefore, in order to find a vertex cover of no more than k vertices, one must include v in the vertex cover, and recursively look for a vertex cover of $k - 1$ vertices in the remaining graph.

The following fact was observed on vertices of degree less than 3.

Theorem 1 *There is a linear time kernelization algorithm that on each instance (G, k) of vertex cover, where the graph G contains a vertex of degree less than 3, produces an equivalent instance* (G', k') *such that* $|G'| < |G|$ *and/or* $k < k'$.

Therefore, vertices of high degree (i.e., degree $> k$) and low degree (i.e., degree < 3) can always be handled efficiently before any more time-consuming process.

Nemhauser-Trotter Theorem

Let G be a graph with vertices $v_1, v_2, \ldots, v_n$. Consider the following integer programming problem:

$$\begin{aligned} \text{(IP)Minimize} \quad & x_1 + x_2 + \cdots + x_n \\ \text{Subject to} \quad & x_i + x_j \geq 1 \\ & \text{for each edge } [v_i, v_j] \text{ in G} \\ & x_i \in \{0, 1\}, \quad 1 \leq i \leq n \end{aligned}$$

It is easy to see that there is a one-to-one correspondence between the set of feasible solutions to (IP) and the set of vertex covers of the graph G. A natural LP-relaxation (LP) of the problem (IP) is to replace the restrictions $x_i \in \{0, 1\}$ with $x_i \geq 0$ for all i. Note that the resulting linear programming problem (LP) now can be solved in polynomial time.

Let $\sigma = \{x_1^0, \ldots, x_n^0\}$ be an optimal solution to the linear programming problem (LP). The vertices in the graph G can be partitioned into three disjoint parts according to σ:

$$\begin{aligned} I_0 &= \{v_i \mid x_i^0 < 0.5\}, \\ C_0 &= \{v_i \mid x_i^0 > 0.5\}, \text{ and} \\ V_0 &= \{v_i \mid x_i^0 = 0.5\} \end{aligned}$$

The following nice property of the above vertex partition of the graph G was first observed by Nemhauser and Trotter [5].

Theorem 2 ***(Nemhauser-Trotter)*** *Let* $G[V_0]$ *be the subgraph of G induced by the vertex set* V_0. *Then (1) every vertex cover of* $G[V_0]$ *contains at least* $|V_0|/2$ *vertices; and (2) every minimum vertex cover of* $G[V_0]$ *plus the vertex set* C_0 *makes a minimum vertex cover of the graph G.*

Let k be any integer, and let $G' = G[V_0]$ and $k' = k - |C_0|$. As first noted in [3], by Theorem 2, the instances (G, k) and (G', k') are equivalent, and $|G'| \leq 2k'$ is a necessary condition for the graph G' to have a vertex cover of size k'. This observation gives the following kernelization result.

Theorem 3 *There is a polynomial-time algorithm that for a given instance (G, k) for the vertex cover problem, constructs an equivalent instance* (G', k') *such that* $k' \leq k$ *and* $|G'| \leq 2k'$.

A Faster Nemhauser-Trotter Construction

Theorem 3 suggests a polynomial-time kernelization algorithm for VERTEX COVER. The algorithm is involved in solving the linear programming problem (LP) and partitioning the graph vertices into the sets I_0, C_0, and V_0. Solving the linear programming problem (LP) can be done in polynomial time but is kind of costly in particular when the input graph G is dense. Alternatively, Nemhauser and Trotter [5] suggested the following algorithm without using linear programming. Let G be the input graph with vertex set $\{v_1, \ldots, v_n\}$.

1. construct a bipartite graph B with vertex set $\{v_1^L, \ldots, v_n^L, v_1^R, \ldots, v_n^R\}$ such that $[v_i^L, v_j^R]$ is an edge in B if and only if $[v_i, v_j]$ is an edge in G;
2. find a minimum vertex cover C_B for B;
3. $I_0' = \{v_i \mid \text{if neither } v_i^L \text{ nor } v_i^R \text{ is in } C_B\}$;
   $C_0' = \{v_i \mid \text{if both } v_i^L \text{ and } v_i^R \text{ are in } C_B\}$;
   $V_0' = \{v_i \mid \text{if exactly one of } v_i^L \text{ and } v_i^R \text{ is in } C_B\}$

It can be proved [5] (see also [2]) that Theorem 2 still holds true when the sets C_0 and V_0 in the theorem are replaced by the sets C_0' and V_0', respectively, constructed in the above algorithm.

The advantage of this approach is that the sets C_0' and V_0' can be constructed in time $O(m\sqrt{n})$ because the minimum vertex cover C_B for the bipartite graph B can be constructed via a maximum matching of B, which can be constructed in time $O(m\sqrt{n})$ using Dinic's maximum flow algorithm, which is in general faster than solving the linear programming problem (LP).

Crown Reduction

For a set S of vertices in a graph G, denote by $N(S)$ the set of vertices that are not in S but adjacent to some vertices in S. A *crown* in a graph G is a pair (I, H) of subsets of vertices in G satisfying the following conditions: (1) $I \neq \emptyset$ is an independent set, and $H = N(I)$; and (2) there is a matching M on the edges connecting I and H such that all vertices in H are matched in M.

It is quite easy to see that for a given crown (I, H), there is a minimum vertex cover that includes all vertices in H and excludes all vertices in I. Let G' be the graph obtained by removing all vertices in I and H from G. Then, the instances (G, k) and (G', k') are equivalent, where $k' = k - |H|$. Therefore, identification of crowns in a graph provides an effective way for kernelization.

Let G be the input graph. The following algorithm is proposed.

1. construct a maximal matching M_1 in G; let O be the set of vertices unmatched in M_1;
2. construct a maximum matching M_2 of the edges between O and $N(O)$; $i = 0$; let I_0 be the set of vertices in O that are unmatched in M_2;
3. repeat until $I_i = I_{i-1}$ $\{H_i = N(I_i)$; $I_{i+1} = I_i \cup N_{M_2}(H_i)$;$i = i + 1$; $\}$; (where $N_{M_2}(H_i)$ is the set of vertices in O that match the vertices in H_i in the matching M_2)
4. $I = I_i$; $H = N(I_i)$; output (I, H).

Theorem 4 *(1) if the set I_0 is not empty, then the above algorithm constructs a crown (I, H); (2) if both $|M_1|$ and $|M_2|$ are bounded by k, and $I_0 = \emptyset$, then the graph G has at most 3k vertices.*

According to Theorem 4, the above algorithm on an instance (G, k) of VERTEX COVER either (1) finds a matching of size larger than k – which implies that there is no vertex cover of k vertices in the graph G; or (2) constructs a crown (I, H) – which will reduce the size of the instance; or (3) in case neither of (1) and (2) holds true, concludes that the graph G contains at most $3k$ vertices. Therefore, repeatedly applying the algorithm either derives a direct solution to the given instance, or constructs an equivalent instance (G', k') with $k' \leq k$ and $|G'| \leq 3k'$.

Applications

The research of the current paper was directly motivated by authors' research in bioinformatics. It is shown that for many computational biological problems, such as the construction of phylogenetic trees, phenotype identification, and analysis of microarray data, preprocessing based on the kernelization techniques has been very effective.

Experimental Results

Experimental results are given for handling graphs obtained from the study of phylogenetic trees based on protein domains, and from the analysis of microarray data. The results show that in most cases the best way to kernelize is to start handling vertices of high and low degrees (i.e., vertices of degree larger than k or smaller than 3) before attempting any of the other kernelization techniques. Sometimes, kernelization based on Nemhauser-Trotter Theorem can solve the problem without any further branching. It is also observed that sometimes particularly on dense graphs, kernelization techniques based on Nemhauser-Trotter Theorem are kind of time-consuming but do not reduce the instance size by much. On the other hand, the techniques based on high-degree vertices and crown reduction seem to work better.

Data Sets

The experiments were performed on graphs obtained based on data from NCBI and SWISS-PROT, well known open-source repositories of biological data.

Cross-References

- ▸ Data Reduction for Domination in Graphs
- ▸ Local Approximation of Covering and Packing Problems
- ▸ Vertex Cover Search Trees

Recommended Reading

1. Abu-Khzam F, Collins R, Fellows M, Langston M, Suters W, Symons C (2004) Kernelization algorithms for the vertex cover problem: theory and experiments. In: Proceedings of the workshop on algorithm engineering and experiments (ALENEX), pp 62–69
2. Bar-Yehuda R, Even S (1985) A local-ratio theorem for approximating the weighted vertex cover problem. Ann Discret Math 25:27–45
3. Chen J, Kanj IA, Jia W (2001) Vertex cover: further observations and further improvements. J Algorithm 41:280–301
4. Garey M, Johnson D (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco
5. Nemhauser GL, Trotter LE (1975) Vertex packing: structural properties and algorithms. Math Program 8:232–248

Vertex Cover Search Trees

Jianer Chen
Department of Computer Science, Texas A&M University, College Station, TX, USA

Keywords

Branch and bound; Branch and search

Years and Authors of Summarized Original Work

2001; Chen, Kanj, Jia

Problem Definition

The VERTEX COVER problem is one of the six "basic" NP-complete problems according to Garey and Johnson [7]. Therefore, the problem cannot be solved in polynomial time unless P = NP. However, the NP-completeness of the problem does not obviate the need for solving it because of its fundamental importance and wide applications.

One approach is to develop *parameterized algorithms* for the problem, with the computational complexity of the algorithms being measured in terms of both input size and a parameter value. This approach was initiated based on the observation that in many applications, the instances of the problem are associated with a small parameter. Therefore, by taking the advantages of the small parameters, one may be able to solve this NP-complete problem effectively and practically.

The problem is formally defined as follows. Let G be an (undirected) graph. A subset C of vertices in G is a *vertex cover* for G if every edge in G has at least one end in C. An instance of the (parameterized) VERTEX COVER problem consists of a pair (G, k), where G is a graph and k is an integer (the parameter), which is to determine whether the graph G has a vertex cover of k vertices. The goal is to develop parameterized algorithms of running time $O(f(k)p(n))$ for the VERTEX COVER problem, where $p(n)$ is a lower-degree polynomial of the input size n, and $f(k)$ is the non-polynomial part that is a function of the parameter k but independent of the input size n. It would be expected that the non-polynomial function $f(k)$ is as small as possible. Such an algorithm would become "practically effective" when the parameter value k is small. It should be pointed out that unless an unlikely consequence occurs in complexity theory, the function $f(k)$ is at least an exponential function of the parameter k [8].

Key Results

A number of techniques have been proposed in the development of parameterized algorithms for the VERTEX COVER problem.

Kernelization

Suppose (G, k) is an instance for the VERTEX COVER problem, where G is a graph and k is the parameter. The *kernelization* operation applies a polynomial time preprocessing on the instance (G, k) to construct another instance (G', k'), where G' is a smaller graph (the *kernel*) and $k' \leq k$, such that G' has a vertex cover of k' vertices if and only if G has a vertex cover of k vertices. Based on a classical result by Nemhauser and Trotter [9], the following kernelization result was derived.

Theorem 1 *There is an algorithm of running time $O(kn + k^3)$ that for a given instance (G, k) for the* VERTEX COVER *problem, constructs another instance (G′, k′) for the problem, where the graph G′ contains at most 2k′ vertices andk′ ≤ k, such that the graph G has a vertex cover of k vertices if and only if the graph G′ has a vertex cover of k′ vertices.*

Therefore, kernelization provides an efficient preprocessing for the VERTEX COVER problem, which allows one to concentrate on graphs of small size (i.e., graphs whose size is only related to k).

Folding

Suppose v is a degree-2 vertex in a graph G with two neighbors u and w such that u and w are not adjacent to each other. Construct a new graph G' as follows: remove the vertices v, u, and w and introduce a new vertex v_0 that is adjacent to all remaining neighbors of the vertices u and w in G. The graph G' is said being obtained from the graph G by *folding the vertex* v. The following result was derived.

Theorem 2 *Let G′ be a graph obtained by folding a degree-2 vertex v in a graph G, where the two neighbors of v are not adjacent to each other. Then the graph G has a vertex cover of k vertices if and only if the graph G′ has a vertex cover of $k - 1$ vertices.*

An folding operation allows one to decrease the value of the parameter k without branching. Therefore, folding operations are regarded as very efficient in the development of exponential time algorithms for the VERTEX COVER problem. Recently, the folding operation has be generalized to apply to a set of more than one vertex in a graph [6].

Branch and Search

A main technique is the *branch and search* method that has been extensively used in the development of algorithms for the VERTEX COVER problem (and for many other NP-hard problems). The method can be described as follows. Let (G, k) be an instance of the VERTEX COVER problem. Suppose that somehow a collection $\{C_1, \ldots, C_b\}$ of vertex subsets in the graph G is identified, where for each i, the subset C_i has c_i vertices, such that if the graph G contains a vertex cover of k vertices, then at least for one C_i of the vertex subsets in the collection, there is a vertex cover of k vertices for G that contains all vertices in C_i. Then a collection of (smaller) instances (G_i, k_i) can be constructed, where $1 \leq i \leq b, k_i = k - c_i$, and G_i is obtained from G by removing all vertices in C_i. Note that the original graph G has a vertex cover of k vertices if and only if for one (G_i, k_i) of the smaller instances the graph G_i has a vertex cover of k_i vertices. Therefore, now the process can be branched into b sub-processes, each on a smaller instance (G_i, k_i) recursively searches for a vertex cover of k_i vertices in the graph G_i.

Let $T(k)$ be the number of leaves in the search tree for the above branch and search process on the instance (G, k), then the above branch operation gives the following recurrence relation:

$$T(k) = T(k - c_1) + T(k - c_2) + \cdots + T(k - c_b)$$

To solve this recurrence relation, let $T(k) = x^k$ so that the above recurrence relation becomes

$$x^k = x^{k-c_1} + x^{k-c_2} + \cdots + x^{k-c_b}$$

It can be proved [3] that the above polynomial equation has a unique root x_0 larger than 1. From this, one gets $T(k) = x_0^k$, which, up to a polynomial factor, gives an upper bound on the running time of the branch and search process on the instance (G, k).

The simplest case is that a vertex v of degree $d > 0$ in the graph G is picked. Let $w_1, \ldots, w_d$ be the neighbors of v. Then either v is contained in a vertex cover C of k vertices, or, if v is not contained in C, then all neighbors $w_1, \ldots, w_d$ of v must be contained in C. Therefore, one obtains a collection of two subsets $C_1 = \{v\}$ and $C_2 = \{w_1, \ldots, w_d\}$, on which the branch and search process can be applied.

The efficiency of a branch and search operation depends on how effectively one can identify the collection of the vertex subsets. Intuitively, the larger the sizes of the vertex subsets, the more efficient is the operation. Much effort has been made in the development of VERTEX COVER algorithms to achieve larger vertex subsets. Improvements on the size of the vertex subsets have been involved with very complicated and tedious analysis and enumerations of combinatorial structures of graphs. The current paper [3] achieved a collection of two subsets C_1 and C_2 of sizes $c_1 = 1$ and $c_2 = 6$, respectively, and other collections of vertex subsets that are at least as good as this (the techniques of kernelization and vertex folding played important roles in achieving these collections). This gives the following algorithm for the VERTEX COVER problem.

Theorem 3 *The* VERTEX COVER *problem can be solved in time* $O(kn + 1.2852^k)$.

Very recently, a further improvement over Theorem 3 has been achieved that gives an algorithm of running time $O(kn + 1.2738^k)$ for the VERTEX COVER problem [4].

Applications

The study of parameterized algorithms for the VERTEX COVER problem was motivated by ETH Zürich's DARWIN project in computational biology and computational biochemistry (see, e.g., [10, 11]). A number of computational problems in the project, such as multiple sequence alignments [10] and biological conflict resolving [11], can be formulated into the VERTEX COVER problem in which the parameter value is in general not larger than 100. Therefore, an algorithm of running time $O(kn + 1.2852^k)$ for the problem becomes very effective and practical in solving these problems.

The parameterized algorithm given in Theorem 3 has also induced a faster algorithm for another important NP-hard problem, the MAXIMUM INDEPENDENT SET problem on sparse graphs [3].

Open Problems

The main open problem in this line of research is how far one can go along this direction. More specifically, how small the constant $c > 1$ can be for the VERTEX COVER problem to have an algorithm of running time $O(c^k n^{O(1)})$? With further more careful analysis on graph combinatorial structures, it seems possible to slightly improve the current best upper bound [4] for the problem. Some new techniques developed more recently [6] also seem very promising to improve the upper bound. On the other hand, it is known that the constant c cannot be arbitrarily close to 1 unless certain unlikely consequence occurs in complexity theory [8].

Experimental Results

A number of research groups have implemented some of the ideas of the algorithm in Theorem 3 or its variations, including the Parallel Bioinformatics project in Carleton University [2], the High Performance Computing project in University of Tennessee [1], and the DARWIN project in ETH Zürich [10, 11]. As reported in [5], these implementations showed that this algorithm and the related techniques are "quite practical" for the VERTEX COVER problem with parameter value k up to around 400.

Cross-References

- Data Reduction for Domination in Graphs
- Exact Algorithms for k SAT Based on Local Search
- Local Approximation of Covering and Packing Problems
- Vertex Cover Kernelization

Recommended Reading

1. Abu-Khzam F, Collins R, Fellows M, Langston M, Suters W, Symons C (2004) Kernelization algorithms for the vertex cover problem: theory and experiments. In: Proceedings of the workshop on algorithm engineering and experiments (NLENEX), pp 62–69
2. Cheetham J, Dehne F, Rau-Chaplin A, Stege U, Taillon P (2003) Solving large FPT problems on coarse grained parallel machines. J Comput Syst Sci 67:691–706
3. Chen J, Kanj IA, Jia W (2001) Vertex cover: further observations and further improvements. J Algorithm 41:280–301
4. Chen J, Kanj IA, Xia G (2006) Improved parameterized upper bounds for vertex cover. In: MFCS 2006. Lecture notes in computer science, vol 4162. Springer, Berlin, pp 238–249
5. Fellows M (2001) Parameterized complexity: the main ideas and some research frontiers. In: ISAAC 2001. Lecture notes in computer science, vol 2223. Springer, Berlin, pp 291–307
6. Fomin F, Grandoni F, Kratsch D (2006) Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm. In: Proceedings of the 17th annual ACM-SIAM symposium on discrete algorithms (SODA 2006), pp 18–25
7. Garey M, Johnson D (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco
8. Impagliazzo R, Paturi R (2001) Which problems have strongly exponential complexity? J Comput Syst Sci 63:512–530
9. Nemhauser GL, Trotter LE (1975) Vertex packing: structural properties and algorithms. Math Program 8:232–248
10. Roth-Korostensky C (2000) Algorithms for building multiple sequence alignments and evolutionary trees. PhD thesis, ETH Zürich, Institute of Scientific Computing
11. Stege U (2000) Resolving conflicts from problems in computational biology. PhD thesis, ETH Zürich, Institute of Scientific Computing

Visualization Techniques for Algorithm Engineering

Camil Demetrescu[1,2] and Giuseppe F. Italiano[1,2]
[1]Department of Computer and Systems Science, University of Rome, Rome, Italy
[2]Department of Information and Computer Systems, University of Rome, Rome, Italy

Keywords

Using visualization in the empirical assessment of algorithms

Years and Authors of Summarized Original Work

2002; Demetrescu, Finocchi, Italiano, Näher

Problem Definition

The whole process of designing, analyzing, implementing, tuning, debugging and experimentally evaluating algorithms can be referred to as *Algorithm Engineering*. Algorithm Engineering views algorithmics also as an engineering discipline rather than a purely mathematical discipline. Implementing algorithms and engineering algorithmic codes is a key step for the transfer of algorithmic technology, which often requires a high-level of expertise, to different and broader communities, and for its effective deployment in industry and real applications.

Experiments can help measure practical indicators, such as implementation constant factors, real-life bottlenecks, locality of references, cache effects and communication complexity, that may be extremely difficult to predict theoretically. Unfortunately, as in any empirical science, it may be sometimes difficult to draw general conclusions about algorithms from experiments. To this aim, some researchers have proposed accurate and comprehensive guidelines on different

aspects of the empirical evaluation of algorithms maturated from their own experience in the field (see, for example [1, 15, 16, 20]). The interested reader may find in [18] an annotated bibliography of experimental algorithmics sources addressing methodology, tools and techniques.

The process of implementing, debugging, testing, engineering and experimentally analyzing algorithmic codes is a complex and delicate task, fraught with many difficulties and pitfalls. In this context, traditional low-level textual debuggers or industrial-strength development environments can be of little help for algorithm engineers, who are mainly interested in high-level algorithmic ideas rather than in the language and platform-dependent details of actual implementations. Algorithm visualization environments provide tools for abstracting irrelevant program details and for conveying into still or animated images the high-level algorithmic behavior of a piece of software.

Among the tools useful in algorithm engineering, visualization systems exploit interactive graphics to enhance the development, presentation, and understanding of computer programs [27]. Thanks to the capability of conveying a large amount of information in a compact form that is easily perceivable by a human observer, visualization systems can help developers gain insight about algorithms, test implementation weaknesses, and tune suitable heuristics for improving the practical performances of algorithmic codes. Some examples of this kind of usage are described in [12].

Key Results

Systems for algorithm visualization have matured significantly since the rise of modern computer graphic interfaces and dozens of algorithm visualization systems have been developed in the last two decades [2, 3, 4, 5, 6, 8, 9, 10, 13, 17, 25, 26, 29]. For a comprehensive survey the interested reader can be referred to [11, 27] and to the references therein. The remainder of this entry discusses the features of algorithm visualization systems that appear to be most appealing for their deployment in algorithm engineering.

Critical Issues

From the viewpoint of the algorithm developer, it is desirable to rely on systems that offer visualizations at a *high level of abstraction*. Namely, one would be more interested in visualizing the behavior of a complex data structure, such as a graph, than in obtaining a particular value of a given pointer.

Fast prototyping of visualizations is another fundamental issue: algorithm designers should be allowed to create visualization from the source code at hand with little effort and without heavy modifications. At this aim, *reusability* of visualization code could be of substantial help in speeding up the time required to produce a running animation.

One of the most important aspects of algorithm engineering is the development of *libraries*. It is thus quite natural to try to interface visualization tools to algorithmic software libraries: libraries should offer default visualizations of algorithms and data structures that can be refined and customized by developers for specific purposes.

Software visualization tools should be able to animate *not just "toy programs"*, but significantly complex algorithmic codes, and to test their behavior on large data sets. Unfortunately, even those systems well suited for large information spaces often lack advanced navigation techniques and methods to alleviate the screen bottleneck. Finding a solution to this kind of limitations is nowadays a challenge.

Advanced debuggers take little advantage of sophisticated graphical displays, even in commercial software development environments. Nevertheless, software visualization tools may be very beneficial in addressing problems such as finding memory leaks, understanding anomalous program behavior, and studying performance. In particular, environments that provide interpreted execution may more easily integrate advanced facilities in support to *debugging and performance monitoring*, and

many recent systems attempt at exploring this research direction.

Techniques

One crucial aspect in visualizing the dynamic behavior of a running program is the way it is conveyed into graphic abstractions. There are two main approaches to bind visualizations to code: the event-driven and the state-mapping approach.

Event-Driven Visualization

A natural approach to algorithm animation consists of annotating the algorithmic code with calls to visualization routines. The first step consists of identifying the relevant actions performed by the algorithm that are interesting for visualization purposes. Such relevant actions are usually referred to as *interesting events*. As an example, in a sorting algorithm the swap of two items can be considered an interesting event. The second step consists of associating each interesting event with a modification of a graphical scene. Animation scenes can be specified by setting up suitable visualization procedures that drive the graphic system according to the actual parameters generated by the particular event. Alternatively, these visualization procedures may simply log the events in a file for a *post-mortem* visualization. The calls to the visualization routines are usually obtained by annotating the original algorithmic code at the points where the interesting events take place. This can be done either by hand or by means of specialized editors. Examples of toolkits based on the event-driven approach are Polka [28] and *GeoWin*, a `C++` data type that can be easily interfaced with algorithmic software libraries of great importance in algorithm engineering such as CGAL [14] and LEDA [19].

State Mapping Visualization

Algorithm visualization systems based on state mapping rely on the assumption that observing how the variables change provides clues to the actions performed by the algorithm. The focus is on capturing and monitoring the data modifications rather than on processing the interesting events issued by the annotated algorithmic code. For this reason they are also referred to as "data driven" visualization systems. Conventional debuggers can be viewed as data driven systems, since they provide direct feedback of variable modifications. The main advantage of this approach over the event-driven technique is that a much greater ignorance of the code is allowed: indeed, only the interpretation of the variables has to be known to animate a program. On the other hand, focusing only on data modification may sometimes limit customization possibilities making it difficult to realize animations that would be natural to express with interesting events. Examples of tools based on the state mapping approach are Pavane [23, 25], which marked the first paradigm shift in algorithm visualization since the introduction of interesting events, and Leonardo [10] an integrated environment for developing, visualizing, and executing C programs.

A comprehensive discussion of other techniques used in algorithm visualization appears in [7, 21, 22, 24, 27].

Applications

There are several applications of visualization in algorithm engineering, such as testing and debugging of algorithm implementations, visual inspection of complex data structures, identification of performance bottlenecks, and code optimization. Some examples of uses of visualization in algorithm engineering are described in [12].

Open Problems

There are many challenges that the area of algorithm visualization is currently facing. First of all, the real power of an algorithm visualization system should be in the hands of the final user, possibly inexperienced, rather than of a professional programmer or of the developer of the tool. For instance, instructors may greatly benefit from fast and easy methods for tailoring animations to their specific educational needs, while they might be discouraged from using systems that are difficult to install or heavily dependent on particular software/hardware platforms. In addition to

being easy to use, a software visualization tool should be able to animate significantly complex algorithmic codes without requiring a lot of effort. This seems particularly important for future development of visual debuggers. Finally, visualizing the execution of algorithms on large data sets seems worthy of further investigation. Currently, even systems designed for large information spaces often lack advanced navigation techniques and methods to alleviate the screen bottleneck, such as changes of resolution and scale, selectivity, and elision of information.

Cross-References

▸ Experimental Methods for Algorithm Analysis

Recommended Reading

1. Anderson RJ (2002) The role of experiment in the theory of algorithms. In: Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges. DIMACS series in discrete mathematics and theoretical computer science, vol 59. American Mathematical Society, Providence, pp 191–195
2. Baker J, Cruz I, Liotta G, Tamassia R (1996) Animating geometric algorithms over the web. In: Proceedings of the 12th annual ACM symposium on computational geometry. Philadelphia, 24–26 May 1996, pp C3–C4
3. Baker J, Cruz I, Liotta G, Tamassia R (1996) The Mocha algorithm animation system. In: Proceedings of the 1996 ACM workshop on advanced visual interfaces, Gubbio, 27–29 May 1996, pp 248–250
4. Baker J, Cruz I, Liotta G, Tamassia R (1996) A new model for algorithm animation over the WWW. ACM Comput Surv 27:568–572
5. Baker R, Boilen M, Goodrich M, Tamassia R, Stibel B (1999) Testers and visualizers for teaching data structures. In: Proceeding of the 13th SIGCSE technical symposium on computer science education, New Orleans, 24–28 Mar 1999, pp 261–265
6. Brown M (1988) Algorithm animation. MIT Press, Cambridge
7. Brown M (1988) Perspectives on algorithm animation. In: Proceedings of the ACM SIGCHI'88 conference on human factors in computing systems. Washington, DC, 15–19 May 1988, pp 33–38
8. Brown M (1991) Zeus: a system for algorithm animation and multi-view editing. In: Proceedings of the 7th IEEE workshop on visual languages, Kobe, 8–11 Oct 1991, pp 4–9
9. Cattaneo G, Ferraro U, Italiano GF, Scarano V (2002) Cooperative algorithm and data types animation over the net. J Vis Lang Comp 13(4):391
10. Crescenzi P, Demetrescu C, Finocchi I, Petreschi R (2008) Reversible execution and visualization of programs with LEONARDO. J Vis Lang Comp 11:125–150, Leonardo is available at: http://www.dis.uniroma1.it/~demetres/Leonardo/. Accessed 15 Jan 2008
11. Demetrescu C (2001) Fully dynamic algorithms for path problems on directed graphs. PhD thesis, Department of Computer and Systems Science, University of Rome "La Sapienza"
12. Demetrescu C, Finocchi I, Italiano GF, Näher S (2002) Visualization in algorithm engineering: tools and techniques. In: Experimental algorithm design to robust and efficient software. Lecture notes in computer science, vol 2547, Springer, Berlin, pp 24–50
13. Demetrescu C, Finocchi I, Liotta G (2000) Visualizing algorithms over the web with the publication-driven approach. In: Proceedings of the 4th workshop on algorithm engineering (WAE'00), Saarbrücken, 5–8 Sept 2000
14. Fabri A, Giezeman G, Kettner L, Schirra S, Schönherr S (1996) The cgal kernel: a basis for geometric computation. In: Applied computational geometry: towards geometric engineering proceedings (WACG'96), Philadelphia, 27–28 May 1996, pp 191–202
15. Goldberg A (1999) Selecting problems for algorithm evaluation. In: Proceedings of the 3rd workshop on algorithm engineering (WAE'99), London, 19–21 July 1999. Lecture notes in computer science, vol 1668, pp 1–11
16. Johnson D (2002) A theoretician's guide to the experimental analysis of algorithms. In: Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS series in discrete mathematics and theoretical computer science, vol 59. American Mathematical Society, Providence, 215–250
17. Malony A, Reed D (1989) Visualizing parallel computer system performance. In: Simmons M, Koskela R, Bucher I (eds) Instrumentation for future parallel computing systems. ACM Press, New York, pp 59–90
18. McGeoch C (2002) A bibliography of algorithm experimentation. In: Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges. DIMACS series in discrete mathematics and theoretical computer science, vol 59. American Mathematical Society, Providence, pp 251–254
19. Mehlhorn K, Naher S (1999) LEDA: a platform of combinatorial and geometric computing. Cambridge University Press, Cambridge. ISBN 0-521-56329-1
20. Moret B (2002) Towards a discipline of experimental algorithmics. In: Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges. dimacs series in discrete mathematics and theoretical computer Science,

vol 59. American Mathematical Society, Providence, pp 197–214
21. Myers B (1990) Taxonomies of visual programming and program visualization. J Vis Lang Comp 1:97–123
22. Price B, Baecker R, Small I (1993) A principled taxonomy of software visualization. J Vis Lang Comp 4:211–266
23. Roman G, Cox K (1989) A declarative approach to visualizing concurrent computations. Computer 22:25–36
24. Roman G, Cox K (1993) A taxonomy of program visualization systems. Computer 26:11–24
25. Roman G, Cox K, Wilcox C, Plun J (1992) PAVANE: a system for declarative visualization of concurrent computations. J Vis Lang Comp 3:161–193
26. Stasko J (1992) Animating algorithms with X-TANGO. SIGACT News 23:67–71
27. Stasko J, Domingue J, Brown M, Price B (1997) Software visualization: programming as a multimedia experience. MIT Press, Cambridge
28. Stasko J, Kraemer E (1993) A methodology for building application- specific visualizations of parallel programs. J Parallel Distrib Comp 18:258–264
29. Tal A, Dobkin D (1995) Visualization of geometric algorithms. IEEE Trans Vis Comp Graph 1:194–204

Voltage Scheduling

Ming Min Li
Computer Science and Technology, Tsinghua University, Beijing, China

Keywords

Dynamic speed scaling

Years and Authors of Summarized Original Work

2005; Li, Yao

Problem Definition

This problem is concerned with scheduling jobs with as little energy as possible by adjusting the processor speed wisely. This problem is motivated by *dynamic voltage scaling (DVS)* (or *speed scaling*) technique, which enables a processor to operate at a range of voltages and frequencies. Since energy consumption is at least a quadratic function of the supply voltage (hence CPU frequency/speed), it saves energy to execute jobs as slowly as possible while still satisfying all timing constraints. The associated scheduling problem is referred to as min-energy DVS scheduling. Previous work showed that the min-energy DVS schedule can be computed in cubic time. The work of Li and Yao [7] considers the discrete model where the processor can only choose its speed from a finite speed set. This work designs an $O(dn \log n)$ two-phase algorithm to compute the min-energy DVS schedule for the discrete model (d represents the number of speeds) and also proves a lower bound of $\Omega(n \log n)$ for the computation complexity.

Notations and Definitions

In the variable voltage scheduling model, there are two important sets:

1. Set J (job set) consists of n jobs: $j_1, j_2, \ldots j_n$. Each job j_k has three parameters as its information: a_k representing the arrival time of j_k, b_k representing the deadline of j_k, and R_k representing the total CPU cycles required by j_k. The parameters satisfy $0 \le a_k < b_k \le 1$.
2. Set SD (speed set) consists of the possible speeds that can be used by the processor. According to the property of SD, the scheduling model is divided into the following two categories:

   Continuous model: The set SD is the set of positive real numbers.
   Discrete model: The set SD consists of d positive values: $s_1 > s_2 > \cdots > s_d$.

A schedule S consists of the following two functions: $s(t)$ which specifies the processor speed at time t and job(t) which specifies the job executed at time t. Both functions are piecewise constant with finitely many discontinuities.

A feasible schedule must give each job its required number of cycles between arrival time and deadline, therefore satisfying the property $\int_{a_k}^{b_k} s(t)\delta(k, \text{job}(t))\text{d}t = R_k$, where $\delta(i, j) = 1$ if $i = j$ and $\delta(i, j) = 0$ if otherwise.

The EDF principle defines an ordering on the jobs according to their deadlines. At any time t, among jobs j_k that are available for execution, that is, j_k satisfying $t \in [a_k, b_k)$ and j_k not yet finished by t, it is the job with minimum b_k that will be executed during $[t, t + \epsilon]$.

The power P, or energy consumed per unit of time, is a convex function of the processor speed. The energy consumption of a schedule $S = (s(t), \text{job}(t))$ is defined as $E(S) = \int_0^1 P(s(t))\text{d}t$.

A schedule is called an optimal schedule if its energy consumption is the minimum possible among all the feasible schedules. Note that for the continuous model, the optimal schedule uses the same speed for the same job.

The work of Li and Yao considers the problem of computing an optimal schedule for the discrete model under the following assumptions.

Assumptions

1. **Single processor:** At any time t, only one job can be executed.
2. **Preemptive:** Any job can be interrupted during its execution.
3. **Non-precedence:** There is no precedence relationship between any pair of jobs.
4. **Offline:** The processor knows the information of all the jobs at time 0.

This problem is called min-energy discrete dynamic voltage scaling (MEDDVS).

Problem 1 (MEDDVS$_{J,SD}$)

INPUT: Integer n, set $J = \{j_1, j_2, \ldots, j_n\}$ and $SD = \{s_1, s_2, \ldots, s_d\} \cdot j_k = \{a_k, b_k, R_k\}$.
OUTPUT: Feasible schedule $S = (s(t), \text{job}(t))$ that minimizes $E(S)$.

Kwon and Kim [6] proved that the optimal schedule for the discrete model can be obtained by first calculating the optimal schedule for the continuous model and then individually adjusting the speed of each job appropriately to adjacent levels in set SD. The time complexity is $O(n^3)$.

Key Results

The work of Li and Yao finds a direct approach for solving the MEDDVS problem without first computing the optimal schedule for the continuous model.

Definition 1 An s-schedule for J is a schedule which conforms to the EDF principle and uses constant speed s in executing any job of J.

Lemma 1 *The s-schedule for J can be computed in $O(n \log n)$ time.*

Definition 2 Given a job set J and any speed s, let $J^{\geq s}$ and $J^{<s}$ denote the subset of J consisting of jobs whose executing speeds are $\geq s$ and $< s$, respectively, in the optimal schedule for J in the continuous model. The partition $J^{\geq s}, J^{<s}$ is referred to as the s-partition of J.

By extracting information from the s-schedule, a partition algorithm is designed to prove the following lemma:

Lemma 2 *The s-partition of J can be computed in $O(n \log n)$ time.*

By applying s-partition to J using all the d speeds in SD consecutively, one can obtain d subsets $J_1, J_2, \ldots, J_d$ of J where jobs in the same subset J_i use the same two speeds s_i and s_{i+1} in the optimal schedule for the Discrete Model ($s_{d+1} = 0$).

Lemma 3 *Optimal schedule for job set J_i using speeds s_i and s_{i+1} can be computed in $O(n \log n)$ time.*

Combining the above three lemmas together, the main theorem follows:

Theorem 1 *The min-energy discrete DVS schedule can be computed in $O(dn \log n)$ time.*

A lower bound to compute the optimal schedule for the *discrete model* under the algebraic decision tree model is also shown by Li and Yao.

Theorem 2 *Any deterministic algorithm for computing min-energy discrete DVS schedule with* $d \geq 2$ *voltage levels requires* $\Omega(n \log n)$ *time for n jobs.*

Applications

Currently, dynamic voltage scaling technique is being used by the world's largest chip companies, e.g., Intel's SpeedStep technology and AMD's PowerNow technology. Although the scheduling algorithms being used are mostly online algorithms, offline algorithms can still find their places in real applications. Furthermore, the techniques developed in the work of Li and Yao for the computation of optimal schedules may have potential applications in other areas.

People also study energy-efficient scheduling problems for other kinds of job sets. Yun and Kim [10] proved that it is NP-hard to compute the optimal schedule for jobs with priorities and gave an FPTAS for that problem. Aydin et al. [1] considered energy-efficient scheduling for real-time periodic jobs and gave an $O(n^2 \log n)$ scheduling algorithm. Chen et al. [4] studied the weakly discrete model for non-preemptive jobs where speed is not allowed to change during the execution of one job. They proved the NP-hardness to compute the optimal schedule.

Another important application for this work is to help investigating scheduling model with more hardware restrictions (Burd and Brodersen [3] explained various design issues that may happen in dynamic voltage scaling). Besides the single-processor model, people are also interested in the multiprocessor model [11].

Open Problems

A number of problems related to the work of Li and Yao remain open. In the discrete model, Li and Yao's algorithm for computing the optimal schedule requires time $O(dn \log n)$. There is a gap between this and the currently known lower bound $\Omega(n \log n)$. Closing this gap when considering d as a variable is an open problem.

Another open research area is the computation of the optimal schedule for the continuous model. Li, Yao, and Yao [8] obtained an $O(n^2 \log n)$ algorithm for computing the optimal schedule. The bottleneck for the log n factor is in the computation of s-schedules. Reducing the time complexity for computing s-schedules is an open problem. It is also possible to look for other methods to deal with the continuous model.

Cross-References

- List Scheduling
- Online Load Balancing of Temporary Tasks
- Parallel Algorithms for Two Processors Precedence Constraint Scheduling
- Shortest Elapsed Time First Scheduling

Recommended Reading

1. Aydin H, Melhem R, Mosse D, Alvarez PM (2001) Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In: Euromicro conference on real-time systems, Madrid. IEEE Computer Society, Washington, DC, pp 225–232
2. Bansalm N, Kimbrel T, Pruhs K (2004) Dynamic speed scaling to manage energy and temperature. In: Proceedings of the 45th annual IEEE symposium on foundations of computer science, Rome. IEEE Computer Society, Washington, DC, pp 520–529
3. Burd TD, Brodersen RW (2000) Design issues for dynamic voltage scaling. In: Proceedings of the 2000 international symposium on low power electronics and design, Rapallo. ACM, New York, pp 9–14
4. Chen JJ, Kuo TW, Lu HI (2005) Power-saving scheduling for weakly dynamic voltage scaling devices workshop on algorithms and data structures (WADS). LNCS, vol 3608. Springer, Berlin, pp 338–349
5. Irani S, Pruhs K (2005) Algorithmic problems in power management. ACM SIGACT News 36(2):63–76. New York

6. Kwon W, Kim T (2005) Optimal voltage allocation techniques for dynamically variable voltage processors. ACM Trans Embed Comput Syst 4(1):211–230. New York
7. Li M, Yao FF (2005) An efficient algorithm for computing optimal discrete voltage schedules. SIAM J Comput 35(3):658–671. Society for Industrial and Applied Mathematics, Philadelphia
8. Li M, Yao AC, Yao FF (2005) Discrete and continuous min-energy schedules for variable voltage processors. In: Proceedings of the National Academy of Sciences USA, Washington, DC, vol 103. National Academy of Science of the United States of America, Washington, DC, pp 3983–3987
9. Yao F, Demers A, Shenker S (1995) A scheduling model for reduced CPU energy. In: Proceedings of the 36th annual IEEE symposium on foundations of computer science, Milwaukee. IEEE Computer Society, Washington, DC, pp 374–382
10. Yun HS, Kim J (2003) On energy-optimal voltage scheduling for fixed-priority hard real-time systems. ACM Trans Embed Comput Syst 2:393–430. ACM, New York
11. Zhu D, Melhem R, Childers B (2001) Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. In: Proceedings of the 22nd IEEE real-time systems symposium (RTSS'01), London. IEEE Computer Society, Washington, DC, pp 84–94

Voronoi Diagrams and Delaunay Triangulations

Rolf Klein
Institute for Computer Science, University of Bonn, Bonn, Germany

Keywords

Computational geometry; Delaunay triangulation; Distance problem; Edge flip; Minimum spanning tree; Sweepline; Voronoi diagram

Years and Authors of Summarized Original Work

1975; Shamos, Hoey
1987; Fortune

Problem Definition

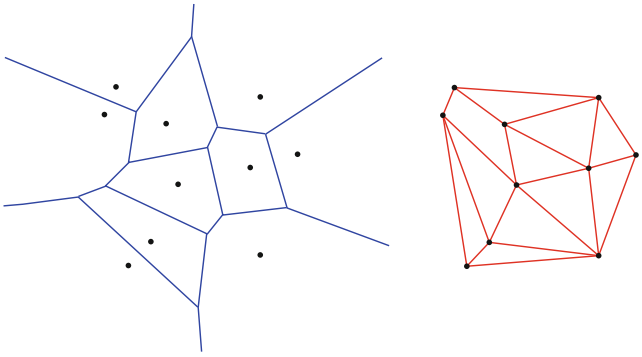
Suppose there is some set of objects p called *sites* that exert influence over their surrounding space, M. For each site p, we consider the set of all points z in M for which the influence of p is strongest.

Such decompositions have already been considered by R. Descartes [5] for the fixed stars in solar space. In mathematics and computer science, they are called *Voronoi diagrams*, honoring work by G.F. Voronoi on quadratic forms. Other sciences know them as *domains of action, Johnson-Mehl model, Thiessen polygons, Wigner-Seitz zones*, or *medial axis transform*.

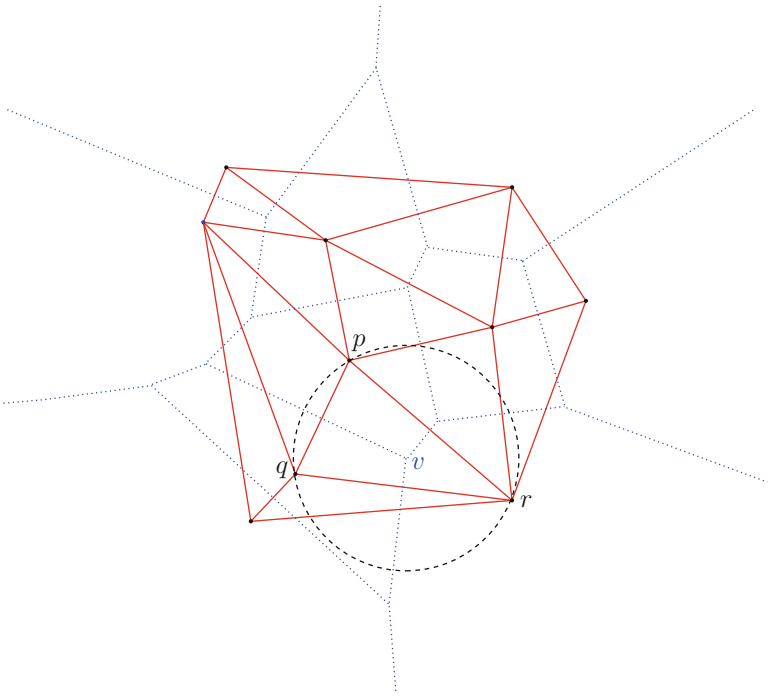
In the case most frequently studied, the space M is the real plane, the sites are n points, and influence corresponds to proximity in the Euclidean metric, so that the points most strongly influenced by site p are those for which p is the nearest neighbor among all sites. They form a convex region called the *Voronoi region* of p. The common boundary of two adjacent regions of p and q is a segment of their *bisector* $B(p,q)$, the locus of all points of equal distance to p and q. An example of 10 point sites is depicted in Fig. 1.

Let us assume that the set S of point sites is in general position, so that no three points are situated on a line, and no four on a circle. Then the Voronoi diagram $V(S)$ of S is a connected planar graph. Its vertices are those points in the plane which have three nearest neighbors in S, while the interior edge points have two. As a consequence of the Euler formula, $V(S)$ has only $O(n)$ many edges and vertices.

If we connect with line segments, those sites in S whose Voronoi regions share an edge in $V(S)$, a triangulation $D(S)$ of S results, called the *Delaunay triangulation* or *Dirichlet tessellation*; see Fig. 1. Each triangle with vertices p,q,r in S is dual to a vertex v of $V(S)$ situated on the boundary of the Voronoi regions of p,q, and r. Because p,q,r are the nearest neighbors of v in S, the circle through p,q,r centered at v contains no other point of S. Thus, $D(S)$ consists of triangles with vertices in S whose circumcircles are empty of points in S; see Fig. 2. Conversely,



Voronoi Diagrams and Delaunay Triangulations, Fig. 1
Voronoi diagram and Delaunay triangulation of 10 point sites in the Euclidean plane



Voronoi Diagrams and Delaunay Triangulations, Fig. 2
The empty circle property

each triangle with empty circumcircle occurs in $D(S)$.

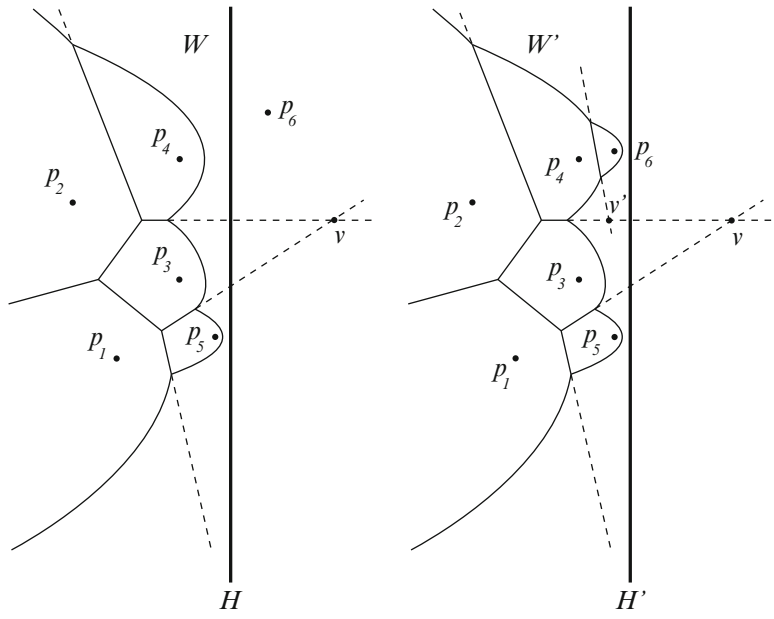
Given a set S of n point sites, the problem is to efficiently construct one of $V(S)$ or $D(S)$; the dual structure can then easily be obtained in linear time.

Generalizations

Voronoi diagrams can be generalized in several ways. Instead of point sites, other geometric objects can be considered. One can replace the Euclidean distance with distance measures more suitable to model a given situation. Instead of forming regions of all points that have the same nearest site, one can consider higher-order Voronoi diagrams where all points share a region for which the nearest k sites are the same, for some k between 2 and $n-1$. Many more variants can be found in [9] and [1]. Abstract Voronoi diagrams provide a unifying framework for some of the variants mentioned; see the corresponding chapter in this encyclopedia.

Key Results

Quite a few algorithms for constructing the Voronoi diagram or the Delaunay triangulation



Voronoi Diagrams and Delaunay Triangulations, Fig. 3 The sweepline advancing to the right

of n points in the Euclidean plane have been developed.

Divide and Conquer

The first algorithm was presented in the seminal paper [11], which gave birth to the field of computational geometry. It applies the *divide and conquer* paradigm. Site set S is split by a line into subsets L and R of equal cardinality. After recursively computing $V(L)$ and $V(R)$, one needs to compute the bisector $B(L, R)$, the locus of all points in the plane that have a nearest neighbor in L and in R. This bisector is an unbounded monotone polygonal chain. In time $O(n)$ one can find a starting segment of $B(L, R)$ at infinity, and trace the chain through $V(L)$ and $V(R)$ simultaneously. Thus, the algorithm runs in time $O(n \log n)$ and linear space, which is optimal.

Sweep

How to design a left-to-right *sweepline* algorithm for constructing $V(S)$ is not obvious. When the advancing sweepline H enters the Voronoi region of p before site p has been detected, it is not clear how to correctly maintain the Voronoi diagram along H. This difficulty has been overcome in [7] by applying a transformation that ensures that each site is the leftmost point of its Voronoi region. In [10] and [4], a more direct version of this approach was suggested. At each time during the sweep, one maintains the Voronoi diagram of all point sites to the left of sweepline H, and of H itself, which is considered a site of its own; see Fig. 3. Because the bisector of a point and a line is a parabola, the Voronoi region of H is bounded by a connected chain of parabolic segments, called the *wavefront* W. As H moves to the right, W follows at half the speed. Each point z to the left of W is closer to some point site p left of H than to H and, all the more, to all point sites to the right of H that are yet to be discovered. Thus, the Voronoi regions of the point sites to the left of W keep growing, as sweepline H proceeds, along the extensions of Voronoi edges beyond W; these *spikes* are depicted by dashed lines in Fig. 3.

There are two kinds of events one needs to handle during the sweep. When sweepline H hits a new point site (like point p_6 in Fig. 3), a new wave separating this point site from H must be added to W. When wavefront W hits the intersection of two neighboring spikes, the wave between them must be removed from W;

this will first happen in Fig. 3 when W' arrives at v'. Intersections between neighboring spikes can be determined as in the standard line segment intersection algorithm [2]. There are only $O(n)$ many events, one for each point site and one for each Voronoi vertex v of $V(S)$. Since wavefront W is always of linear size, the sweepline algorithm runs in $O(n \log n)$ time using linear space.

Reduction to Convex Hull

A rather different approach [3] obtains the Delaunay triangulation in dimension 2 from the convex hull in dimension 3, which can itself be constructed in time $O(n \log n)$. As suggested in [6], one vertically lifts the point sites to the paraboloid $Z = X^2 + Y^2$ in 3-space. The lower convex hull of the lifted points, projected onto the XY-plane, equals the Delaunay triangulation, $D(S)$.

Incremental Construction

Another, very intuitive algorithm first suggested in [8] constructs the Delaunay triangulation incrementally. In order to insert a new point site p_i into an existing Delaunay triangulation $D(S_{i-1})$, one first finds the triangle containing p_i and connects p_i to its vertices by line segments. Should p_i be contained in the circumcircles of adjacent triangles, the Delaunay property must be restored by *edge flips* that replace the common edge of two adjacent triangles T, T' by the other diagonal of the convex quadrilateral formed by T and T'. If the insertion sequence of the p_i is randomly chosen, a running time in $O(n \log n)$ can be expected. Details on all algorithms can be found in [1].

Applications

Although of linear size, Voronoi diagram and Delaunay triangulation contain a lot of information on the point set S. Once $V(S)$ or $D(S)$ are available, quite a few distance problems can be solved very efficiently. We mention only the most basic applications here and refer to [1] and [9] for further reading.

By definition, the Voronoi diagram reduces the *post office* or *nearest neighbor* problem to a point location problem: given an arbitrary query point z, the site in S nearest to z can be found by determining the Voronoi region containing z. In order to find the *largest empty circle* whose center z lies inside a convex polygon C over m vertices, one needs to inspect only three types of candidates for z, the vertices of $V(S)$, the intersections of the edges of $V(S)$ with the boundary of C, and the vertices of C. All these can be done in time $O(n + m)$.

If the site set S is split into subsets L and R, then the closest pair $p \in L$ and $q \in R$ forms an edge of the Delaunay triangulation $D(S)$ (which crosses the Voronoi edge separating the regions of p and q). This fact has nice consequences. First, the nearest neighbor of a site $p \in S$ must be one of its neighboring vertices in $D(S)$. Hence, *all nearest neighbors* and the *closest pair* in S can be found in linear time once $D(S)$ is available, because $D(S)$ has only $O(n)$ many edges. Second, $D(S)$ contains the *minimum spanning tree* of S, which can be extracted from $D(S)$ in linear time.

Remarkable and useful is the *equiangularity property* of $D(S)$. Of all (exponentially many) triangulations of S, the Delaunay triangulation maximizes the ascending sequence of angles occurring in the triangles, with respect to lexicographic order. In particular, the minimum angle is as large as possible. In fact, if a triangulation is not Delaunay, it must contain two adjacent triangles, such that the circumcircle of one contains the third vertex of the other. By flipping their common edge, a new triangulation with larger angles is obtained.

Cross-References

- 3D Conforming Delaunay Triangulation
- Abstract Voronoi Diagrams
- Delaunay Triangulation and Randomized Constructions
- Optimal Triangulation

Recommended Reading

1. Aurenhammer F, Klein R, Lee DT (2013) Voronoi diagrams and delaunay triangulations. World Scientific, Singapore
2. Bentley J, Ottman T (1979) Algorithms for reporting and counting geometric intersections. IEEE Trans Comput C-28:643–647
3. Brown KQ (1979) Voronoi diagrams from Convex Hulls. Inf Process Lett 9:223–228
4. Cole R (1989) as reported by ÓDúnlaing, oral communication
5. Descartes R (1644) Principia philosophiae. Ludovicus Elsevirius, Amsterdam
6. Edelsbrunner H, Seidel R (1986) Voronoi diagrams and arrangements. Discret Comput Geom 1:25–44
7. Fortune S (1978) A sweepline algorithm for Voronoi diagrams. Algorithmica 2:153–174
8. Green PJ, Sibson RR (1978) Computing dirichlet tessellations in the plane. Comput J 21: 168–173
9. Okabe A, Boots B, Sugihara K, Chiu SN (2000) Spatial tessellations: concepts and applications of Voronoi diagrams. Wiley, Chichester
10. Seidel R (1988) Constrained delaunay triangulations and Voronoi diagrams with obstacles. Technical report 260, TU Graz
11. Shamos MI, Hoey D (1975) Closest-point problems. In: Proceedings 16th annual IEEE symposium on foundations of computer science, Berkeley, pp 151–162

W

Wait-Free Synchronization

Mark Moir
Sun Microsystems Laboratories, Burlington, MA, USA

Years and Authors of Summarized Original Work

1991; Herlihy

Problem Definition

The traditional use of locking to maintain consistency of shared data in concurrent programs has a number of disadvantages related to software engineering, robustness, performance, and scalability. As a result, a great deal of research effort has gone into *nonblocking* synchronization mechanisms over the last few decades.

Herlihy's seminal paper *Wait-Free Synchronization* [12] studied the problem of implementing concurrent data structures in a *wait-free manner*, i.e., so that every operation on the data structure completes in a finite number of steps by the invoking thread, regardless of how fast or slow other threads run and even if some or all of them halt permanently. Implementations based on locks are not wait-free because, while one thread holds a lock, others can take an unbounded number of steps waiting to acquire the lock. Thus, by requiring implementations to be wait-free, some of the disadvantages of locks may potentially be eliminated.

The first part of Herlihy's paper examined the power of different synchronization primitives for wait-free computation. He defined the *consensus number* of a given primitive as the maximum number of threads for which we can solve wait-free consensus using that primitive (together with read-write registers). The consensus problem requires participating threads to *agree* on a value (e.g., true or false) amongst values proposed by the threads. The ability to solve this problem is a key indicator of the power of synchronization primitives because it is central to many natural problems in concurrent computing. For example, in a software transactional memory system, threads must agree that a particular transaction either committed or aborted.

Herlihy established a *hierarchy* of synchronization primitives according to their consensus number. He showed (i) that the consensus number of read-write registers is 1 (so wait-free consensus cannot be solved for even two threads), (ii) that the consensus number of stacks and FIFO queues is 2, and (iii) that there are so-called *universal* primitives, which have consensus number ∞. Common examples include `compare-and-swap` (`CAS`) and the `load-linked/store-conditional` (`LL/SC`) pair.

There are a number of papers which examine Herlihy's hierarchy in more detail. These show that seemingly minor variations in the model or

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

in the semantics of primitives can have a surprising effect on results. Most of this work is primarily of theoretical interest. The key practical point to take away from Herlihy's hierarchy is that we need universal primitives to support effective wait-free synchronization in general. Recognizing this fact, all modern shared-memory multiprocessors provide some form of universal primitive.

Herlihy additionally showed that a solution to consensus can be used to implement any shared object in a wait-free manner, and thus that any universal primitive suffices for this purpose. He demonstrated this idea using a so-called *universal construction*, which takes sequential code for an object and creates a wait-free implementation of the object using consensus to resolve races between concurrent operations. Despite the important practical ramifications of this result, the universal construction itself was quite impractical. The basic idea was to build a list of operations, using consensus to determine the order of operations, and to allow threads to iterate over the list applying the operations in order to determine the current state of the object. The construction required $O(N^3)$ space to ensure enough operations are retained to allow the current state to be determined. It was also very slow, requiring many threads to recompute the same information, and thus preventing parallelism between operations in addition.

Later, Herlihy [13] presented a more concrete universal construction based on the `LL`/`SC` instruction pair. This construction required $N + 1$ copies of the object for N threads and still did not admit any parallelism; thus it was also not practical. Despite this, work following on from Herlihy's has brought us to the point today that we can support practical programming models that provide nonblocking implementations of arbitrary shared objects. The remainder of this chapter discusses the state of nonblocking synchronization today, and mentions some history along the way.

Weaker Nonblocking Progress Conditions

Various researchers, including us, have had some success attempting to overcome the disadvantages of Herlihy's wait-free constructions. However, the results remain impractical due to excessive overhead and overly complicated algorithms. In fact, there are still no nontrivial wait-free shared objects in widespread practical use, either implemented directly or using universal constructions.

The biggest advances towards practicality have come from considering weaker progress conditions. While theoreticians worked on wait-free implementations, more pragmatic researchers sought lock-free implementations of shared objects. A *lock-free* implementation guarantees that, after a finite number of steps of any operation, *some* operation completes. In contrast to wait-free algorithms, it is in principle possible for one operation of a lock-free data structure to be continually starved by others. However, this rarely occurs in practice, especially because contention control techniques such as exponential backoff [1] are often used to reduce contention when it occurs, which makes repeated interference even more unlikely. Thus, the lack of a strong progress guarantee like wait-freedom has often been found to be acceptable in practice.

The observation that weaker nonblocking progress conditions allow simpler and more practical algorithms led Herlihy et al. [15] to define an even weaker condition: An *obstruction-free* algorithm does not guarantee that an operation completes unless it eventually encounters no more interference from other operations. In our experience, obstruction-free algorithms are easier to design, simpler, and faster in the common uncontended case than lock-free algorithms. The price paid for these benefits is that obstruction-free algorithms can "livelock", with two or more operations repeatedly interfering with each other forever. This is not merely a theoretical concern: it has been observed to occur in practice [16]. Fortunately, it is usually straightforward to eliminate livelock in practice through contention control mechanisms that control and manipulate when operations are executed to avoid repeated interference.

The obstruction-free approach to synchronization is thus to design simple and efficient

algorithms for the weak obstruction-free progress condition, and to integrate orthogonal contention control mechanisms to facilitate progress when necessary. By largely separating the difficult issues of correctness and progress, we significantly ease the task of designing effective nonblocking implementations: the algorithms are not complicated by tightly coupled mechanisms for achieving lock-freedom, and it is easy to modify and experiment with contention control mechanisms because they are separate from the algorithm and do not affect its correctness. We have found this approach to be very powerful.

Transactional Memory

The severe difficulty of designing and verifying correct nonblocking data structures has led researchers to investigate the use of tools to produce them, rather than designing them directly. In particular, *transactional memory* [5, 17, 23] has emerged as a promising direction. Transactional memory allows programmers to express sections of code that should be executed atomically, and the transactional memory system (implemented in hardware, software, or a combination of the two) is responsible for managing interactions between concurrent transactions to ensure this atomicity. Here we concentrate on software transactional memory (STM).

The progress guarantee made by a concurrent data structure implemented using STM depends on the STM implementation. It is possible to characterize the progress conditions of transactional memory implementations in terms of a system of threads in which each operation on a shared data structure is executed by repeatedly attempting to apply it using a transaction until an attempt successfully commits. In this context, say the transactional memory implementation is obstruction-free if it guarantees that, if a thread repeatedly executes transactions and eventually encounters no more interference from other threads, then it eventually successfully commits a transaction.

Key Results

This section briefly discusses some of the most relevant results concerning nonblocking synchronization, and obstruction-free synchronization in particular.

While progress towards practicality was made with lock-free implementations of shared objects as well as lock-free STM systems, this progress was slow because simultaneously ensuring correctness and lock-freedom proved difficult. Before the introduction of obstruction-freedom, the lock-free STMs still had some severe disadvantages such as the need to declare and initialize all memory to be accessed by transactions in advance, the need for transactions to know in advance which memory locations they will access, unacceptable constraints on the layout of such memory, etc.

In addition to the work on tools such as STM for building nonblocking data structures, there has been a considerable amount of work on direct implementations. While this work has not yielded any practical wait-free algorithms, a handful of practical lock-free implementations for simple data structures such as queues and stacks have been achieved [21, 24]. There are also a few slightly more ambitious implementations in the literature that are arguably practical, but the algorithms are complicated and subtle, many are incorrect, and almost none has a formal proof. Proofs for such algorithms are challenging, and minor changes to the algorithm require the proofs to be redone.

The next section, discusses some of the results that have been achieved by applying the obstruction-free approach. The remainder of this section, briefly discusses a few results related to the approach itself.

An important practical aspect of using an obstruction-free algorithm is how contention is managed when it arises. In introducing obstruction-freedom, Herlihy et al. [15] explained that contention control is necessary to facilitate progress in the face of contention because obstruction-free algorithms do not directly make any progress guarantee in this case. However,

they did not directly address *how* contention control mechanisms could be used in practice.

Subsequently, Herlihy et al. [16] presented a dynamic STM system (see next section) that provides an interface for a modular contention manager, allowing for experimentation with alternative contention managers. Scherer and Scott [22] experimented with a number of alternatives, and found that the best contention manager depends on the workload. Guerraoui et al. [9] described an implementation that supports changing contention managers on the fly in response to changing workload conditions.

All of the contention managers discussed in the above-mentioned papers are ad hoc contention managers based on intuition; no analysis is given of what guarantees (if any) are made by the contention managers. Guerraoui et al. [10] made a first step towards a formal analysis of contention managers by showing that their `Greedy` contention manager guarantees that every transaction eventually completes. However, using the `Greedy` contention manager results in a blocking algorithm, so their proof necessarily assumes that threads do not fail while executing transactions.

Fich et al. [7] showed that any obstruction-free algorithm can be automatically transformed into one that is *practically* wait-free in any real system. "Practically" is said because the wait-free progress guarantee depends on partial synchrony that exists in any real system, but the transformed algorithm is not technically wait-free, because this term is defined in the context of a fully asynchronous system. Nonetheless, an algorithm achieved by applying the transformation of Fich et al. to an obstruction-free algorithm does guarantee progress to non-failed transactions, even if other transactions fail.

Work on incorporating contention management techniques into obstruction-free algorithms has mostly been done in the context of STM, so the contention manager can be called directly from the STM implementation. Thus, the programmer using the STM need not be concerned with how contention management is integrated, but this does not address how contention management is integrated into direct implementations of obstruction-free data structures.

One option is for the programmer to manually insert calls to a contention manager, but this approach is tedious and error prone. Guerraoui et al. [11] suggested a version of this approach in which the contention manager is abstracted out as a failure detector. They also explored what progress guarantees can be made by what failure detectors.

Attiya et al. [4] and Aguilera et al. [2] suggested changing the semantics of the data structure's operations so that they can return a special value in case of contention, thus allowing contention management to be done outside the data structure implementation. These approaches still leave a burden on the programmer to ensure that these special values are always returned by an operation that cannot complete due to contention, and that the correct special value is returned according to the prescribed semantics.

Another option is to use system support to ensure that contention management calls are made frequently enough to ensure progress. This support could be in the form of compiled-in calls, runtime support, signals sent upon expiration of a timer, etc. But all of these approaches have disadvantages such as not being applicable in general purpose environments, not being portable, etc.

Given that it remains challenging to design and verify direct obstruction-free implementations of shared data structures, and that there are disadvantages to the various proposals for integrating contention control mechanisms into them, using tools such as STMs with built-in contention management interfaces is the most convenient way to build nonblocking data structures.

Applications

The obstruction-free approach to nonblocking synchronization was introduced by Herlihy et al. [15], who used it to design a double-ended queue (deque) based on the widely available `CAS` instruction. All previous nonblocking deques

either require exotic synchronization instructions such as `double-compare-and-swap` (`DCAS`), or have the disadvantage that operations at opposite ends of the queue always interfere with each other.

Herlihy et al. [16] introduced Dynamic STM (DSTM), the first STM that is dynamic in the following two senses: new objects can be allocated on the fly and subsequently accessed by transactions, and transactions do not need to know in advance what objects will be accessed. These two advantages made DSTM much more useful than previous STMs for programming dynamic data structures. As a result, nonblocking implementations of sophisticated shared data structures such as balanced search trees, skip lists, dynamic hash tables, etc. were suddenly possible.

The obstruction-free approach played a key role in the development of both of the results mentioned above: Herlihy et al. [16] could concentrate on the functionality and correctness of DSTM without worrying about how to achieve stronger progress guarantees such as lock-freedom.

The introduction of DSTM and of the obstruction-free approach have led to numerous improvements and variations by a number of research groups, and most of these have similarly followed the obstruction-free approach. However, Harris and Fraser [8] presented a dynamic STM called OSTM with similar advantages to DSTM, but it is lock-free. Experiments conducted at the University of Rochester [20] showed that DSTM outperformed OSTM by an order of magnitude on some workloads, but that OSTM outperformed DSTM by a factor of 2 on others. These differences are probably due to various design decisions that are (mostly) orthogonal to the progress condition, so it is not clear what we can conclude about how the choice of progress condition affects performance in this case.

Perhaps a more direct comparison can be made between another pair of algorithms, again an obstruction-free one by Herlihy et al. [14] and a similar but lock-free one by Harris and Fraser [8]. These algorithms, invented independently of each other, implement `MCAS` (`CAS` generalized to access M independently chosen memory locations). The two algorithms are very similar, and a close comparison revealed that the only real differences between them were due to Harris and Fraser's desire to have a lock-free implementation. As a result of this, their algorithm is somewhat more complicated, and also requires a minimum of $3M + 1$ `CAS` operations, whereas the algorithm of Herlihy et al. [14] requires only $2M + 1$. The authors are unaware of any direct performance comparison of these algorithms, but they believe the obstruction-free one would outperform the lock-free one, particularly in the absence of conflicting `MCAS` operations.

Open Questions

Because transactional memory research has grown out of research into nonblocking data structures, it was long considered mandatory for STM implementations to support the development of nonblocking data structures. Recently, however, a number of researchers have observed that at least the software engineering benefits of transactional memory can be delivered even by a blocking STM. There are ongoing debates whether STM needs to be nonblocking and whether there is a fundamental cost to being nonblocking.

While we agree that blocking STMs are considerably easier to design, and that in many cases a blocking STM is acceptable, this is not always true. Consider, for example, an interrupt handler that shares data with the interrupted thread. The interrupted thread will not run again until the interrupt handler completes, so it is critical that the interrupted thread does not block the interrupt handler. Thus, if using STM is desired to simplify the code for accessing this shared data, the STM *must* be nonblocking. The authors are therefore motivated to continue research aimed at improving nonblocking STMs and to understand what fundamental gap, if any, exists between blocking and nonblocking STMs.

Progress in improving the common-case performance of nonblocking STMs continues [19], and the authors see no reason to believe that nonblocking STMs should not be very competitive

with blocking STMs in the common case, i.e., until the system decides that one transaction should not wait for another that is delayed (an option that is not available with blocking STMs).

It is conjectured that indeed a separation between blocking and nonblocking STMs can be proved according to some measure, but that this will not imply significant performance differences in the common case. Indeed results of Attiya et al. [3] show a separation between obstruction-free and blocking algorithms according to a measure that counts the number of distinct base objects accessed by the implementation plus the number of "memory stalls", which measure how often the implementation can encounter contention for a variable from another thread. While this result is interesting, it is not clear that it is useful for deciding whether to implement blocking or obstruction-free objects, because the measure does not account for the time spent waiting by blocking implementations, and thus is biased in their favor. For now, remain optimistic that STMs can be made to be nonblocking without paying a severe performance price in the common case.

Another interesting question, which is open as far as the authors know, is whether there is a fundamental cost to implementing stronger nonblocking progress conditions versus obstruction-freedom. Again, they conjecture that there is. It is known that there is a fundamental difference between obstruction-freedom and lock-freedom in systems that support only reads and writes: It is possible to solve obstruction-free consensus but not lock-free consensus in this model [15]. While this is a fascinating observation, it is mostly irrelevant from a practical standpoint as all modern shared memory multiprocessors support stronger synchronization primitives such as `CAS`, with which it is easy to solve consensus, even wait-free. The interesting question therefore is whether there is a fundamental cost to being lock-free as opposed to obstruction-free in real systems.

To have a real impact on design directions, such results need to address common case performance, or some other measure (perhaps space) that is relevant to everyday use. Many lower bound results establish a separation in worst-case time complexity, which does not necessarily have a direct impact on design decisions, because the worst case may be very rare. So far, efforts to establish a separation according to potentially useful measures have only led to stronger results than we had conjectured were possible. In the authors first attempt [18], they tried to establish a separation in the number of CAS instructions needed in the absence of contention to solve consensus, but found that this was not a very useful measure, as were able to come up with a wait-free implementation that avoids CAS in the absence of contention. The second attempt [6] was to establish a separation according to the *obstruction-free step complexity* measure, which counts the maximum number of steps to complete an operation once the operation encounters no more contention. They knew we could implement obstruction-free DCAS with constant obstruction-free step complexity, and attempt to prove this impossible for lock-free DCAS, but achieved such an algorithm. These experiences suggest that, in addition to their direct advantages, obstruction-free algorithms may provide a useful stepping stone to algorithms with stronger progress properties.

Finally, while a number of contention managers have proved effective for various workloads, it is an open question whether a single contention manager can adapt to be competitive with the best on all workloads, and how close it can come to making optimal contention management decisions. Experience to date suggests that this will be very challenging to achieve. Therefore, as in any system, the first priority should be avoiding contention in the first place. Fortunately, transactional memory has the potential to make this much easier than in lock-based programming models, because it offers the benefits of fine-grained synchronization without the programming complexity that accompanies fine-grained locking schemes.

Cross-References

- ▸ Concurrent Programming, Mutual Exclusion
- ▸ Linearizability

Recommended Reading

1. Agarwal A, Cherian M (1989) Adaptive backoff synchronization techniques. In: Proceedings of the 16th annual international symposium on computer architecture. ACM Press, New York, pp 396–406
2. Aguilera MK, Frolund S, Hadzilacos V, Horn SL, Toueg S (2006) Brief announcement: abortable and query-abortable objects. In: Proceedings of the 20th annual international symposium on distributed computing
3. Attiya H, Guerraoui R, Hendler D, Kouznetsov P (2006) Synchronizing without locks is inherently expensive. In: PODC'06: proceedings of the twenty-fifth annual ACM symposium on principles of distributed computing. ACM Press, New York, pp 300–307
4. Attiya H, Guerraoui R, Kouznetsov P (2005) Computing with reads and writes in the absence of step contention. In: Proceedings of the 19th annual international symposium on distributed computing
5. Damron P, Fedorova A, Lev Y, Luchangco V, Moir M, Nussbaum D (2006) Hybrid transactional memory. In: Proceedings of the 12th symposium on architectural support for programming languages and operating systems
6. Fich F, Luchangco V, Moir M, Shavit N (2005) Brief announcement: obstruction-free step complexity: lock-free DCAS as an example. In: Proceedings of the 19th annual international symposium on distributed computing
7. Fich F, Luchangco V, Moir M, Shavit N (2005) Obstruction-free algorithms can be practically wait-free. In: Proceedings of the 19th annual international symposium on distributed computing
8. Fraser K, Harris T (2004) Concurrent programming without locks. http://www.cl.cam.ac.uk/netos/papers/2004-cpwl-submission.pdf
9. Guerraoui R, Herlihy M, Pochon B (2005) Polymorphic contention management. In: Proceedings of the 19th annual international symposium on distributed computing
10. Guerraoui R, Herlihy M, Pochon B (2005) Toward a theory of transactional contention managers. In: Proceedings of the 24th annual ACM symposium on principles of distributed computing, pp 258–264
11. Guerraoui R, Kapalka M, Kouznetsov P (2006) The weakest failure detector to boost obstruction freedom. In: Proceedings of the 20th annual international symposium on distributed computing
12. Herlihy M (1991) Wait-free synchronization. ACM Trans Program Lang Syst 13(1):124–149
13. Herlihy M (1993) A methodology for implementing highly concurrent data objects. ACM Trans Program Lang Syst 15(5):745–770
14. Herlihy M, Luchangco V, Moir M (2002) Obstruction-free mechanism for atomic update of multiple non-contiguous locations in shared memory. US Patent Application 20040034673
15. Herlihy M, Luchangco V, Moir M (2003) Obstruction-free synchronization: double-ended queues as an example. In: Proceedings of the 23rd international conference on distributed computing systems
16. Herlihy M, Luchangco V, Moir M, Scherer W III (2003) Software transactional memory for supporting dynamic-sized data structures. In: Proceedings of the 22th annual ACM symposium on principles of distributed computing, pp 92–101
17. Herlihy M, Moss JEB (1993) Transactional memory: architectural support for lock-free data structures. In: Proceedings of the 20th annual international symposium on computer architecture, pp 289–300
18. Luchangco V, Moir M, Shavit N (2005) On the uncontended complexity of consensus. In: Proceedings of the 17th annual international symposium on distributed computing
19. Marathe VJ, Moir M (2008) Toward high performance nonblocking software transactional memory. In: Proceedings of the 13th ACM SIGPLAN symposium on principles and practice of parallel programming. ACM, New York, pp 227–236
20. Marathe V, Scherer W, Scott M (2005) Adaptive software transactional memory. In: Proceedings of the 19th annual international symposium on distributed computing
21. Michael M, Scott M (1998) Nonblocking algorithms and preemption-safe locking on multiprogrammed shared memory multiprocessors. J Parallel Distrib Comput 51(1):1–26
22. Scherer W, Scott M (2005) Advanced contention management for dynamic software transactional memory. In: Proceedings of the 24th annual ACM symposium on principles of distributed computing
23. Shavit N, Touitou D (1997) Software transactional memory. Distrib Comput Special Issue 10:99–116
24. Treiber R (1986) Systems programming: coping with parallelism. Technical Report RJ5118, IBM Almaden Research Center

Wake-Up Problem in Multi-Hop Radio Networks

Tomasz Jurdziński[1] and Dariusz R. Kowalski[2]
[1]Institute of Computer Science, University of Wrocław, Wrocław, Poland
[2]Department of Computer Science, University of Liverpool, Liverpool, UK

Keywords

Ad hoc radio network; Broadcasting; Clock synchronization; Leader election; Probabilistic method; Radio synchronizer; Wake-up

Years and Authors of Summarized Original Work

2004; Chlebus, Kowalski
2004; Chrobak, Gąsieniec, Kowalski
2005; Chlebus, Gąsieniec, Kowalski, Radzik
2007; Chrobak, Gąsieniec, Kowalski

Problem Definition

A *radio network* is modeled as a directed, strongly connected graph G with n *nodes*. The nodes of a network G correspond to transmitting/receiving wireless devices, and directed edges represent their immediately reached neighbors: if a node w is within the transmission range of a node v, then G contains an edge (v, w). We call w an *out-neighbor* of v and v an *in-neighbor* of w.

Each node v has a unique *label* ℓ_v from the set $[N] = \{1, \ldots, N\}$, where $N = O(n)$. Initially, each node knows only its label and the values of n and N.

The time is divided into discrete time steps. It is assumed that nodes have unlimited computing power and can perform arbitrary computations within one time step. However, only one transmission or message receipt is allowed in one time step. Each node has its own local clock, whose initial value at the time of its activation is 0. All local clocks run at the same speed.

A message M transmitted in time step t by a node v is sent instantly to all its out-neighbors. However, an out-neighbor w of v *successfully receives* M in time step t only if no collision occurred in this time step, that is, if no other in-neighbor of w transmits in step t. Collision cannot be distinguished from the background noise: if w does not receive any message in time step t, it knows that either none of its in-neighbors transmitted in step t, or that at least two did, but it does not know which of these two events occurred. It is assumed that nodes only transmit wake-up signals (to their neighbors); no other messages are used.

A *wake-up schedule* is a vector $\omega = (\omega_x)_{x \in V}$, where ω_x denotes the time step in which x wakes-up spontaneously. For any set $X \subseteq V$, ω_X denotes the earliest wake-up time step in X, i.e., $\omega_X = \min_{x \in X} \omega_x$. Without loss of generality, one can assume that $\omega_V = \min_{x \in V} \omega_x = 0$. A *wake-up network* is the pair $\langle G, \omega \rangle$, where G is a radio network G and ω is a wake-up schedule.

A *deterministic wake-up protocol* $\mathcal{W}$ is a function that, for each label ℓ and for each $\tau = 1, 2, 3, \ldots$, given all past messages received by the node v with label $\ell_v = \ell$, specifies whether v will transmit the wake-up signal in time step τ since its activation. A *randomized wake-up protocol* is defined for each node as a probability distribution over the class of deterministic protocols for that node.

The running time of a wake-up protocol $\mathcal{W}$ is the smallest T such that, for any wake-up network $\langle G, \omega \rangle$, all nodes are activated by time T.

Synchronizers

All efficient deterministic wake-up algorithms are based on the combinatorial notion of a *radio synchronizer*, called also a *synchronizer* for short (see also a simpler notion of related structures called selectors [5], efficiently exploited in the context of broadcasting in radio networks).

Let $\mathcal{S} = \{\mathcal{S}^x\}_{x \in [N]}$, where each $\mathcal{S}^x = S_1^x S_2^x \ldots S_m^x$ is a 0-1 sequence of length m. The set $\mathcal{S}$ is a (N, k, m)-*synchronizer* if it satisfies the following property:

(∗) For any nonempty set $X \subseteq [N]$ of cardinality at most k, and for any wake-up schedule ω, there exists t, where $\omega_X < t \leq \omega_X + m$, such that,

$$\sum_{x \in X} S^x_{t-\omega_x} = 1.$$

It is assumed here that $S^x_i = 0$ for $i \leq 0$.

The set $\mathcal{S}$ as above can be interpreted as a transmission protocol, where $S^x_i = 1$ indicates that node x transmits in time step $\omega_x + i$. Thus the condition (∗) states that, in at most m time steps after the first node in X wakes-up, there will be a time step when exactly one node in X transmits.

More details about radio synchronizers and synchronization protocols can be found in the survey [8].

Key Results

A Deterministic Wake-Up Protocol

Lemma 1 ([3, 4]) *Let $C \geq 31$ be an integer constant. For each N and $k \leq N$, there exists an (N, k, m)-synchronizer with $m = Ck^2 \log N$.*

The key ingredient of the wake-up algorithm in [3, 4] ([3] is a conference version of [4]) is an application of (N, k, m)-synchronizer with $k = N^{1/3}$ and $m = CN^{2/3} \log N$, where $C = 31$. The analysis of this algorithm relies on the fact that it is sufficient to prove that the algorithm satisfies claimed time bounds for *path graphs*. A directed graph H is called a *path graph* if the nodes of H can be partitioned into sets L_i, $i = 0, \ldots, D$, each with a distinguished node $v_i \in L_i$ and the edges of H are of the form (v, v_{i+1}), where $0 \leq i < D$ and $v \in L_i$. Moreover, $L_D = \{v_D\}$.

Theorem 1 ([3, 4]) *There exists a deterministic protocol that completes the wake-up process in each n-node strongly connected directed graph in time $O(n^{5/3} \log n)$.*

A Randomized Wake-Up Protocol

In [7], the authors presented a randomized wake-up protocol Probability Increase for complete networks working in time $O(\log n \log(1/\epsilon))$ with probability $1 - \epsilon$ (see [6] for deterministic wake-up algorithms for complete graphs). Using this protocol along with appropriate formal analysis, one can obtain a randomized Monte Carlo wake-up protocol for general multi-hop radio networks.

Theorem 2 ([3, 4]) *One can build a randomized protocol which completes wake-up in time $O(D \log n \log(n/\epsilon))$ in each wake-up network with n nodes and diameter D with probability at least $1 - \epsilon$.*

The Monte Carlo protocol from Theorem 2 can be modified to obtain Las Vegas protocol with low expected running time.

Theorem 3 ([3, 4]) *One can build a randomized protocol which completes wake-up in expected time $O(D \log^2 n)$ in each wake-up network with n nodes and diameter D.*

All the above randomized protocols do not require labels.

Applications

Universal Synchronizers and Faster Wake-Up Protocols

The notion of a synchronizer has been generalized to a *universal synchronizer* [1].

Let $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a nondecreasing function. Let $\mathcal{S} = \{\mathcal{S}^x\}_{[N]}$, where each $\mathcal{S}^x = S^x_1 S^x_2 \ldots S^x_m$ is a 0-1 sequence of length $g(N, N)$. The set $\mathcal{S}$ is a (N, g)-*universal synchronizer* if it satisfies the following property:

(∗) For any nonempty set $X \subseteq [N]$ and for any wake-up schedule ω, there exists t, where $\omega_X < t \leq \omega_X + g(N, |X|)$, such that,

$$\sum_{x \in X} S^x_{t-\omega_x} = 1.$$

Chlebus and Kowalski proved in [1] that there exist (N, g)-universal synchronizers for $g(k) = O(k \min\{k, \sqrt{n}\} \log n)$. Using this result they

showed that there exists a wake-up protocol running in time $O(n^{3/2} \log n)$. In [2], Chlebus *et al.* provided an existential proof of the fact that much shorter universal synchronizers exist and obtained a corresponding faster wake-up protocol.

Lemma 2 ([2]) *For each N there exists a (N, g)-universal synchronizer for $g(N, k) = ck \log k \log N$, where c is a fixed constant.*

Theorem 4 ([2]) *There exists a deterministic protocol that completes the wake-up process in each n-node strongly connected directed graph in time $O(n \log^2 n)$.*

Leader Election and Clock Synchronization

In [3,4], applications of wake-up protocols for the problems of *leader election* and *clock synchronization* were considered.

In the *leader election* problem, the goal is to designate one node as *the leader*, and to announce its identity to all nodes in the network. In the *clock synchronization* problem, upon the completion of the protocol, all nodes must agree on a common global time. For clock synchronization, messages may include numerical values representing the global time.

It has been shown in [3, 4] that any wake-up protocol $\mathcal{W}$ (deterministic or randomized) can be transformed into a leader election protocol or a clock synchronization protocol with only a logarithmic overhead. The leader election protocol is obtained by an execution of appropriately composed $O(\log n)$ executions of a wake-up protocol, in which nodes gradually learn consecutive bits of the node with the largest label. In the clock synchronization protocol, the leader is elected first and then it broadcasts its clock state over the whole network.

Open Problems

The exact complexity of the wake-up problem is not known – there is a logarithmic gap between the complexities of the best known protocols and lower bounds. No efficient algorithms for a construction of (universal) synchronizers described in Lemmata 1 and 2 are known (i.e., polynomial time construction with a polylogarithmic overhead to the length), and thus the results from Theorems 1 and 4 are nonconstructive either. It is not known whether the logarithmic overhead in the complexity of leader election and clock synchronization with respect to wake-up is necessary.

Cross-References

- Deterministic Broadcasting in Radio Networks
- Randomized Broadcasting in Radio Networks

Recommended Reading

1. Chlebus BS, Kowalski DR (2004) A better wake-up in radio networks. In: Chaudhuri S, Kutten S (eds) PODC. ACM, pp 266–274
2. Chlebus BS, Gasieniec L, Kowalski DR, Radzik T (2005) On the wake-up problem in radio networks. In: Caires L, Italiano GF, Monteiro L, Palamidessi C, Yung M (eds) ICALP. Lecture notes in computer science, Lisbon, Portugal, vol 3580. Springer, pp 347–359
3. Chrobak M, Gasieniec L, Kowalski DR (2004) The wake-up problem in multi-hop radio networks. In: Munro JI (ed) SODA. Portland, Oregon, USA, SIAM, pp 992–1000
4. Chrobak M, Gasieniec L, Kowalski DR (2007) The wake-up problem in multihop radio networks. SIAM J Comput 36(5):1453–1471
5. De Bonis A, Gasieniec L, Vaccaro U (2005) Optimal two-stage algorithms for group testing problems. SIAM J Comput 34(5): 1253–1270
6. Gasieniec L, Pelc A, Peleg D (2001) The wakeup problem in synchronous broadcast systems. SIAM J Discret Math 14(2):207–222
7. Jurdzinski T, Stachowiak G (2005) Probabilistic algorithms for the wake-up problem in single-hop radio networks. Theory Comput Syst 38(3): 347–367
8. Kowalski DR (2011) Coordination problems in ad hoc radio networks. In: Nikoletseas S, Rolim JD (eds) Theoretical aspects of distributed computing in sensor networks. Springer

Wavelet Trees

Roberto Grossi
Dipartimento di Informatica, Università di Pisa, Pisa, Italy

Keywords

2D data; Data compression; Data structures; Geometric points; Permutations; Reorderings; Sequences; Strings; Succinct representations

Years and Authors of Summarized Original Work

2003; Grossi, Gupta, Vitter
2012; Makris
2014; Navarro

Problem Definition

The wavelet tree is a data structure that represents a recursive partition of a sequence S of length n according to its symbols. Letting $\Sigma = \{1, \ldots \sigma\}$ be the alphabet of symbols of S, the wavelet tree for S has the root representing S itself and σ leaves representing the positions of the symbols: leaf $c \in \Sigma$ represents all the positions i such that $S[i] = c$ and $1 \leq i \leq n$. The internal nodes describe how the symbols are grouped. In the original wavelet tree, nodes are binary and thus there are two groups, called the `0`-group and the `1`-group, which form an alphabet partition. In the multi-ary wavelet tree, the nodes are obtained by forming more than two groups each time. We focus on binary wavelet trees in the following.

For example, consider the sequence $S =$ `SENSELESSNESS#` in Fig. 1. Here we divide the symbols in two groups $\{\mathtt{E}, \mathtt{L}\}$, and $\{\mathtt{N}, \mathtt{S}, \#\}$, giving rise to the two children of the root: the left child contains the subsequence of S obtained by copying the symbols in $\{\mathtt{E}, \mathtt{L}\}$; the right child contains the subsequence of S obtained by copying the rest of the symbols (which are in $\{\mathtt{N}, \mathtt{S}, \#\}$). The partition of $\{\mathtt{E}, \mathtt{L}\}$ into $\{\mathtt{E}\}$ and $\{\mathtt{L}\}$ produces two leaves. The partition of $\{\mathtt{N}, \mathtt{S}, \#\}$ into $\{\mathtt{N}\}$ and $\{\mathtt{S}, \#\}$ gives rise to a leaf and an internal node. The latter represents the partition of $\{\mathtt{S}, \#\}$ giving rise to two leaves.

In general each internal node of the wavelet tree represents a subsequence S' of the input sequence S, obtained by selecting certain symbols from S. More precisely, if Σ' is the alphabet for the symbols in S', then S' is formed by selecting *all* the symbols of S belonging to Σ'. Note that $\Sigma' = \{c\}$ if and only if the node storing S' is the leaf whose associated symbol is c. For an internal node, its two children are determined by the choice of the `0`-group Σ'_0 and of the `1`-group Σ'_1 partitioning $\Sigma' = \Sigma'_0 \cup \Sigma'_1$. To this end, a bitvector $B_{S'}$ is associated with S', where the `0`'s mark which positions of S' contain symbols from Σ'_0 and the `1`'s mark which positions contain symbols from `1`-group Σ'_1.

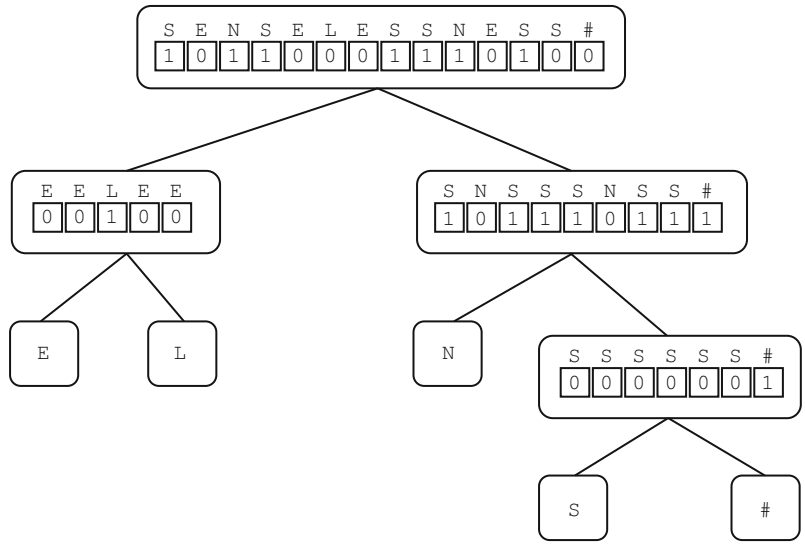
It is worth noting that any choice for the recursive alphabet partitioning in `0`- and `1`-groups can be translated into a simple dichotomy test at each node by suitably reordering the alphabet Σ: without loss of generality, we assume that given a node representing S' over alphabet Σ', there exists a symbol $c' \in \Sigma'$ such that c belongs to the `0`-group Σ'_0 if and only if $c \in \Sigma'$ and $c \leq c'$.

Finally, the sequences S and S' can be actually dropped from the nodes of the wavelet tree: just knowing the symbols from Σ associated with its leaves allows us to reconstruct the dropped sequences in its internal nodes as we discuss next.

Key Results

Despite its simplicity, the wavelet tree is a versatile data structure that offers solutions to a variety of situations using small additional space.

- *Compressed sequences*. Sequence S can be stored using a number of bits close to the 0-order entropy and still supporting random access and other operations such as rank and select of individual symbols.
- *Geometric points and 2D data*. The leaves represent the individual points in x-order, while



Wavelet Trees, Fig. 1 A wavelet tree for the sequence $S = \texttt{SENSELESSNESS\#}$ with $\Sigma = \{\texttt{E}, \texttt{L}, \texttt{N}, \texttt{S}, \#\}$. Only the symbols in the leaves and the bitvectors are actually stored

the root represents the same points but in y-order. Range and percentile queries can be performed in this way.

- *Permutations, shufflings, and reorderings*. The leaves represent the elements in a certain order and the root represents the same set in a permuted order. Mapping these two orders can be done efficiently.

As for the construction of the wavelet tree, it can be easily done in $O(n \log \sigma)$ time and space. More sophisticated algorithms have been developed to lower the construction time and/or the additional working space. In the following, we focus here on the usage of the original wavelet tree.

Compressed Sequences

The first natural question is how to `access` symbol $S[i]$ from the wavelet tree for S. We can only use the bitvectors $B_{S'}$ in the internal nodes and the mapping from the leaves to the symbols of Σ. We start out from the root, check bit $B_S[i]$, and count the number i' of bits equal to $B_S[i]$ in the first i positions of B_S. After that, we repeat the step on the left child (if $B_S[i] = \texttt{0}$) or the right child (if $B_S[i] = \texttt{1}$), setting the new value of $i = i'$ and using the bitvector $B_{S'}$ of that child. Eventually we end up in a leaf, and the symbol c corresponding to that leaf gives the answer that $S[i] = c$.

Regarding the time and space complexity, we need an operation to count how many `1`'s occur in the first i positions of a bitvector (as the number of `0`'s can be obtained by subtracting this count from i). This operation is called `rank` in the literature and takes constant time by preprocessing the bitvector and adding a little-oh number of bits to it. In this way, the cost of `access` operation is given by the height of the wavelet tree, which is $O(\log \sigma)$ in case of a balanced shape.

As for the space complexity, note that any binary tree shape with σ leaves is feasible. Using a Huffman tree shape, less frequent symbols correspond to deeper leaves. Note that each symbol occurrence in a leaf can be charged a bit from each bitvector in its ancestors. Equivalently, the sum of the lengths of the bitvectors in all the internal nodes of the wavelet tree is equal to the sum of the lengths of the Huffman encodings of the symbols of the input sequence S. In other words, the space required by the bitvectors is equal to the space achieved by the Huffman encoding. The additional `rank` data structures use little-oh of that space. Letting $H_0 \leq \log \sigma$ (logarithms in base 2) denote the 0-order entropy of S, the total space to store a wavelet tree is therefore $nH_0 + o(nH_0)$ bits, which can be lowered to $nH_0 + o(n)$ with additional machinery. For compressible sequences S, this is better than storing them in $n \log \sigma$ bits with the standard format. In general,

any prefix-free encoding of the symbols in Σ can be used in place of the Huffman coding, giving the same number of bits of the chosen encoding: it suffices to choose as a shape the resulting prefix tree of the chosen encoding of the symbols in Σ. (Note that storing the tree shape and symbol mappings requires $O(\sigma \log \sigma)$ further bits.)

Interestingly, the above space bound can be obtained using any shape if the bitvectors are stored in compressed format. For example, one compressed bitvector representation stores a bitvector of length m with k `1`'s using the theoretic information minimum of $\log \binom{m}{k} + o(m)$ bits and supports constant time `rank` and `select` operations. The latter operation returns the position of the jth `1` in the bitvector (same for the jth `0`). It can be shown that for *any shape* of the wavelet tree, summing the $\log \binom{m}{k} + o(m)$ contribution of all the bitvectors in its nodes still gives a total space of $nH_0 + o(n)$ bits to store the wavelet tree. In other words, the 0-order entropy bound can be achieved independently of the tree shape.

As a by-product of what we discussed above, the wavelet tree allows us to extend the `rank` and `select` operations from a bitvector to any sequence over an alphabet Σ. To see why, suppose we want to know how many occurrences of symbols c occur in the first i positions of S. We perform the same steps as described above for the `access` operation (where we initially set $i' = i$) except that now we already know that the path to follow is from the root to the leaf representing c. In the generic step, we can easily test if c belongs to the `0`- or `1`-group of the current node, and branch according to the target leaf, updating the value of i'. However, when we reach the leaf for c, we have to return the corresponding value of i' as the answer for `rank` of c, since it tells how many c's are up to position i. As for the `select` operation on c, suppose that we want to identify the jth occurrence of c in S. This time we proceed from the leaf corresponding to c backwards to the root. We initially set $i' = j$ and then reverse the branching process: at the generic step, we are in a node storing S' and on position i'. We reach the parent p of the current node, and select the i'th `0` (if arriving from the left child) or the i'th `1` (if arriving from the right child) in the bitvector stored in p. We set i' to be the resulting position and iterate. Eventually we reach the root and return the current value of i' as the answer for `select` of c.

Time cost is proportional to the height of the wavelet tree, which is $O(\log \sigma)$ in case of a balanced shape. Using multi-ary wavelet trees, the height can be reduced and so does the cost, achieving $O(1 + \log \sigma / \log w)$ time with a word size $w = \Omega(\log n)$.

Geometric Points and Two-Dimensional Data

Given a set of n points $\langle x_i, y_i \rangle$ in the plane, or equivalently 2D data, where $x_1 \leq x_2 \leq \cdots x_n$, we can use the wavelet tree as a space-efficient data structure for storing and querying them. We store these coordinates in two vectors X and Y, such as $X[i] = x_i$ and $Y[i] = y_i$ for $1 \leq i \leq n$. We then build the wavelet tree where $S \equiv Y$ and Σ is the set of distinct values in Y.

As a result, we obtain a compacted hierarchical space decomposition for the n points. To see why, we can conceptually think of the n points as belonging to a $n \times n$ grid stored in the root of the wavelet tree, where the actual coordinates are those stored in X and Y. The geometric interpretation of alphabet partitioning in `0`- and `1`-groups is that of choosing a value y' and splitting the points in two groups, those having coordinate $y_i \leq y'$ (the `0`-group) and those having $y_i > y'$ (the `1`-group). Let n_0 be the size of the `0`-group for the root and n_1 be the size of the `1`-group, where $n = n_0 + n_1$. In each group, only the rows and the columns that still contain points survive. As a result, two grids of size $n_0 \times n_0$ and $n_1 \times n_1$ are produced from one of size $n \times n$. Here, the left child corresponds to a subsequence Y'_0 of Y that represents the n_0 points (with $y_i \leq y'$) in the grid of size $n_0 \times n_0$, and the right child represents the sequence Y'_1 of n_1 points (with $y_i > y'$) in the grid of size $n_1 \times n_1$. The leaves of the wavelet tree are in y-order, and each of them stores the points sharing the same y-coordinate.

As for the storage, we observe that the values in X can be represented compactly as they are in

x-order. The values of Y do not need to be stored separately as they are represented in the wavelet tree. Total space is $O(n \log n)$ bits but can be less if the sequences of coordinates are compressible.

The supported operations exploit the aforementioned hierarchical space decomposition for the two-dimensional data as illustrated next. For example, consider the classical 2D range query, reporting (or counting) the points contained in the range $[a \ldots b] \times [c \ldots d]$. If the current cutting coordinate y' is outside the range $[c \ldots d]$, we move to one of the two children; otherwise, we branch on both children, using respectively the ranges $[a \ldots b] \times [c \ldots y']$ and $[a \ldots b] \times (y' \ldots d]$. Note that the restriction $[a \ldots b]$ on the x-coordinates can be used to test if the grid represented by the reached child has a nonempty intersection with the range: the mechanism is the same as that of `rank`. This means that each reported occurrence potentially requires a traversal down to a leaf; thus the cost is proportional to the number of reported points times the height of the wavelet tree. A refined version of this idea allows for $O(\log n / \log \log n)$ time for a counting query.

Another interesting use is quantile queries. For the range $[a \ldots b]$, consider the values in $V_{ab} = \{y_i \mid a \leq x_i \leq b\}$ obtained as the y-coordinates of the points in that range. For any given a, b, and k, the query asks to find the kth element in V_{ab}. Using the above wavelet tree, we can find the rank i_a of a and that i_b of b. Then, using `rank` operations on the bitvector B_S in the root, we can count how many `0`'s and `1`'s are in $B_s[i_a \ldots i_b]$. If there are at least k `0`'s, we know that the kth value in V_{ab} is smaller than or equal than the cutting coordinate y', and we iterate in the left child; otherwise, we subtract the number of `0`'s from k, and we iterate in the right child. When we reach a leaf, we return the associated value as the answer for the quantile query. Along the same lines, we can also report the topmost k values in V_{ab}. Once again, the cost is proportional to the wavelet tree height for each reported value.

Permutations, Shufflings, and Reorderings

The above discussion brings the combinatorial structure of wavelet tree to light as we can store two orders inside it: the former is the order in the sequence stored at the root, and the latter is obtained by a left-to-right traversal of the leaves. The bitvectors are internal routers that guide how elements are permuted and shuffled to produce the reordering. Mergesort can be modeled according to this view: the internal nodes merge the content of their children, and the bitvectors tell who goes where in the resulting merged reordering.

An immediate application of the above observations is setting S to be a permutation π of the integers in $\{1, 2, \ldots, n\}$. Traversing the wavelet tree upward (as in the `select`) computes $\pi(i)$, while traversing it downward computes the inverse permutation π^{-1}. The best cost is $O(\log n / \log \log n)$ time.

In general, we can store two orders using the wavelet tree, one being a permutation of the other. Inverted lists in information retrieval can store document IDs in increasing order of enumeration, but they also want to store these IDs in decreasing order of importance through some raking function. As it is clear now, these are two orders that can be simultaneously preserved inside the wavelet tree.

Applications

Looking back at previous work, some ideas behind the wavelet tree can be found in Kärkkäinen's PhD thesis and in Chazelle's functional approach to data structures for multidimensional searching. The wavelet tree in its explicit and fully functional form has been introduced by Grossi, Gupta, and Vitter to store the Burrows-Wheeler transform (BWT) for obtaining compressed text indexes (able to support fast pattern searching). Its natural application is supporting `rank` and `select` queries for the symbols of the resulting compressed BWT. Since then, many papers have explored the properties of the wavelet trees in several applications. Apart from compressed full-text indexes, researchers have employed wavelet trees in inverted lists, graphs, binary relations, numeric sequences, colored range queries, XPath queries, semi-structure data, and frequent item sets, to name a few. The wavelet

trie extends the wavelet tree to store a sequence of strings, rather than a sequence of symbols, thus allowing the supported operations to operate also on the prefixes of the strings. The wavelet matrix is a variant of a balanced wavelet tree, in which all the bitvectors on the same level are concatenated, and is particularly efficient for large alphabet size σ.

Cross-References

- Burrows-Wheeler Transform
- Compressed Suffix Array
- Huffman Coding
- Rank and Select Operations on Bit Strings
- Rank and Select Operations on Sequences
- Succinct and Compressed Data Structures for Permutations and Integer Functions

Recommended Reading

1. Barbay J, Navarro G (2013) On compressing permutations and adaptive sorting. Theor Comput Sci 513:109–123
2. Bose P, Chen EY, He M, Maheshwari A, Morin P (2012) Succinct geometric indexes supporting point location queries. ACM Trans Algorithms 8(2):10
3. Chazelle B (1988) A functional approach to data structures and its use in multidimensional searching. SIAM J Comput 17(3):427–462
4. Claude F, Navarro G, Ordóñez A (2015) The wavelet matrix: an efficient wavelet tree for large alphabets. Inf Syst 47:15–32
5. Ferragina P, Manzini G, Mäkinen V, Navarro G (2007) Compressed representations of sequences and full-text indexes. ACM Trans Algorithms 3(2):20
6. Ferragina P, Giancarlo R, Manzini G (2009) The myriad virtues of wavelet trees. Inf Comput 207(8):849–866
7. Gagie T, Puglisi SJ, Turpin A (2009) Range quantile queries: another virtue of wavelet trees. In: String processing and information retrieval, Saariselkä. Springer, pp 1–6
8. Gagie T, Navarro G, Puglisi SJ (2012) New algorithms on wavelet trees and applications to information retrieval. Theor Comput Sci 426:25–41
9. Grossi R, Ottaviano G (2012) The wavelet trie: maintaining an indexed sequence of strings in compressed space. In: Krötzsch M, Lenzerini M, Benedikt M (eds) PODS'12: proceedings of the 31st ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, Scottsdale, 20–24 May 2012. ACM, pp 203–214
10. Grossi R, Gupta A, Vitter JS (2003) High-order entropy-compressed text indexes. In: Proceedings of the fourteenth annual ACM-SIAM symposium on discrete algorithms, Baltimore. Society for Industrial and Applied Mathematics, pp 841–850
11. Grossi R, Gupta A, Vitter JS (2004) When indexing equals compression: experiments with compressing suffix arrays and applications. In: Proceedings of the fifteenth annual ACM-SIAM symposium on discrete algorithms, New Orleans. Society for Industrial and Applied Mathematics, pp 636–645
12. Kärkkäinen J (1999) Repetition-based text indexing. PhD thesis, University of Helsinki, Finland
13. Makris C (2012) Wavelet trees: a survey. Comput Sci Inf Syst 9(2):585–625
14. Navarro G (2014) Wavelet trees for all. J Discret Algorithms 25:2–20

Weighted Connected Dominating Set

Yu Wang[1], Weizhao Wang[2], and Xiang-Yang Li[3]
[1]Department of Computer Science, University of North Carolina, Charlotte, NC, USA
[2]Google Inc., Irvine, CA, USA
[3]Department of Computer Science, Illinois Institue of Technology, Chicago, IL, USA

Keywords

Minimum weighted connected dominating set

Years and Authors of Summarized Original Work

2005; Wang, Wang, Li

Problem Definition

This problem is concerned with a weighted version of the classical minimum connected dominating set problem. This problem has numerous motivations including wireless networks and distributed systems. Previous work [1, 2, 4, 5, 6, 14] in wireless networks focuses on designing efficient distributed

algorithms to construct the connected dominating set which can be used as the virtual backbone for the network. Most of the proposed methods try to minimize the number of nodes in the backbone (i.e., the number of clusterheads). However, in many applications, minimizing the size of the backbone is not sufficient. For example, in wireless networks different wireless nodes may have different costs for serving as a clusterhead, due to device differences, power capacities, and information loads to be processed. Thus, by assuming each node has a cost to being in the backbone, there is a need to study distributed algorithms for weighted backbone formation. Centralized algorithms to construct a weighted connected dominating set with minimum weight have been studied [3, 7, 9]. Recently, the work of Wang, Wang, and Li [12, 13] proposes an efficient distributed method to construct a weighted backbone with low cost. They proved that the total cost of the constructed backbone is within a small constant factor of the optimum when either the nodes' costs are smooth (i.e., the maximum ratio of costs of adjacent nodes is bounded) or the network maximum node degree is bounded. To the best knowledge of the entry authors, this work is the first to consider this weighted version of minimum connected dominating set problem and provide a distributed approximation algorithm.

Notations

A communication graph $G = (V, E)$ over a set V of wireless nodes has an edge uv between nodes u and v if and only if u and v can communicate directly with each other, i.e., inside the transmission region of each other. Let $d_G(u)$ be the degree of node u in a graph G and Δ be the maximum node degree of all wireless nodes (i.e., $\Delta = \max_{u \in V} d_G(u)$). Each wireless node u has a cost $c(u)$ of being in the backbone. Let $\delta = \max_{ij \in E} c(i)/c(j)$, where ij is the edge between nodes i and j, E is the set of communication links in the wireless network G, and the maximum operation is taken on all pairs of adjacent nodes i and j in G. In other words, δ is the maximum ratio of costs of two adjacent nodes and can be called the *cost smoothness* of the network. When δ is bounded by some small constant, the node costs are *smooth*. When the transmission region of every wireless node is modeled by a unit disk centered at itself, the communication graph is often called a *unit disk graph*, denoted by $UDG(V)$. Such networks are also called *homogeneous networks*.

A subset S of V is a *dominating set* if each node in V is either in S or is adjacent to some node in S. Nodes from S are called dominators, while nodes not in S are called dominatees. A subset B of V is a *connected dominating set* (CDS) if B is a dominating set and B induces a connected subgraph. Consequently, the nodes in B can communicate with each other without using nodes in $V - B$. A dominating set with minimum cardinality is called *minimum dominating set* (MDS). A CDS with minimum cardinality is the *minimum connected dominating set* (MCDS). In the weighted version, assume that each node u has a cost $c(u)$. Then a CDS B is called *weighted connected dominating set* (WCDS). A subset B of V is a *minimum weighted connected dominating set* (MWCDS) if B is a WCDS with minimum total cost. It is well-known that finding either the *minimum connected dominating set* or the *minimum weighted connected dominating set* is a NP-hard problem even when G is a unit disk graph. The work of Wang et al. studies efficient approximation algorithms to construct a low-cost backbone which can approximate the MWCDS problem well. For a given communication graph $G = (V, E, C)$ where V is the set of nodes, E is the edge set, and C is the set of weights for edges, the corresponding minimum weighted connected dominating set problem is as follows.

Problem 1 (Minimum Weighted Connected Dominating Set)
INPUT: The weighted communication graph $G = (V, E, C)$.
OUTPUT: A subset A of V is a *minimum weighted connected dominating set*, i.e., (1) A is a dominating set; (2) A induces a connected subgraph; (3) the total cost of A is minimum.

Another related problem is independent set problem. A subset of nodes in a graph G is an

independent set if for any pair of nodes, there is no edge between them. It is a *maximal independent set* if no more nodes can be added to it to generate a larger independent set. Clearly, any maximal independent set is a dominating set. It is a *maximum independent set* (MIS) if no other independent set has more nodes. The independence number, denoted as $\alpha(G)$, of a graph G is the size of the MIS of G. The *k-local independence number*, denoted by $\alpha^{[k]}(G)$, is defined as $\alpha^{[k]}(G) = \max_{u \in V} \alpha(G_k(u))$. Here, $G_k(u)$ is the induced graph of G on k-hop neighbors of u (denoted by $N_k(u)$), i.e., $G_k(u)$ is defined on $N_k(u)$, and contains all edges in G with both end-points in $N_k(u)$. It is well-known that for a unit disk graph, $\alpha^{[1]}(UDG) \leq 5$ [2] and $\alpha^{[2]}(UDG) \leq 18$ [11].

Key Results

Since finding the minimum weighted connected dominating set (MWCDS) is NP-hard, centralized approximation algorithms for MWCDS have been studied [3, 7, 9]. In [9], Klein and Ravi proposed an approximation algorithm for the node-weighted Steiner tree problem. Their algorithm can be generalized to compute a $O(\log \Delta)$ approximation for MWCDS. Guha and Khuller [7] also studied the approximation algorithms for node-weighted Steiner tree problem and MWCDS. They developed an algorithm for MWCDS with an approximation factor of $(1.35 + \epsilon) \log \Delta$ for any fixed $\epsilon > 0$. Recently, Ambuhl et al. [3] provided a constant approximation algorithm for MWCDS under UDG model. Their approximation ratio is bounded by 89. All these algorithms are centralized algorithms, while the applications in wireless ad hoc networks prefer distributed solutions for MWCDS.

In [12, 13], Wang et al. proposed a distributed algorithm that constructs a weighted connected dominating set for a wireless ad hoc network G. Their method has two phases: the first phase (clustering phase, Algorithm 1 in [12, 13]) is to find a set of wireless nodes as the dominators (clusterheads) and the second phase (Algorithm 2 in [12, 13]) is to find a set of nodes, called *connectors*, to connect these dominators to form the final backbone. Wang et al. proved that the total cost of the constructed backbone is no more than $\min(\alpha^{[2]}(G) \log (\Delta + 1), (\alpha^{[1]}(G) - 1)\delta + 1) + 2\alpha^{[1]}(G)$ times of the optimum solution.

Algorithm 1 first constructs a maximal independent set (MIS) using classical greedy method with the node cost as the selection criterion. For each node v in MIS, it then runs a local greedy set cover method on the *local neighborhood* $N_2(v)$ to find some nodes ($GRDY_v$) to cover all one-hop neighbors of v. If $GRDY_v$ has a total cost smaller than v, then it uses $GRDY_v$ to replace v, which further reduces the cost of MIS. The following theorem of the total cost of this selected set is proved in [12, 13].

Theorem 1 *For a network modeled by a graph G, Algorithm 1 (in* [12, 13]*) constructs a dominating set whose total cost is no more than* $\min(\alpha^{[2]}(G) \log(\Delta+1), (\alpha^{[1]}(G)-1)\delta+1)$ *times of the optimum.*

Algorithm 2 finds some *connectors* among all the dominatees to connect the dominators into a backbone (CDS). It forms a CDS by finding connectors to connect any pair of dominators u and v if they are connected in the original graph G with at most 3 hops. A distributed algorithm to build a MST then is performed on the CDS. The following theorem of the total cost of these connectors is proved in [12, 13].

Theorem 2 *The connectors selected by Algorithm 2 (in* [12, 13]*) have a total cost no more than* $2 \cdot \alpha^{[1]}(G)$ *times of the optimum for networks modeled by G.*

Combining Theorems 1 and 2, the following theorem is the main contributions of the work of Wang et al..

Theorem 3 *For any communication graph G, Algorithm 1 and Algorithm 2 construct a weighted connected dominating set whose total cost is no more than*

$$\min(\alpha^{[2]}(G)\log(\Delta+1), (\alpha^{[1]}(G)-1)\delta+1) + 2\alpha^{[1]}(G)$$

times of the optimum.

Notice that, for homogeneous wireless networks modeled by UDG, it implies that the constructed backbone has a cost no more than $\min(18\log(\Delta+1), 4\delta+1)+10$ times of the optimum. The advantage of the constructed backbone is that the total cost is small compared with the optimum when either the costs of wireless nodes are smooth, i.e., two neighboring nodes' costs differ by a small constant factor, or the maximum node degree is low.

In term of time complexity, the most time-consuming step in the proposed distributed algorithm is building the MST. In [10], Kuhn et al. gave a lower bound on the distributed time complexity of any distributed algorithm that wants to compute a minimum dominating set in a graph. Essentially, they proved that even for the unconnected and unweighted case, any distributed approximation algorithm with poly-logarithmic approximation guarantee for the problem has to have a time-complexity of at least $\Omega(\log \Delta / \log\log \Delta)$.

Applications

The proposed distributed algorithms for MWCDS can be used in ad hoc networks or distributed system to form a low-cost network backbone for communication application. The cost used as the input of the algorithms could be a *generic* cost, defined by various practical applications. It may represent the *fitness* or *priority* of each node to be a clusterhead. The lower cost means the higher priority. In practice, the cost could represent the power consumption rate of the node if a backbone with small power consumption is needed; the robustness of the node if fault-tolerant backbone is needed; or a function of its security level if a secure backbone is needed; or a combined weight function to integrate various metrics such as traffic load, signal overhead, battery level, and coverage. Therefore, by defining different costs, the proposed low-cost backbone formation algorithms can be used in various practical applications. Beside forming the backbone for routing, the weighted clustering algorithm (Algorithm 1) can also be used in other applications, such as selecting the mobile agents to perform intrusion detection in ad hoc networks [8] (to achieve more robust and power efficient agent selection), or select the rendezvous points to collect and store data in sensor networks [15] (to achieve the energy efficiency and storage balancing).

Open Problems

A number of problems related to the work of Wang, Wang, and Li [12, 13] remain open. The proposed method assumes that the nodes are almost-static in a reasonable period of time. However, in some network applications, the network could be highly dynamic (both the topology or the cost could change). Therefore, after the generation of the weighted backbone, the dynamic maintenance of the backbone is also an important issue. It is still unknown how to update the topology efficiently while preserving the approximation quality.

In [12, 13], the following assumptions on wireless network model is used: omni-directional antenna, single transmission received by all nodes within the vicinity of the transmitter. The MWCDS problem will become much more complicated if some of these assumptions are relaxed.

Experimental Results

In [12, 13], simulations on random networks are conducted to evaluate the performances of the proposed weighted backbone and several backbones built by previous methods. The simulation results confirm the theoretical results.

Cross-References

▸ Connected Dominating Set

Recommended Reading

1. Alzoubi K, Wan P-J, Frieder O (2002) New distributed algorithm for connected dominating set in wireless ad hoc networks. In: Proceedings of IEEE 35th Hawaii international conference on system sciences (HICSS-35), Hawaii, 7–10 Jan 2002
2. Alzoubi K, Li X-Y, Wang Y, Wan P-J, Frieder O (2003) Geometric spanners for wireless ad hoc networks. IEEE Trans Parallel Distrib Process 14:408–421
3. Ambuhl C, Erlebach T, Mihalak M, Nunkesser M (2006) Constant factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In: Proceedings of the 9th international workshop on approximation algorithms for combinatorial optimization problems (APPROX 2006), Barcelona, 28–30 Aug 2006. LNCS, vol 4110. Springer, Berlin/Heidelberg, pp 3–14
4. Bao L, Garcia–Aceves JJ (2003) Topology management in ad hoc networks. In: Proceedings of the 4th ACM international symposium on mobile ad hoc networking & computing, Annapolis, 1–3 June 2003. ACM Press, New York, pp 129–140
5. Chatterjee M, Das S, Turgut D (2002) WCA: a weighted clustering algorithm for mobile ad hoc networks. J Clust Comput 5:193–204
6. Das B, Bharghavan V (1997) Routing in ad-hoc networks using minimum connected dominating sets. In: Proceedings of IEEE international conference on communications (ICC'97), Montreal, 8–12 June 1997, vol 1, pp 376–380
7. Guhaa S, Khuller S (1999) Improved methods for approximating node weighted Steiner trees and connected dominating sets. Inf Comput 150:57–74
8. Kachirski O, Guha R (2002) Intrusion detection using mobile agents in wireless ad hoc networks. In: Proceedings of IEEE workshop on knowledge media networking, Kyoto, 10–12 July 2002
9. Klein P, Ravi R (1995) A nearly best-possible approximation algorithm for node-weighted Steiner trees. J Algorithms 19:104–115
10. Kuhn F, Moscibroda T, Wattenhofer R (2004) What cannot be computed locally! In: Proceedings of the 23rd ACM symposium on the principles of distributed computing (PODC), St. John's, July 2004
11. Li X-Y, Wan P-J (2005) Theoretically good distributed CDMA/OVSF code assignment for wireless ad hoc networks. In: Proceedings of 11th international computing and combinatorics conference (COCOON), Kunming, 16–19 Aug 2005
12. Wang Y, Wang W, Li X-Y (2005) Efficient distributed low-cost backbone formation for wireless networks. In: Proceedings of 6th ACM international symposium on mobile ad hoc networking and computing (MobiHoc 2005), Urbana-Champaign, 25–27 May 2005
13. Wang Y, Wang W, Li X-Y (2006) Efficient distributed low cost backbone formation for wireless networks. IEEE Trans Parallel Distrib Syst 17:681–693
14. Wu J, Li H (2001) A dominating-set-based routing scheme in ad hoc wireless networks. Spec Iss Wirel Netw Telecommun Syst J 3:63–84
15. Zheng R, He G, Gupta I, Sha L (2004) Time indexing in sensor networks. In: Proceedings of 1st IEEE international conference on mobile ad-hoc and sensor systems (MASS), Fort Lauderdale, 24–27 Oct 2004

Weighted Popular Matchings

Julián Mestre
Department of Computer Science, University of Maryland, College Park, MD, USA
School of Information Technologies, The University of Sydney, Sydney, NSW, Australia

Years and Authors of Summarized Original Work

2006; Mestre

Problem Definition

Consider the problem of matching a set of individuals X to a set of items Y where each individual has a weight and a personal preference over the items. The objective is to construct a matching M that is stable in the sense that there is no matching M' such that the weighted majority vote will choose M' over M.

More formally, a bipartite graph (X, Y, E), a weight $w(x) \in R^+$ for each individual $x \in X$, and a rank function $r : E \to \{1, \ldots, |Y|\}$ encoding the individual preferences are given. For every applicant x and items $y_1, y_2 \in Y$ say applicant x prefers y_1 over y_2 if $r(x, y_1) < r(x, y_2)$, and x is indifferent between y_1 and y_2 if $r(x, y_1) = r(x, y_2)$. The preference

lists are said to be strictly ordered if applicants are never indifferent between two items, otherwise the preference lists are said to contain ties.

Let M and M' be two matchings. An applicant x prefers M over M' if x prefers the item he/she gets in M over the item he/she gets in M'. A matching M is *more popular than* M' if the applicants that prefer M over M' outweigh those that prefer M' over M. Finally, a matching M is *weighted popular* if there is no matching M' more popular than M.

In the *weighted popular matching problem* it is necessary to determine if a given instance admits a popular matching, and if so, to produce one. In the *maximum weighted popular matching problem* it is necessary to find a popular matching of maximum cardinality, provided one exists.

Abraham et al. [2] gave the first polynomial time algorithms for the special case of these problems where the weights are uniform. Later, Mestre [8] introduced the weighted variant and developed polynomial time algorithms for it.

Key Results

Theorem 1 *The weighted popular matching and maximum weighted popular matching problems on instances with strictly ordered preferences can be solved in* $O(|X| + |E|)$ *time.*

Theorem 2 *The weighted popular matching and maximum weighted popular matching problems on instances with arbitrary preferences can be solved in* $O(\min\{k\sqrt{|X|}, |X|\}|E|)$ *time.*

Both results rely on an alternative easy-to-compute characterization of weighted popular matchings called *well-formed* matchings. It can be shown that every popular matching is well-formed. While in unweighted instances every well-formed matching is popular [2], in weighted instances there may be well-formed matchings that are not popular. These non-popular well-formed matchings can be weeded out by pruning certain bad edges that cannot be part of any popular matching. In other words, the instance can be pruned so that a matching is popular if and only if it is well-formed and is contained in the pruned instance [8].

Applications

Many real-life problems can be modeled using one-sided preferences. For example, the assignment of graduates to training positions [5], families to government-subsidized housing [10], students to projects [9], and Internet rental markets [1] such as Netflix where subscribers are assigned DVDs.

Furthermore, the weighted framework allows one to model the naturally occurring situation in which some subset of users has priority over the rest. For example, an Internet rental site may offer a "premium" subscription plan and promise priority over "regular" subscribers.

Cross-References

- Ranked Matching
- Stable Marriage

Recommended Reading

1. Abraham DJ, Chen N, Kumar V, Mirrokni V (2006) Assignment problems in rental markets. In: Proceedings of the 2nd workshop on internet and network economics, Patras, 15–17 Dec 2006
2. Abraham DJ, Irving RW, Kavitha T, Mehlhorn K (2005) Popular matchings. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms (SODA), pp 424–432
3. Abraham DJ, Kavitha T (2006) Dynamic matching markets and voting paths. In: Proceedings of the 10th Scandinavian workshop on algorithm theory (SWAT), Riga, 6–8 July 2006, pp 65–76
4. Gardenfors P (1975) Match making: assignments based on bilateral preferences. Behav Sci 20:166–173
5. Hylland A, Zeeckhauser R (1979) The efficient allocation of individuals to positions. J Polit Econ 87(2):293–314
6. Mahdian M (2006) Random popular matchings. In: Proceedings of the 7th ACM conference on electronic commerce (EC), Venice, 10–14 July 2006, pp 238–242
7. Manlove D, Sng C (2006) Popular matchings in the capacitated house allocation problem. In: Proceed-

ings of the 14th annual European symposium on algorithms (ESA), pp 492–503
8. Mestre J (2006) Weighted popular matchings. In: Proceedings of the 16th international colloquium on automata, languages, and programming (ICALP), pp 715–726
9. Proll LG (1972) A simple method of assigning projects to students. Oper Res Q 23(23):195–201
10. Yuan Y (1996) Residence exchange wanted: a stable residence exchange problem. Eur J Oper Res 90:536–546

Weighted Random Sampling

Pavlos Efraimidis[1] and Paul (Pavlos) Spirakis[2,3,4]
[1]Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece
[2]Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece
[3]Computer Science, University of Liverpool, Liverpool, UK
[4]Computer Technology Institute (CTI), Patras, Greece

Keywords

Random number generation; Sampling

Years and Authors of Summarized Original Work

2005; Efraimidis, Spirakis

Problem Definition

The problem of random sampling without replacement (RS) calls for the selection of m distinct random items out of a population of size n. If all items have the same probability to be selected, the problem is known as uniform RS. Uniform random sampling in one pass is discussed in [1, 6, 11]. Reservoir-type uniform sampling algorithms over data streams are discussed in [12]. A parallel uniform random sampling algorithm is given in [10]. In weighted random sampling (WRS) the items are weighted and the probability of each item to be selected is determined by its relative weight. WRS can be defined with the following algorithm D:

Algorithm D, a definition of WRS

Input: A population V of n weighted items
Output: A set S with a WRS of size m
1: For $k = 1$ to m do
2: Let $p_i(k) = w_i / \sum_{s_j \in V-S} w_j$ be the probability of item v_i to be selected in round k
3: Randomly select an item $v_i \in V - S$ and insert it into S
4: End-For

Problem 1 (WRS)
INPUT: A population V of n weighted items.
OUTPUT: A set S with a weighted random sample.

The most important algorithms for WRS are the Alias Method, Partial Sum Trees and the Acceptance/Rejection method (see [9] for a summary of WRS algorithms). *None of these algorithms is appropriate for one-pass WRS.* In this work, an algorithm for WRS is presented. The algorithm is simple, very flexible, and solves the WRS problem over data streams. Furthermore, the algorithm admits parallel or distributed implementation. To the best knowledge of the entry authors, this is the first algorithm for WRS over data streams and for WRS in parallel or distributed settings.

Definitions

One-pass WRS is the problem of generating a weighted random sample in one-pass over a population. If additionally the population size is initially unknown (e.g., a data streams), the random sample can be generated with *reservoir sampling* algorithms. These algorithms keep an auxiliary storage, the reservoir, with all items that are candidates for the final sample.

Notation and Assumptions

The item weights are initially unknown, strictly positive reals. The population size is n, the size of the random sample is m and the weight of item v_i is w_i. The function $random(L, H)$ generates a uniform random number in (L, H). X denotes a random variable. Infinite precision arithmetic is assumed. Unless otherwise specified, all sampling problems are without replacement. Depending on the context, WRS is used to denote a weighted random sample or the operation of weighted random sampling.

Key Results

All the results with their proofs can be found in [4].

The crux of the WRS approach of this work is given with the following **algorithm A:**

Algorithm A

Input: A population V of n weighted items
Output: A WRS of size m
1: For each $v_i \in V$, $u_i = random(0, 1)$ and $k_i = u_i^{(1/w_i)}$
2: Select the m items with the largest keys k_i as a WRS

Theorem 1 *Algorithm A generates a WRS.*

A reservoir-type adaptation of algorithm A is the following **algorithm A-Res**:

Algorithm A with a Reservoir (A-Res)

Input: A population V of n weighted items
Output: A reservoir R with a WRS of size m
1: The first m items of V are inserted into R
2: For each item $v_i \in R$: Calculate a key $k_i = u_i^{(1/w_i)}$, where $u_i = random(0, 1)$
3: Repeat Steps 4–7 for $i = m + 1, m + 2, \ldots, n$
4: The smallest key in R is the current threshold T
5: For item v_i: Calculate a key $k_i = u_i^{(1/w_i)}$, where $u_i = random(0, 1)$
6: If the key k_i is larger than T, then:
7: The item with the minimum key in R is replaced by item v_i

Algorithm A-Res performs the calculations required by algorithm A and hence by Theorem 1 A-Res generates a WRS. The number of reservoir operations for algorithm A-Res is given by the following Proposition:

Theorem 2 *If A-Res is applied on n weighted items, where the weights $w_i > 0$ are independent random variables with a common continuous distribution, then the expected number of reservoir insertions (without the initial m insertions) is:*

$$\sum_{i=m+1}^{n} P\,[\text{item } i \text{ is inserted into } S] = \sum_{i=m+1}^{n} \frac{m}{i} = O\left(m \cdot \log\left(\frac{n}{m}\right)\right).$$

Let S_w be the sum of the weights of the items that will be skipped by A-Res until a new item enters the reservoir. If T_w is the current threshold to enter the reservoir, then S_w is a continuous random variable that follows an exponential distribution. Instead of generating a key for every item, it is possible to generate random jumps that correspond to the sum S_w. Similar techniques have been applied for uniform random sampling (see for example [3]). The following algorithm A-ExpJ is an exponential jumps-type adaptation of algorithm A:

Theorem 3 *Algorithm A-ExpJ generates a WRS.*

The number of exponential jumps of A-ExpJ is given by Proposition 2. Hence algorithm A-ExpJ reduces the number of random variates that have to be generated from $O(n)$ (for A-Res) to $O(m \log(n/m))$. Since generating high-quality random variates can be a costly operation this is a significant improvement for the complexity of the sampling algorithm.

Applications

Random sampling is a fundamental problem in computer science with applications in many fields

Algorithm A with exponential jumps (A-ExpJ)

Input: A population V of n weighted items
Output: A reservoir R with a WRS of size m

1: The first m items of V are inserted into R
2: For each item $v_i \in R$: Calculate a key $k_i = u_i^{(1/w_i)}$, where $u_i = random(0, 1)$
3: The threshold T_w is the minimum key of R
4: Repeat Steps 5–10 until the population is exhausted
5: Let $r = random(0, 1)$ and $X_w = \log(r)/log(T_w)$
6: From the current item v_c skip items until item v_i, such that:
7: $w_c + w_{c+1} + \cdots + w_{i-1} < X_w \leq w_c + w_{c+1} + \cdots + w_{i-1} + w_i$
8: The item in R with the minimum key is replaced by item v_i
9: Let $t_w = T_w^{w_i}$, $r_2 = random(t_w, 1)$ and v_i's key: $k_i = r_2^{(1/w_i)}$
10: The new threshold T_w is the new minimum key of R

including databases (see [5, 9] and the references therein), data mining, and approximation algorithms and randomized algorithms [7]. Consequently, algorithm A for WRS is a general tool that can find applications in the design of randomized algorithms. For example, algorithm A can be used within approximation algorithms for the k-Median [7].

The reservoir based versions of algorithm A, A-Res and A-ExpJ, have very small requirements for auxiliary storage space (m keys organized as a heap) and during the sampling process their reservoir continuously contains a weighted random sample that is valid for the already processed data. This makes the algorithms applicable to the emerging area of algorithms for processing data streams [2, 8].

Algorithms A-Res and A-ExpJ can be used for weighted random sampling with replacement from data streams. In particular, it is possible to generate a weighted random sample with replacement of size k with A-Res or A-ExpJ, by running concurrently, in one pass, k instances of A-Res or A-ExpJ respectively. Each algorithm instance must be executed with a trivial reservoir of size 1. At the end, the union of all reservoirs is a WRS with replacement.

URL to Code

The algorithms presented in this work are easy to implement. An experimental implementation in Java can be found at: http://utopia.duth.gr/~pefraimi/projects/WRS/index.html

Cross-References

- Online Paging and Caching
- Randomization in Distributed Computing

Recommended Reading

1. Ahrens JH, Dieter U (1985) Sequential random sampling. ACM Trans Math Softw 11:157–169
2. Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. ACM Press, pp 1–16
3. Devroye L (1986) Non-uniform random variate generation. Springer, New York
4. Efraimidis P, Spirakis P (2006) Weighted random sampling with a reservoir. Inf Process Lett J 97(5):181–185
5. Jermaine C, Pol A, Arumugam S (2004) Online maintenance of very large random samples. In: SIGMOD'04: proceedings of the 2004 ACM SIGMOD international conference on management of data. ACM Press, New York, pp 299–310
6. Knuth D (1981) The art of computer programming, vol 2, 2nd edn, Seminumerical algorithms. Addison-Wesley Publishing Company, Reading
7. Lin J-H, Vitter J (1992) ϵ-approximations with minimum packing constraint violation. In: 24th ACM STOC, pp 771–782
8. Muthukrishnan S (2005) Data streams: algorithms and applications. Found Trends Theor Comput Sci 1:1–126
9. Olken F (1993) Random sampling from databases. Ph.D. thesis, Department of Computer Science, University of California, Berkeley
10. Rajan V, Ghosh R, Gupta P (1989) An efficient parallel algorithm for random sampling. Inf Process Lett 30:265–268
11. Vitter J (1984) Faster methods for random sampling. Commun ACM 27:703–718
12. Vitter J (1985) Random sampling with a reservoir. ACM Trans Math Softw 11:37–57

Well Separated Pair Decomposition

Rolf Klein
Institute for Computer Science, University of Bonn, Bonn, Germany

Keywords

Proximity algorithms for growth-restricted metrics; Unit-disk graphs

Years and Authors of Summarized Original Work

1995; Callahan, Kosaraju

Problem Definition

Well-separated pair decomposition, introduced by Callahan and Kosaraju [4], has found numerous applications in solving proximity problems for points in the Euclidean space. A pair of point sets (A, B) is *c well separated* if the distance between A and B is at least c times the diameters of both A and B. A well-separated pair decomposition of a point set consists of a set of well-separated pairs that "cover" all the pairs of distinct points, i.e., any two distinct points belong to the different sets of some pair. Callahan and Kosaraju [4] showed that for any point set in a Euclidean space and for any constant $c \geq 1$, there always exists a c-well-separated pair decomposition (c-WSPD) with linearly many pairs. This fact has been very useful for obtaining nearly linear-time algorithms for many problems, such as computing k-nearest neighbors, N-body potential fields, geometric spanners, approximate minimum spanning trees, etc. Well-separated pair decomposition has also been shown to be very useful for obtaining efficient dynamic, parallel, and external memory algorithms.

The definition of well-separated pair decomposition can be naturally extended to any metric space. However, a general metric space may not admit a well-separated pair decomposition with a subquadratic size. Indeed, even for the metric induced by the shortest path distance in a star tree with unit weight on each edge, any well-separated pair decomposition requires quadratically many pairs. This makes the well-separated pair decomposition useless for such a metric. However, it has been shown that for the unit-disk graph metric, there do exist well-separated pair decompositions with almost linear size, and therefore many proximity problems under the unit-disk graph metric can be solved efficiently.

Unit-Disk Graphs

Denote by $\mathrm{d}(\cdot,\cdot)$ the Euclidean metric. For a set of points S in the plane, the unit-disk graph $I(S) = (S, E)$ is defined to be the weighted graph where an edge $e = (p, q)$ is in the graph if $\mathrm{d}(p, q) \leq 1$, and the weight of e is $\mathrm{d}(p, q)$. Likewise, one can define the unit-ball graph for points in higher dimensions [5].

Unit-disk graphs have been used extensively to model the communication or influence between objects [9, 12] and have been studied in many different contexts [5, 10]. For an example, wireless ad hoc networks can be modeled by unit-disk graphs [8], as two wireless nodes can directly communicate with each other only if they are within a certain distance. In unsupervised learning, for a dense sampling of points from some unknown manifold, the length of the shortest path on the unit-ball graph is a good approximation of the geodesic distance on the underlying (unknown) manifold if the radius is chosen appropriately [6, 14]. By using well-separated pair decomposition, one can encode the all-pair distances approximately by a compact data structure that supports approximate distance queries in $O(1)$ time.

Metric Space

Suppose that (S, π) is a metric space where S is a set of elements and π the distance function defined on $S \times S$. For any subset $S_1 \subseteq S$, the *diameter* $D_\pi(S_1)$ (or $D(S_1)$ when π is clear from the context) of S is defined to be $\max_{s_1, s_2 \in S_1} \pi(s_1, s_2)$. The *distance* $\pi(S_1, S_2)$ between two sets $S_1, S_2 \subseteq S$ is defined to be $\min_{s_1, \in S_1, s_2 \in S_2} \pi(s_1, s_2)$.

Well-Separated Pair Decomposition

For a metric space (S, π), two nonempty subsets $S_1, S_2 \subseteq S$ are called *c well separated* if $\pi(S_1, S_2) \geq c \cdot \max(D_\pi(S_1), D_\pi(S_2))$.

Following the definition in [4], for any two sets A and B, a set of pairs $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$, where $P_i = (A_i, B_i)$, is called a *pair decomposition* of (A, B) (or of A if $A = B$) if:

- For all the i's, $A_i \subseteq A$, and $B_i \subseteq B$.
- $A_i \cap B_i = \varnothing$.
- For any two elements $a \in A$ and $b \in B$, there exists a unique i such that $a \in A_i$, and $b \in B_i$. Call (a, b) is *covered* by the pair (A_i, B_i).

If in addition, every pair in $\mathcal{P}$ is c well separated, $\mathcal{P}$ is called a *c-well-separated pair decomposition* (or c-WSPD for short). Clearly, any metric space admits a c-WSPD with quadratic size by using the trivial family that contains all the pairwise elements.

Key Results

In [7], it was shown that for the metric induced by the unit-disk graph on n points and for any constant $c \geq 1$, there does exist a c-WSPD with $O(n \log n)$ pairs, and such a decomposition can be computed in $O(n \log n)$ time. It was also shown that the bounds can be extended to higher dimensions. The following theorems state the key results for two and higher dimensions:

Theorem 1 *For any set S of n points in the plane and any $c \geq 1$, there exists a c-WSPD $\mathcal{P}$ of S under the unit-disk graph metric where $\mathcal{P}$ contains $O\left(c^4 n \log n\right)$ pairs and can be computed in $O\left(c^4 n \log n\right)$ time.*

Theorem 2 *For any set S of n points in $\mathbb{R}^k$, for $k \geq 3$, and for any constant $c \geq 1$, there exists a c-WSPD $\mathcal{P}$ of S under the unit-ball graph metric where $\mathcal{P}$ contains $O\left(n^{2-2/k}\right)$ pairs and can be constructed in $O\left(n^{4/3} \text{polylog}\, n\right)$ time for $k = 3$ and in $O\left(n^{2-2/k}\right)$ time for $k \geq 4$.*

The difficulty in obtaining a well-separated pair decomposition for the unit-disk graph metric is that two points that are close in space are not necessarily close under the graph metric. The above bounds are first shown for the point set with constant-bounded density, i.e., a point set where any unit disk covers only a constant number of points in the set. The upper bound on the number of pairs is obtained by using a packing argument similar to the one used in [1].

For a point set with unbounded density, one applies a clustering technique similar to the one used in [8] to the point set and obtains a set of "clusterheads" with a bounded density. Then the result for bounded density is applied to those clusterheads. Finally, the well-separated pair decomposition is obtained by combining the well-separated pair decomposition for the bounded density point sets and for the Euclidean metric. The number of pairs is dominated by the number of pairs constructed for a constant density set, which is in turn dominated by the bound given by the packing argument. It has been shown that the bounds on the number of pairs is tight for $k \geq 3$.

Applications

For a pair of well-separated sets, the distance between two points from different sets can be approximated by the "distance" between the two sets or the distance between any pair of points in different sets. In other words, a well-separated pair decomposition can be thought of as a compressed representation to approximate the $\Theta(n^2)$ pairwise distances. Many problems that require the pairwise distances to be checked can therefore be approximately solved by examining those distances between the well-separated pairs of sets. When the size of the well-separated pair decomposition is subquadratic, it often results in more efficient algorithms than examining all the pairwise distances. Indeed, this is the intuition behind many applications of the geometric well-separated pair decomposition. By using the same intuition, one can apply the well-separated pair decomposition in several proximity problems under the unit-disk graph metric.

Suppose that (S, d) is a metric space. Let $S_1 \subseteq S$. Consider the following natural proximity problems:

- **Furthest neighbor, diameter, center.** The furthest neighbor of $p \in S_1$ is the point in S_1 that maximizes the distance to p. Related problems include computing the *diameter*, the maximum pairwise shortest distance for points in S_1, and the *center*, the point that minimizes the maximum distance to all the other points.
- **Nearest neighbor, closest pair.** The nearest neighbor of $p \in S_1$ is the point in S_1 with the minimum distance to p. Related problems include computing the *closest pair*, the pair with the minimum shortest distance, and the *bichromatic closest pair*, the pair that minimizes the distance between points from two different sets.
- **Median.** The median of S is the point in S that minimizes the average (or total) distance to all the other points.
- **Stretch factor.** For a graph G defined on S, its stretch factor with respect to the unit-disk graph metric is defined to be the maximum ratio $\pi_G(p,q)/\pi(p,q)$, where π_G, π are the distances induced by G and by the unit-disk graph, respectively.

All the above problems can be solved or approximated efficiently for points in the Euclidean space. However, for the metric induced by a graph, even for planar graphs, very little is known besides solving the expensive all-pair shortest-path problem. For computing the diameter, there is a simple linear-time method that achieves a 2-approximation (Select an arbitrary node v and compute the shortest-path tree rooted at v. Suppose that the furthest node from v is distance D away. Then the diameter of the graph is no longer than $2D$, by triangle inequality.) and a 4/3-approximate algorithm with running time $O\left(m\sqrt{n\log n} + n^2 \log n\right)$, for a graph with n vertices and m edges, by Aingworth et al. [2].

By using the well-separated pair decomposition, Gao and Zhang [7] showed that one can obtain better approximation algorithms for the above proximity problems for the unit-disk graph metric. Specifically, one can obtain almost linear-time algorithms for computing the 2.42-approximation and $O\left(n\sqrt{n\log n}/\varepsilon^3\right)$ time algorithms for computing the $(1+\varepsilon)$-approximation for any $\varepsilon > 0$. In addition, the well-separated pair decomposition can be used to obtain an $O(n\log n/\varepsilon^4)$ space distance oracle so that any $(1+\varepsilon)$ distance query in the unit-disk graph can be answered in $O(1)$ time.

The bottleneck of the above algorithms turns out to be computing the approximation of the shortest-path distances between $O(n\log n)$ pairs. The algorithm in [7] only constructs well-separated pair decompositions without computing a good approximation of the distances. The approximation ratio and the running time are dominated by that of the approximation algorithms used to estimate the distance between each pair in the well-separated pair decomposition. Once the distance estimation has been made, the rest of the computation only takes almost linear time.

For a general graph, it is unknown whether $O(n\log n)$ pairs shortest-path distances can be computed significantly faster than all-pair shortest-path distances. For a planar graph, one can compute the $O(n\log n)$ pairs shortest-path distances in $O\left(n\sqrt{n\log n}\right)$ time by using separators with $O\left(\sqrt{n}\right)$ size [3]. This method extends to the unit-disk graph with constant-bounded density since such graphs enjoy a separator property similar to that of planar graphs [13]. As for approximation, Thorup [15] recently discovered an algorithm for planar graphs that can answer any $(1+\varepsilon)$-shortest-distance query in $O(1/\varepsilon)$ time after almost linear-time preprocessing. Unfortunately, Thorup's algorithm uses balanced shortest-path separators in planar graphs which do not obviously extend to the unit-disk graphs. On the other hand, it is known that there does exist a planar 2.42-spanner for a unit-disk graph [11]. By applying Thorup's algorithm to that planar spanner, one can compute

the 2.42-approximate shortest-path distance for $O(n \log n)$ pairs in almost linear time.

Open Problems

The most notable open problem is the gap between $\Omega(n)$ and $O(n \log n)$ on the number of pairs needed in the plane. Also, the time bound for $(1 + \varepsilon)$-approximation is still about $\tilde{O}\left(n\sqrt{n}\right)$ due to the lack of efficient methods for computing the $(1 + \varepsilon)$-approximate shortest-path distances between $O(n)$ pairs of points. Any improvement to the algorithm for that problem will immediately lead to improvement to all the $(1 + \varepsilon)$-approximate algorithms presented in this entry.

Cross-References

- Applications of Geometric Spanner Networks
- Separators in Graphs
- Sparse Graph Spanners
- Well Separated Pair Decomposition for Unit-Disk Graph

Recommended Reading

1. Agarwal P, Guibas L, Ngyuen A, Russel D, Zhang L (2004) Collision detection for deforming necklaces. Comput Geom Theory Appl 28(2):137–163
2. Aingworth D, Chekuri C, Indyk P, Motwani R (1999) Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM J Comput 28(4):1167–1181
3. Arikati SR, Chen DZ, Chew LP, Das G, Smid MHM, Zaroliagis CD (1996) Planar spanners and approximate shortest path queries among obstacles in the plane. In: Díaz J, Serna M (eds) Proceedings of the 4th annual European symposium on algorithms, Barcelona, pp 514–528
4. Callahan PB, Kosaraju SR (1995) A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. J ACM 42:67–90
5. Clark BN, Colbourn CJ, Johnson DS (1990) Unit disk graphs. Discret Math 86:165–177
6. Fischl B, Sereno M, Dale A (1999) Cortical surface-based analysis II: inflation, flattening, and a surface-based coordinate system. NeuroImage 9:195–207
7. Gao J, Zhang L (2003) Well-separated pair decomposition for the unit-disk graph metric and its applications. In: Procroceedings of the 35th ACM symposium on theory of computing (STOC'03), San Diego, pp 483–492
8. Gao J, Guibas LJ, Hershberger J, Zhang L, Zhu A (2005) Geometric spanners for routing in mobile networks. IEEE J Sel Areas Commun Wirel Ad Hoc Netw (J-SAC) 23(1):174–185
9. Hale WK (1980) Frequency assignment: theory and applications. Proc IEEE 68(12): 1497–1513
10. Hunt HB III, Marathe MV, Radhakrishnan V, Ravi SS, Rosenkrantz DJ, Stearns RE (1998) NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. J Algorithms 26(2):238–274
11. Li XY, Calinescu G, Wan PJ (2002) Distributed construction of a planar spanner and routing for ad hoc wireless networks. In: Proceedings of IEEE INFOCOM 2002, New York, 23–27 June 2002
12. Mead CA, Conway L (1980) Introduction to VLSI systems. Addison-Wesley, Reading
13. Miller GL, Teng SH, Vavasis SA (1991) An unified geometric approach to graph separators. In: Proceedings of the 32nd annual IEEE symposium on foundations of computer science, San Juan, pp 538–547
14. Tenenbaum J, de Silva V, Langford J (2000) A global geometric framework for nonlinear dimensionality reduction. Science 290:22
15. Thorup M (2004) Compact oracles for reachability and approximate distances in planar digraphs. J ACM 51(6):993–1024

Well Separated Pair Decomposition for Unit-Disk Graph

Jie Gao[1] and Li Zhang[2]
[1]Department of Computer Science, Stony Brook University, Stony Brook, NY, USA
[2]Microsoft Research, Mountain View, CA, USA

Keywords

Unit Disk Graphs; Well Separated Pair Decomposition

Years and Authors of Summarized Original Work

2003; Gao, Zhang

Problem Definition

Notations

Given a finite point set A in $\mathbb{R}^d$, its *bounding box* $R(A)$ is the d-dimensional hyperrectangle $[a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d]$ that contains A and has minimum extension in each dimension.

Two point sets A, B are said to be *well separated* with respect to a separation parameter $s > 0$ if there exist a real number $r > 0$ and two d-dimensional spheres C_A and C_B of radius r each, such that the following properties are fulfilled:

1. $C_A \cap C_B = \emptyset$
2. C_A contains the bounding box $R(A)$ of A
3. C_B contains the bounding box $R(B)$ of B
4. $|C_A C_B| \geq s \cdot r$.

Here $|C_A C_B|$ denotes the smallest Euclidean distance between two points of C_A and C_B, respectively. An example is depicted in Fig. 1. Given the bounding boxes $R(A)$, $R(B)$, it takes time only $O(d)$ to test if A and B are well separated with respect to s.

Two points of the same set, A or B, have a Euclidean distance at most $2/s$ times the distance any pair $(a, b) \in A \times B$ can have. Also, any two such pairs $(a, b), (a', b')$ differ in their distances $|a - b|, |a' - b'|$ by a factor of at most $1 + 4/s$.

Given a set S of n points in $\mathbb{R}^d$, a *well-separated pair decomposition* of S with respect to separation parameter s is a sequence $(A_1, B_1), (A_2, B_2), \ldots, (A_m, B_m)$ where

1. $A_i, B_i \subset S$, for $i = 1 \ldots m$.
2. A_i and B_i are well separated with respect to s, for $i = 1 \ldots m$.
3. For all points $a, b \in S, a \neq b$, there exists a unique index i in $1 \ldots m$ such that $a \in A_i$ and $b \in B_i$, or $b \in A_i$ and $a \in B_i$ hold.

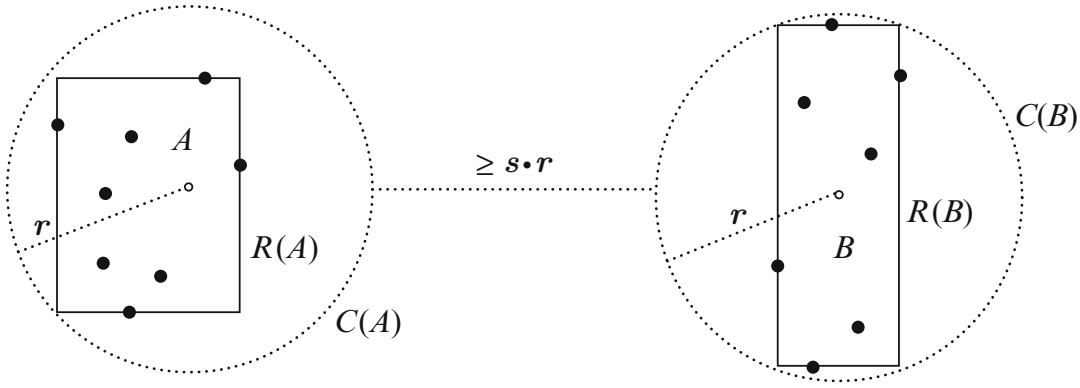
Obviously, each set $S = \{s_1, \ldots, s_n\}$ possesses a well-separated pair decomposition. One can simply use all singleton pairs $(\{s_i\},\{s_j\})$ where $i < j$. The question is if decompositions consisting of fewer than $O(n^2)$, many pairs exist and how to construct them efficiently.

Key Results

In fact, the following result has been shown by Callahan and Kosaraju [1, 2].

Theorem 1 *Given a set S of n points in $\mathbb{R}^d$ and a separation parameter s, there exists a well-separated pair decomposition of S with respect to s that consists of $O(s^d d^{d/2} n)$ many pairs (A_i, B_i). It can be constructed in time $O(dn \log n + s^d d^{d/2+1} n)$.*

Thus, if dimension d and separation parameter s are fixed – which is the case in many applications – then the number of pairs is in



Well Separated Pair Decomposition for Unit-Disk Graph, Fig. 1 The sets A, B are well-separated with respect to s

$O(n)$, and the decomposition can be computed in time $O(n \log n)$.

The main tool in constructing the well-separated pair decomposition is the split tree $T(S)$ of S. The root, r, of $T(S)$ contains the bounding box $R(S)$ of S. Its two child nodes are obtained by cutting through the middle of the longest dimension of $R(S)$, using an orthogonal hyperplane. It splits S into two subsets S_a, S_b, whose bounding boxes $R(S_a)$ and $R(S_b)$ are stored at the two children a and b of root r. This process continues until only one point of S remains in each subset. These singleton sets form the leaves of $T(S)$. Clearly, the split tree $T(S)$ contains $O(n)$ many nodes. It needs not be balanced, but it can be constructed in time $O(dn \log n)$.

A well-separated pair decomposition of S, with respect to a given separation parameter s, can now be obtained from $T(S)$ in the following way. For each internal node of $T(S)$ with children v and w, the following recursive procedure FindPairs(v, w) is called. If S_v and S_w are well separated, then the pair (S_v, S_w) is reported. Otherwise, one may assume that the longest dimension of $R(S_v)$ exceeds in length the longest dimension of $R(S_w)$ and that v_l, v_r are the child nodes of v in $T(S)$. Then, FindPairs(v_l, w) and FindPairs(v_r, w) are invoked.

The total number of procedure calls is bounded by the number of well-separated pairs reported, which can be shown to be in $O(s^d d^{d/2} n)$ by a packing argument. However, the total size of all sets A_i, B_i in the decomposition is in general quadratic in n.

Applications

From now on the dimension d is assumed to be a constant. The well-separated pair decomposition can be used in efficiently solving proximity problems for points in $\mathbb{R}^d$.

Theorem 2 *Let S be a set of n points in $\mathbb{R}^d$. Then a closest pair in S can be found in optimal time $O(n \log n)$.*

Indeed, let $q \in S$ be a nearest neighbor of $p \in S$. One can construct a well-separated pair decomposition with separation parameter $s > 2$ in time $O(n \log n)$, and let (A_i, B_i) be the pair where $p \in A_i$ and $q \in B_i$. If there were another point p' of S in A_i, one would obtain $|pp'| \leq 2/s \cdot |pq| < |pq|$, which is impossible. Hence, A_i is a singleton set. If (p, q) is a closest pair in S, then B_i must be singleton, too. Therefore, a closest pair can be found by inspecting all singleton pairs among the $O(n)$ many pairs of the well-separated pair decomposition.

With more effort, the following generalization can be shown.

Theorem 3 *Let S be a set of n points in $\mathbb{R}^d$, and let $k \leq n$. Then for each $p \in S$, its k nearest neighbors in S can be computed in total time $O(n \log n + nk)$. In particular, for each point in S can a nearest neighbor in S be computed in optimal time $O(n \log n)$.*

In dimension $d = 2$, one would typically use the Voronoi diagram for solving these problems. But as the complexity of the Voronoi diagram of n points can be as large as $n^{\lfloor d/2 \rfloor}$, the well-separated pair decomposition is much more convenient to use in higher dimensions.

A major application of the well-separated pair decomposition is the construction of good spanners for a given point set S. A spanner of S of dilation t is a geometric network N with vertex set S such that for any two vertices $p, q \in S$, the Euclidean length of a shortest path connecting p and q in N is at most t times the Euclidean distance $|pq|$.

Theorem 4 *Let S be a set of n points in $\mathbb{R}^d$, and let $t > 1$. Then a spanner of S of dilation t containing $O(s^d n)$ edges can be constructed in time $O(s^d n + n \log n)$, where $s = 4(t+1)(t-1)$.*

Indeed, if one edge (a_i, b_i) is chosen from each pair (A_i, B_i) of a well-separated pair decomposition of S with respect to s, these edges form a t-spanner of S, as can be shown by induction on the rank of each pair $(p, q) \in S^2$ in the list of all such pairs, sorted by distance.

Since spanners have many interesting applications of their own, several articles of this encyclopedia are devoted to this topic.

Open Problems

An important open question is which metric spaces admit well-separated pair decompositions. It is easy to see that the packing arguments used in the Euclidean case carry over to the case of convex distance functions in $\mathbb{R}^d$. More generally, Talwar [6] has shown how to compute well-separated pair decompositions for point sets of bounded aspect ratio in metric spaces of bounded doubling dimension.

On the other hand, for the metric induced by a disk graph in $\mathbb{R}^2$, a quadratic number of pairs may be necessary in the well-separated pair decomposition. (In a disk graph, each point $p \in S$ is center of a disk D_p of radius r_p. Two points p, q are connected by an edge if and only if $D_p \cap D_q \neq \emptyset$. The metric is defined by Euclidean shortest path length in the resulting graph. If this graph is a star with rays of identical length, a well-separated pair decomposition with respect to $s > 4$ must consist of singleton pairs.) Even for a unit disk graph, $\Omega(n^{2-2/d})$ many pairs may be necessary for points in $\mathbb{R}^d$, as Gao and Zhang [4] have shown.

Cross-References

- ▸ Applications of Geometric Spanner Networks
- ▸ Geometric Spanners

Recommended Reading

1. Callahan P (1995) Dealing with higher dimensions: the well-separated pair decomposition and its applications. Ph.D. thesis, The Johns Hopkins University
2. Callahan PB, Kosaraju SR (1995) A decomposition of multidimensional point sets with applications to k-nearest neighbors and n-body potential fields. J ACM 42(1):67–90
3. Eppstein D (1999) Spanning trees and spanners. In: Sack JR, Urrutia J (eds) Handbook of computational geometry. Elsevier, Amsterdam, pp 425–461
4. Ghao J, Zhang L (2005) Well-separated pair decomposition for the unit disk graph metric and its applications. SIAM J Comput 35(1):151–169
5. Narasimhan G, Smid M (2007) Geometric spanner networks. Cambridge University Press, New York
6. Talwar K (2004) Bypassing the embedding: approximation schemes and compact representations for low dimensional metrics. In: Proceedings of the thirty-sixth annual ACM symposium on theory of computing (STOC'04), Chicago, pp 281–290

Wire Sizing

Chris Chu
Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA

Keywords

Interconnect optimization; VLSI physical design; Wire sizing; Wire tapering

Years and Authors of Summarized Original Work

1999; Chu, Wong

Problem Definition

The problem is about minimizing the delay of an interconnect wire in a very-large-scale integration (VLSI) circuit by changing the width (i.e., sizing) of the wire. The delay of interconnect wire has become a dominant factor in determining VLSI circuit performance for advanced VLSI technology. Wire sizing has been shown to be an effective technique to minimize the interconnect delay. The work of Chu and Wong [1] shows that the wire sizing problem can be transformed into a convex quadratic program. This quadratic programming approach is very efficient and can be naturally extended to simultaneously consider buffer insertion, which is another popular interconnect delay minimization technique. Previous approaches apply either a dynamic programming approach [2], which is computationally more ex-

pensive, or an iterative greedy approach [3, 4], which is hard to combine with buffer insertion.

The wire sizing problem is formulated as follows and is illustrated in Fig. 1. Consider a wire of length L. The wire is connecting a driver with driver resistance R_D to a load with load capacitance C_L. In addition, there is a set $H = \{h_1, \ldots, h_n\}$ of n wire widths allowed by the fabrication technology. Assume $h_1 > \cdots > h_n$. The wire sizing problem is to determine the wire width function $f(x) : [0, L] \to H$ so that the delay for a signal to travel from the driver through the wire to the load is minimized.

As in most previous works on wire sizing, the work of Chu and Wong uses the Elmore delay model to compute the delay. The Elmore delay model is a delay model for RC circuits (i.e., circuits consisting of resistors and capacitors). The Elmore delay for a signal path is equal to the sum of the delays associated with all resistors along the path, where the delay associated with each resistor is equal to its resistance times its total downstream capacitance. For a wire segment of length l and width h, its resistance is $r_0 l/h$ and its capacitance is $c(h)l$, where r_0 is the wire sheet resistance and $c(h)$ is the unit length wire capacitance. $c(h)$ is an increasing function in practice. The wire segment can be modeled as a π-type RC circuit as shown in Fig. 2.

Key Results

Lemma 1 *The optimal wire width function $f(x)$ is a monotonically decreasing function.*

Lemma 1 above can be used to greatly simplify the wire sizing problem. It implies that an optimally sized wire can be divided into n segments such that the width of i-th segment is h_i. The length of each segment is to be determined. The simplified problem is illustrated in Fig. 3.

Lemma 2 *For the wire in Fig. 3, the Elmore delay is*

$$D = \frac{1}{2}\boldsymbol{l}^T \boldsymbol{\Phi} \boldsymbol{l} + \boldsymbol{\rho}^T \boldsymbol{l} + R_D C_L$$

where

$$\boldsymbol{\Phi} = \begin{pmatrix} c(h_1)r_0/h_1 & c(h_2)r_0/h_1 & c(h_3)r_0/h_1 & \cdots & c(h_n)r_0/h_1 \\ c(h_2)r_0/h_1 & c(h_2)r_0/h_2 & c(h_3)r_0/h_2 & \cdots & c(h_n)r_0/h_2 \\ c(h_3)r_0/h_1 & c(h_3)r_0/h_2 & c(h_3)r_0/h_3 & \cdots & c(h_n)r_0/h_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c(h_n)r_0/h_1 & c(h_n)r_0/h_2 & c(h_n)r_0/h_3 & \cdots & c(h_n)r_0/h_n \end{pmatrix},$$

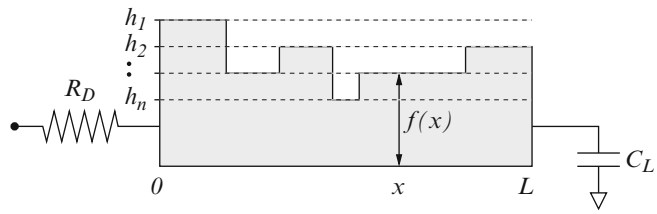
$$\boldsymbol{\rho} = \begin{pmatrix} R_D c(h_1) + C_L r_0/h_1 \\ R_D c(h_2) + C_L r_0/h_2 \\ R_D c(h_3) + C_L r_0/h_3 \\ \vdots \\ R_D c(h_n) + C_L r_0/h_n \end{pmatrix} \text{ and } \boldsymbol{l} = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \\ \vdots \\ l_n \end{pmatrix}.$$

So the wire sizing problem can be written in the following quadratic program:

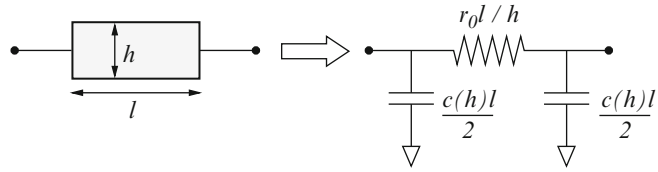
$$\begin{aligned} \mathcal{WS}: \text{minimize } & \tfrac{1}{2}\boldsymbol{l}^T \boldsymbol{\Phi} \boldsymbol{l} + \boldsymbol{\rho}^T \boldsymbol{l} \\ \text{subject to } & l_1 + \cdots + l_n = L \\ & l_i \geq 0 \text{ for } 1 \leq i \leq n \end{aligned}$$

Quadratic programming is NP-hard in general. In order to solve $\mathcal{WS}$ efficiently, some properties of the Hessian matrix $\boldsymbol{\Phi}$ are explored.

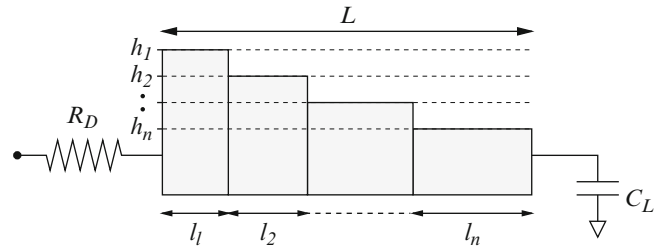
Definition 1 (Symmetric Decomposable Matrix) Let $\boldsymbol{Q} = (q_{ij})$ be an $n \times n$ symmetric matrix. If for some $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^T$ and



Wire Sizing, Fig. 1 The wire sizing problem



Wire Sizing, Fig. 2 The model of a wire segment by a π-type RC circuit



Wire Sizing, Fig. 3 The simplified wire sizing problem

$\boldsymbol{v} = (v_1, \ldots, v_n)^T$ such that $0 < \alpha_1 < \cdots < \alpha_n$, $q_{ij} = q_{ji} = \alpha_i v_i v_j$ for $i \leq j$, then $\boldsymbol{Q}$ is called a symmetric decomposable matrix. Let $\boldsymbol{Q}$ be denoted as $SDM(\boldsymbol{\alpha}, \boldsymbol{v})$.

Lemma 3 *If $\boldsymbol{Q}$ is symmetric decomposable, then $\boldsymbol{Q}$ is positive definite.*

Lemma 4 *$\boldsymbol{\Phi}$ in $\mathcal{WS}$ is symmetric decomposable.*

Lemma 3 together with Lemma 4 implies that the Hessian matrix $\boldsymbol{\Phi}$ of $\mathcal{WS}$ is positive definite. Hence, the problem $\mathcal{WS}$ is a *convex* quadratic program and is solvable in polynomial time [5].

The work of Chu and Wong proposes to solve $\mathcal{WS}$ by active set method. The active set method transforms a problem with some inequality constraints into a sequence of problems with only equality constraints. The method stops when the solution of the transformed problem satisfies both the feasibility and optimality conditions of the original problem. For the problem $\mathcal{WS}$, the active set method keeps track of an active set $\mathcal{A}$ in each iteration. The method sets $l_j = 0$ for all $j \in \mathcal{A}$ and ignores the constraints $l_j \geq 0$ for all $j \notin \mathcal{A}$. Let $\{j_1, \ldots, j_r\} = \{1, \ldots, n\} - \mathcal{A}$. Then $\mathcal{WS}$ is transformed into the following equality-constrained wire sizing problem:

$$\mathcal{ECWS}: \text{minimize } \tfrac{1}{2}\boldsymbol{l}_{\mathcal{A}}^T \boldsymbol{\Phi}_{\mathcal{A}} \boldsymbol{l}_{\mathcal{A}} + \boldsymbol{\rho}_{\mathcal{A}}^T \boldsymbol{l}_{\mathcal{A}} \quad \text{subject to } \boldsymbol{\Gamma}_{\mathcal{A}} \boldsymbol{l}_{\mathcal{A}} = L$$

where $\boldsymbol{l}_{\mathcal{A}} = (l_{j_1}, \ldots, l_{j_r})^T$, $\boldsymbol{\Gamma}_{\mathcal{A}} = (1\ 1\ \cdots\ 1)$, $\boldsymbol{\rho}_{\mathcal{A}} = (R_D c(h_{j_1}) + C_L r_0 / h_{j_1}, \ldots, R_D c(h_{j_r}) + C_L r_0 / h_{j_r})^T$, and $\boldsymbol{\Phi}_{\mathcal{A}}$ is the symmetric decomposable matrix corresponding to $\mathcal{A}$ (i.e., $\boldsymbol{\Phi}_{\mathcal{A}} = SDM(\boldsymbol{\alpha}_{\mathcal{A}}, \boldsymbol{v}_{\mathcal{A}})$ with $\boldsymbol{\alpha}_{\mathcal{A}} = \left(\frac{r_0}{c(h_{j_1}) h_{j_1}}, \ldots, \frac{r_0}{c(h_{j_r}) h_{j_r}} \right)^T$ and $\boldsymbol{v}_{\mathcal{A}} = (c(h_{j_1}), \ldots, c(h_{j_r}))^T$).

Lemma 5 *The solution of $\mathcal{ECWS}$ is*

$$\lambda_{\mathcal{A}} = -(\boldsymbol{\Gamma}_{\mathcal{A}} \boldsymbol{\Phi}_{\mathcal{A}}^{-1} \boldsymbol{\Gamma}_{\mathcal{A}}^T)^{-1} (\boldsymbol{\Gamma}_{\mathcal{A}} \boldsymbol{\Phi}_{\mathcal{A}}^{-1} \boldsymbol{\rho}_{\mathcal{A}} + L)$$
$$\boldsymbol{l}_{\mathcal{A}} = -\boldsymbol{\Phi}_{\mathcal{A}}^{-1} \boldsymbol{\Gamma}_{\mathcal{A}}^T \lambda_{\mathcal{A}} - \boldsymbol{\Phi}_{\mathcal{A}}^{-1} \boldsymbol{\rho}_{\mathcal{A}}$$

Lemma 6 *If $\boldsymbol{Q}$ is symmetric decomposable, then $\boldsymbol{Q}^{-1}$ is tridiagonal. In particular, if $\boldsymbol{Q} = SDM(\boldsymbol{\alpha}, \boldsymbol{v})$, then $\boldsymbol{Q}^{-1} = (\theta_{ij})$*

$$\textit{where} \quad \theta_{ii} = \frac{1}{(\alpha_i - \alpha_{i-1})v_i^2} + \frac{1}{(\alpha_{i+1} - \alpha_i)v_i^2},$$
$$\theta_{i,i+1} = \theta_{i+1,i} = \frac{-1}{(\alpha_{i+1} - \alpha_i)v_i v_{i+1}} \textit{ for } 1 \le i \le n-1, \; \theta_{nn} = \frac{1}{(\alpha_n - \alpha_{n-1})v_n^2}, \textit{ and } \theta_{ij} = 0 \textit{ otherwise.}$$

By Lemmas 5 and 6, $\mathcal{ECWS}$ can be solved in $O(n)$ time. To solve $\mathcal{WS}$, in practice, the active set method takes less than n iterations and hence the total runtime is $O(n^2)$. Note that unlike previous works, the runtime of this convex quadratic programming approach is independent of the wire length L.

Applications

The wire sizing technique is commonly applied to minimize the wire delay and hence to improve the performance of VLSI circuits. As there are typically millions of wires in modern VLSI circuits, and each wire may be sized many times in order to explore different architecture, logic design, and layout during the design process, it is very important for wire sizing algorithms to be very efficient.

Another popular technique for delay minimization of slow signals is to insert buffers (also called repeaters) to strengthen and accelerate the signals. The work of Chu and Wong can be naturally extended to simultaneously handle buffer insertion. It is shown in [1] that the delay minimization problem for a wire by simultaneous buffer insertion and wire sizing can also be formulated as a convex quadratic program and be solved by active set method. The runtime is only m times more than that of wire sizing, where m is the number of buffers inserted. m is typically 5 or less in practice.

About one third of all nets in a typical VLSI circuit are multi-pin nets (i.e., nets with a tree structure to deliver a signal from a source to several sinks). It is important to minimize the delay of multi-pin nets. The work of Chu and Wong can also be applied to optimize multi-pin nets. The extension is described in Mo and Chu [6]. The idea is to integrate the quadratic programming approach into a dynamic programming framework. Each branch of the net is solved as a convex quadratic program, while the overall tree structure is handled by dynamic programming.

Open Problems

After two decades of active research, the wire sizing problem by itself is now considered a well-solved problem. Some important solutions are [1–4, 6–15]. The major remaining challenge is to simultaneously apply wire sizing with other interconnect optimization techniques to improve circuit performance. Wire sizing, buffer insertion, and gate sizing are three most commonly used interconnect optimization techniques. It has been demonstrated that better performance can be achieved by applying these three techniques simultaneously rather sequentially. One very practical problem is to perform simultaneous wire sizing, buffer insertion, and gate sizing to a combinational circuit such that the total resource usage (e.g., wire/buffer/gate area, power consumption) is minimized while the delay of all input-to-output paths are less than a given target.

Cross-References

- Circuit Retiming
- Circuit Retiming: An Incremental Approach
- Gate Sizing

Recommended Reading

1. Chu CCN, Wong DF (1999) A quadratic programming approach to simultaneous buffer insertion/sizing and wire sizing. IEEE Trans Comput-Aided Des 18(6):787–798
2. Lillis J, Cheng C-K, Lin T-T (1995) Optimal and efficient buffer insertion and wire sizing. In: Proceedings of the custom integrated circuits conference, Santa Clara, pp 259–262
3. Cong J, Leung K-S (1995) Optimal wiresizing under the distributed Elmore delay model. IEEE Trans Comput-Aided Des 14(3):321–336
4. Chen C-P, Wong DF (1996) A fast algorithm for optimal wire-sizing under Elmore delay model. In: Proceedings of the IEEE ISCAS, Atlanta, vol 4, pp 412–415

5. Kozlov MK, Tarasov SP, Khachiyan LG (1979) Polynomial solvability of convex quadratic programming. Sov Math Dokl 20:1108–1111
6. Mo Y-Y, Chu C (2001) A hybrid dynamic/quadratic programming algorithm for interconnect tree optimization. IEEE Trans Comput-Aided Des 20(5):680–686
7. Sapatnekar SS (1994) RC interconnect optimization under the Elmore delay model. In: Proceedings of the ACM/IEEE design automation conference, San Diego, pp 387–391
8. Fishburn JP, Schevon CA (1995) Shaping a distributed-RC line to minimize Elmore delay. IEEE Trans Circuits Syst-I Fundam Theory Appl 42(12):1020–1022
9. Chen C-P, Chen Y-P, Wong DF (1996) Optimal wire-sizing formula under the Elmore delay model. In: Proceedings of the ACM/IEEE design automation conference, Las Vegas, pp 487–490
10. Cong J, He L (1996) Optimal wiresizing for interconnects with multiple sources. ACM Trans Des Autom Electron Syst 1(4):568–574
11. Fishburn JP (1997) Shaping a VLSI wire to minimize Elmore delay. In: Proceedings of the European design and test conference, Paris, pp 244–251
12. Chen C-P, Wong DF (1997) Optimal wire-sizing function with fringing capacitance consideration. In: Proceedings of the ACM/IEEE design automation conference, Anaheim, pp 604–607
13. Kay R, Bucheuv G, Pileggi L (1997) EWA: efficient wire-sizing algorithm. In: Proceedings of the international symposium on physical design, Napa Valley, pp 178–185
14. Chu CCN, Wong DF (1999) Greedy wire-sizing is linear time. IEEE Trans Comput-Aided Des 18(4):398–405
15. Gao Y, Wong DF (2000) Wire-sizing for delay minimization and ringing control using transmission line model. In: Proceedings of the conference on design automation and test in Europe, Paris, pp 512–516

Work-Function Algorithm for *k*-Servers

Marek Chrobak
Computer Science, University of California, Riverside, CA, USA

Keywords

Analysis; K-server problem; On-line algorithms; Work funktions

Years and Authors of Summarized Original Work

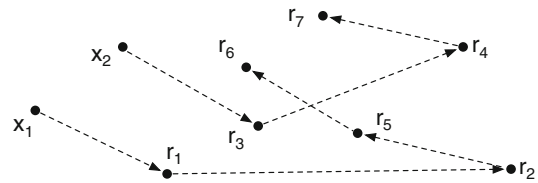
1994; Koutsoupias, Papadimitriou

Problem Definition

In the *k-Server Problem*, the task is to schedule the movement of k-servers in a metric space $\mathbb{M}$ in response to a sequence $\varrho = r_1, r_2, \ldots, r_n$ of *requests*, where $r_i \in \mathbb{M}$ for all i. The servers initially occupy some configuration $X_0 \subseteq \mathbb{M}$. After each request r_i is issued, one of the k-servers must move to r_i. A *schedule* S specifies which server moves to each request. The task is to compute a schedule with minimum *cost*, where the cost of a schedule is defined as the total distance traveled by the servers. The example below shows a schedule for 2 servers on a sequence of requests (Fig. 1).

In the offline case, if the complete request sequence ϱ is known, the optimal schedule can be computed in polynomial time [9].

Most of the research on the k-Server Problem focussed on the *online* variant, where the requests are issued one at a time. After the ith request r_i is issued, an online algorithm must decide, irrevocably, which server to move to r_i before the next request r_{i+1} is issued. It is quite easy to see that in this Online scenario it is not possible to guarantee an optimal schedule for all request sequences. The accuracy of solutions produced by



Work-Function Algorithm for *k*-Servers, Fig. 1 A schedule for 2 servers on a request sequence $\varrho = r_1, r_2, \ldots, r_7$. The initial configuration is $X_0 = \{x_1, x_2\}$. Server 1 serves r_1, r_2, r_5, r_6, while server 2 serves r_3, r_4, r_7. The cost of this schedule is $d(x_1, r_1) + d(r_1, r_2) + d(r_2, r_5) + d(r_5, r_6) + d(x_2, r_3) + d(r_3, r_4) + d(r_4, r_7)$, where $d(x, y)$ denotes the distance between points x, y

such online algorithms is often evaluated within the framework of competitive analysis. Denote by $cost_{\mathcal{A}}(\varrho)$ the cost of the schedule produced by an online k-server algorithm $\mathcal{A}$ on a request sequence ϱ, and let $opt(\varrho)$ be the cost of an optimal schedule on ϱ. $\mathcal{A}$ is called *R-competitive* if $cost_{\mathcal{A}}(\varrho) \leq R \cdot opt(\varrho) + B$, where B is a constant that may depend on $\mathbb{M}$ and X_0. The smallest such R is called the *competitive ratio* of $\mathcal{A}$. Of course, the smaller the ratio R the better.

The k-Server Problem was introduced by Manasse, McGeoch, and Sleator [14, 15], who proved that no (deterministic) online algorithm can achieve a competitive ratio smaller than k, in any metric space with at least $k+1$ points. They also gave a 2-competitive algorithm for $k = 2$ and stated what is now known as the *k-Server Conjecture*, which postulates that there exists a k-competitive online algorithm for all k. Koutsoupias and Papadimitriou [11, 12] (see also [3, 8, 10]) proved that the *Work-Function Algorithm*, presented in the next section, has competitive ratio at most $2k-1$, which to date remains the best upper bound on the competitive ratio.

Key Results

The idea of the Work-Function Algorithm is to balance two greedy strategies when a new request is issued. The first one is to simply serve the request with the closest server. The second strategy attempts to follow the optimum schedule. Roughly, from among the k possible new configurations, this strategy chooses the one where the optimum schedule would be at this time, if no more requests remained to be issued.

To formalize this idea, for each request sequence ϱ and a k-server configuration X, let $\omega_\varrho(X)$ be the minimum cost of serving ϱ under the constraint that at the end the server configuration is X. (Assume, for simplicity, that the initial configuration X_0 is fixed.) The function $\omega_\varrho(\cdot)$ is called the *work function* after the request sequence ϱ.

Algorithm WFA. Denote by σ the sequence of past requests, and suppose that the current server configuration is $S = \{s_1, s_2, \ldots, s_k\}$, where s_j is the location of the j-th server. Let r be the new request. Choose $s_j \in S$ that minimizes the quantity $\omega_{\sigma r}(S - \{s_j\} \cup \{r\}) + d(s_j, r)$, and move server j to r.

Theorem 1 ([11, 12]) *Algorithm* WFA *is* $(2k-1)$*-competitive.*

As observed in [6], Algorithm WFA can be interpreted as a primal-dual algorithm.

Applications

The k-Server Problem can be viewed as an abstraction of online problems that arise in emergency crew scheduling, caching (or paging) in two-level memory systems, scheduling of disk heads, and other. Nevertheless, in its pure abstract form, it is mostly of theoretical interest.

Algorithm WFA can be applied to some generalizations of the k-Server Problem. In particular, it is $(2n-1)$-competitive for n-state metrical task systems, matching the lower bound [3, 4, 8]. See [1, 3, 5] for other applications and extensions.

Open Problems

Theorem 1 comes tantalizingly close to settling the k-Server Conjecture described earlier in this section. In fact, it has been even conjectured that Algorithm WFA itself is k-competitive for k-servers, but the proof of this conjecture, so far, remains elusive.

For $k \geq 3$, k-competitive online k-server algorithms are known only for some restricted metric spaces, including trees, metric spaces with up to $k+2$ points, and the Manhattan plane for $k = 3$ (see [2, 7, 9, 13]). As the analysis of Algorithm WFA in the general case appears difficult, it would be of interest to prove its k-competitiveness for some natural special cases, for example in the plane (with any reasonable metric) for $k \geq 4$ servers.

Very little is known about the competitive ratio of the k-Server Problem in the randomized case. In fact, it is not even known whether a ratio better than 2 can be achieved for $k = 2$.

Cross-References

- ▸ Deterministic Searching on the Line
- ▸ Generalized Two-Server Problem
- ▸ Metrical Task Systems
- ▸ Online Paging and Caching

Recommended Reading

1. Anderson EJ, Hildrum K, Karlin AR, Rasala A, Saks M (2002) On list update and work function algorithms. Theor Comput Sci 287:393–418
2. Bein W, Chrobak M, Larmore LL (2002) The 3-server problem in the plane. Theor Comput Sci 287:387–391
3. Borodin A, El-Yaniv R (1998) Online computation and competitive analysis. Cambridge University Press, Cambridge
4. Borodin A, Linial N, Saks M (1987) An optimal online algorithm for metrical task systems. In: Proceedings of the 19th symposium on theory of computing (STOC). ACM, New York, pp 373–382
5. Burley WR (1996) Traversing layered graphs using the work function algorithm. J Algorithms 20:479–511
6. Chrobak M (2007) Competitiveness via primal-dual. SIGACT News 38:100–105
7. Chrobak M, Larmore LL (1991) An optimal online algorithm for k servers on trees. SIAM J Comput 20:144–148
8. Chrobak M, Larmore LL (1998) Metrical task systems, the server problem, and the work function algorithm. In: Fiat A, Woeginger GJ (eds) Online algorithms: the state of the art. Springer, Berlin/New York, pp 74–94
9. Chrobak M, Karloff H, Payne TH, Vishwanathan S (1991) New results on server problems. SIAM J Discret Math 4:172–181
10. Koutsoupias E (1999) Weak adversaries for the k-server problem. In: Proceedings of the 40th symposium on foundations of computer science (FOCS). IEEE, New York, pp 444–449
11. Koutsoupias E, Papadimitriou C (1994) On the k-server conjecture. In: Proceedings of the 26th symposium on theory of computing (STOC). ACM, Montreal, pp 507–511
12. Koutsoupias E, Papadimitriou C (1995) On the k-server conjecture. J ACM 42: 971–983
13. Koutsoupias E, Papadimitriou C (1996) The 2-evader problem. Inf Process Lett 57: 249–252
14. Manasse M, McGeoch LA, Sleator D (1988) Competitive algorithms for online problems. In: Proceedings of the 20th symposium on theory of computing (STOC). ACM, Chicago, pp 322–333
15. Manasse M, McGeoch LA, Sleator D (1990) Competitive algorithms for server problems. J Algorithms 11:208–230

List of Entries

M.-Y. Kao (ed.), *Encyclopedia of Algorithms*,
DOI 10.1007/978-1-4939-2864-4

This encyclopedia includes no entries for X, Y & Z